BRNO UNIVERSITY OF TECHNOLOGY

Faculty of Information Technology

PHD THESIS

Brno, 2023 Ing. Marta Jaroš

BRNO UNIVERSITY OF TECHNOLOGY
VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ

FACULTY OF INFORMATION TECHNOLOGY
FAKULTA INFORMAČNÍCH TECHNOLOGIÍ

DEPARTMENT OF COMPUTER SYSTEMS
ÚSTAV POČÍTAČOVÝCH SYSTÉMŮ

AUTOMATIC SCHEDULING, EXECUTION AND MONITORING OF COMPUTATIONAL WORKFLOWS ON DISTRIBUTED SYSTEMS

AUTOMATICKÉ PLÁNOVÁNÍ, SPOUŠTĚNÍ A MONITORING VÝPOČETNÍCH WORKFLOWS NA DISTRIBUOVANÝCH SYSTÉMECH

PHD THESIS
DISERTAČNÍ PRÁCE

AUTHOR — **Ing. MARTA JAROŠ**
AUTOR PRÁCE

SUPERVISOR — **doc. Ing. JIŘÍ JAROŠ, Ph.D.**
ŠKOLITEL

BRNO 2023

Abstract

Automated execution of computational workflows has become a critical issue in achieving high productivity in various research and development fields. Over the last few years, workflows have emerged as a significant abstraction of numerous real-world processes and phenomena, including digital twins, personalized medicine, and simulation-based science in general. Workflow execution can be viewed as an orchestration of multiple tasks with diverse computational requirements and interdependencies, determined by the workflow structure. Due to the complexity of workflows, execution can only be satisfied by remote computing clusters or clouds. As these resources are expensive, workflow scheduling plays a crucial role in the automation process.

The primary objective of this thesis is to enable automated and reliable execution of computational workflows. Moldable tasks, defined within these workflows, permit execution across multiple computational resources. This affects both the workflow makespan and computational cost, but not equally due to varying computational efficiency. Consequently, the thesis investigates various approaches to workflow scheduling and execution optimization, focusing on methods based on genetic algorithms. Three optimization approaches—targeting both on-demand and static computational resource allocations—are examined and discussed. The optimization process is supported by a performance database, which is collected on-the-fly and maintains parallel scaling of executed tasks and diverse inputs. The sparsity and incompleteness of the performance database are addressed through different interpolation methods. The proposed approaches demonstrate better utilization of computing resources while allowing prioritization of various optimization criteria, such as workflow makespan and computational cost. The final implementation was experimentally validated using real workflows executed on high-performance computing clusters at the IT4Innovations national supercomputing center.

Additionally, this thesis presents the design and development of a comprehensive system for automated workflow scheduling, execution offloading and monitoring, completed with features such as accounting, reporting, and fault tolerance. This system, named k-Dispatch, has been commercialized for the neuroscience market by Brainbox, Ltd.

Keywords

Workflows, workflows execution, workflow scheduling, genetic algorithms, multi-criteria optimization, HPC as a service, high performance computing, cloud.

Reference

JAROŠ, Marta. *Automatic scheduling, execution and monitoring of computational workflows on distributed systems.* Brno, 2023. PhD thesis. Brno University of Technology, Faculty of Information Technology. Supervisor doc. Ing. Jiří Jaroš, Ph.D.

Abstrakt

Rutinní automatizované vykonávání složitých výpočetních procesů, tzv. *workflows*, se stalo naprosto klíčovým pro dosažení vysoké produktivity v různých oblastech vědy a výzkumu. Výpočetní workflows se v posledních několika letech staly důležitou abstrakcí mnoha reálných procesů a jevů, jako např. digitálních dvojčat, personalizované medicíny či na simulaci založené vědě obecně. Vykonání workflow lze vnímat jako orchestraci mnoha úloh s různými výpočetními požadavky a vzájemnými závislostmi. Vzhledem k výpočetní složitosti reálných workflows je jejich provádění možné pouze na výpočetních klastrech nebo v cloudu, kde hraje efektivní plánování a optimalizace provedení workflows klíčovou roli.

Hlavním cílem této práce je umožnit automatizované a spolehlivé vykonání výpočetních workflows. Tyto workflows se často skládají z distribuovaných úloh, které jsou schopny běžet na několika výpočetních prostředcích najednou, dokonce umožňují toto množství měnit. Anglicky se tyto úlohy nazývají *moldable tasks*. Množství přiřazených prostředků ovlivňuje jak dobu vykonání workflow, tak i cenu výpočtu, ovšem ne stejnou měrou díky rozdílné výpočetní efektivitě. Proto tato práce zkoumá různé přístupy k plánování a optimalizaci vykonání workflows, převážně se zabývá optimalizačními technikami založenými na genetických algoritmech. Práce představuje tři optimalizační přístupy zkoumající dynamicky i staticky přidělované výpočetních zdroje. V procesu optimalizace hraje důležitou roli výkonnostní databáze, která je průběžně vytvářena a jejíž úlohou je uchovávat paralelní škálování prováděných úloh při různých vstupech. Řídkost a neúplnost výkonnostní databáze je řešena různými interpolačními metodami. Navrhované přístupy vykazují lepší využití výpočetních prostředků a umožňují prioritizaci různých optimalizačních kritérií, např. doby provádění workflow či ceny výpočtu. Finální implementace byla experimentálně ověřena na reálných workflows vykonávaných na klastrech v národním superpočítačovém centru IT4Innovations.

Tato práce rovněž představuje návrh a implementaci komplexního systému pro automatické plánování, vykonávání a monitorování workflows na výpočetních klastrech. Systém rovněž disponuje dalšími funkcemi jako jsou účtování, reportování či odolnost vůči chybám. Tento systém, zvaný *k-Dispatch*, byl úspěšně komercializován v oblasti ultrazvukové neurostimulace a je nabízen společností Brainbox, Ltd.

Klíčová slova

Workflows, spouštění workflows, plánování workflows, genetické algoritmy, vícekriteriální optimalizace, HPC jako služba, vysoce náročné počítání, cloud.

Citace

JAROŠ, Marta. *Automatické plánování, spouštění a monitoring výpočetních workflows na distribuovaných systémech.* Brno, 2023. Disertační práce. Vysoké učení technické v Brně, Fakulta informačních technologií. Školitel doc. Ing. Jiří Jaroš, Ph.D.

Rozšířený abstrakt

Personalizovaná medicína je velice slibně se rozrůstající se odvětví v oblasti péče o pacienty. Při sestavování léčby zde hraje hlavní roli pacient a jeho individuální potřeby. Personalizovaná medicína je významná zejména v oblasti onkologie, kde je zvýšený důraz na včasnou detekci, velice přesnou diagnózu a operační strategie. Vhodně zvolená léčba dokáže maximalizovat svou účinnost a minimalizovat toxicitu. Rakovina se řadí na přední příčky nemocí způsobující smrt v ekonomicky více i méně vyspělých zemích. Nezdravý životní styl přispívá rovněž velkou vahou ke vzniku této nemoci. V roce 2020 bylo po celém světě zaznamenáno přes 18 milionů nových případů rakovinových onemocnění a z toho 10 milionů úmrtí. Předpokládá se, že čísla stále porostou a rakovina se i budoucnu bude řadit mezi nemoci s nejvyšší smrtností. Biomedicínský ultrazvuku dnes stojí v čele nově používaných metod pro diagnostiku rakoviny i její následnou léčbu. V porovnání s konvenčními metodami jako je radioterapie, chemoterapie či otevřené operace, není ultrazvuk invazivní, ionizující a způsobuje méně komplikací při léčbě.

Společným jmenovatelem biomedicínských aplikací ultrazvuku je nutnost přesného, neinvazivního a bezpečného zacílení energie z ultrazvukového vysílače do lékařem specifikovaného místa v lidském těle. Aby mohla být léčba přizpůsobena každému pacientovi, je potřeba provést simulaci dané procedury a odhadnout její dopady na lidské tělo. Odhad výsledků zvolené léčebné procedury je silně závislý na výpočetně náročných simulacích šíření ultrazvuku lidským tělem, termálního a tkáňového modelu. Takové výpočty mohou být provedeny pouze pomocí center pro vysoce náročné počítání (*z angl. High-Performance Computing, HPC*). Soubor úloh, které tvoří takto náročný výpočet, nazýváme *workflow*. Formálně se jedná o orientovaný acyklický graf, zkráceně DAG (*z angl. Directed Acyclic Graph*), který definuje jednotlivé úlohy, závislosti mezi nimi a parametry spuštění.

Koncoví uživatelé, v tomto případně medicínští pracovníci, však nedisponují potřebnými znalostmi a zkušenostmi, aby mohli efektivně využívat HPC zdroje. Z tohoto důvodu jsme navrhli nástroj, který disponuje přehledným a jednoduchým rozhraním, jenž umožňuje koncovým uživatelům pohodlně spouštět a monitorovat jejich úlohy. Stěžejní vlastností tohoto nástroje je, že nepředpokládá specifikaci spouštěcí konfigurace, tj. počet výpočetních uzlů, dobu výpočtu úloh, apod., ze strany uživatele. Na druhou stranu nástroj tyto parametry optimalizuje s cílem maximalizovat propustnost, minimalizovat cenu výpočtu a celkový výpočetní čas. Optimalizací těchto parametrů je významnou oblastí, kterou se tato práce zabývá.

Práce se opírá o pět stěžejních článků autora, které jsou komentovány v textu. Cílem výzkumu bylo poskytnout automatizované a bezporuchové vykonání složitých workflows. Práce takto reaguje na dnešní velice komplexní systémy pro vysoce náročné počítání (HPC), které nemohou být využívány uživateli z akademického prostředí a průmyslu bez řádných a hlubokých znalostí dané problematiky. Vytvořením vhodného rozhraní a efektivního plánování se předpokládá (a) přiblížení HPC a nejnovějších technologií běžným uživatelům, (b) zvýšení efektivity zpracování workflows sofistikovaným plánováním spuštění beroucí v potaz aktuální vytížení HPC, (c) redukce plýtvání výpočetními zdroji, snížení celkové výpočetní doby nebo ceny díky vhodnému výběru parametrů spuštění pro jednotlivé úlohy v rámci workflows, (d) poskytnutí úrovně odolnosti vůči chybám restartováním chybných úloh s ohledem na jejich vzájemné závislosti v rámci workflows, a (e) umožnění vzniku nových metod v dané oblasti optimalizace spouštění úloh a získání praktických zkušeností díky možnosti uživatelů spolupracovat na složitých problémech.

Práce zkoumá přístupy, techniky a nástroje používané v dané oblasti optimalizace úloh a workflows a poskytování služeb HPC. Za účelem optimalizace byly navrženy a implemen-

továny speciální optimalizační moduly, které vzájemně kooperují. Tyto moduly sestávají z (1) *Optimalizátoru*, (2) *Estimátoru*, (3) *Evaluátoru* a (4) *Kolektoru*. Předmětem výzkumu pak byly moduly (1)–(3). Pro účely experimentů byly vytvořeny dvě datové sady obsahující umělá i reálná výkonnostní data. Reálná data byla naměřena s využitím softwarového balíku k-Wave na klastrech superpočítačového centra IT4Innovations. k-Wave je *state-of-the-art* nástrojem využívaným například v aplikacích pro fotoakustické snímkování prsou, transkraniální snímkování mozku, neurostimulace a plánovaní léčby využívající metodu HIFU (*z angl. High Intensity Focused Ultrasound*) při operacích nádorových onemocnění v ledvinách, játrech a prostatě. Dále byly vytvořeny workflow struktury různých velikostí inspirované reálnými medicínskými aplikacemi.

Práce se podrobně zabývá využitím genetických algoritmů pro potřeby optimalizace parametrů spouštění úloh. V rámci modulu (1) byly zkoumány tři přístupy k optimalizaci: (a) lokální optimalizace na úrovni jednotlivých úloh, (b) globální optimalizace na úrovni workflows s využitím sdílené alokace, a (c) globální optimalizace na úrovni workflows s využitím statické alokace. Pro všechny přístupy byly navrženy *fitness funkce*. Tyto funkce mohou být obecně využity jinými optimalizačními metodami. Optimalizace (b) navíc zavádí váhovací koeficient, který umožňuje najít kompromis mezi dvěma protichůdnými optimalizačními kritérii, kterými jsou celková doba výpočtu oproti ceně výpočtu. V rámci modulu (2) byly zkoumány různé metody lineární interpolace pro (a) chybějící data silného škálování v rámci jedné domény a (b) odhad silného škálování pro neznámou doménu. Pro modul (3) bylo zkoumáno využití existujících simulátorů klastru. Nakonec byl navržen a implementován vlastní simulátor s názvem *Tetrisator*, který se vyznačuje krátkou dobou běhu a je tak vhodným kandidátem pro využití v rámci evaluace fitness funkce. Navržené přístupy byly verifikovány v simulátoru a na reálném klastru IT4Innovations s využitím statické alokace.

Experimenty potvrdily, že pomocí představených optimalizačních technik lze najít takové parametry spuštění pro jednotlivé úlohy v rámci workflows, které minimalizují cenu výpočtu, celkovou dobu výpočtu nebo zlepšují propustnost. Doba nalezení parametrů spuštění je závislá na velikosti workflow, tj. z kolika úloh workflow sestává. Pro testované workflows o velikosti až 64 úloh jsme byly schopni nalézt optimální parametry spuštění pod dvě minuty s úspěšností vyšší než 90 %. Výzkum provedený v rámci této práce potvrdil, že jsme schopni navrhnout a vytvořit takové řešení, které poskytuje automatizované a bezporuchové vykonání workflows.

Realizačním výstupem této práce jsou jednak implementované optimalizační techniky, ale také nástroj *k-Dispatch*, který poskytuje HPC jako službu a provádí plánování spuštění workflows (tj. optimalizace parametrů spuštění, výběr výpočetních zdrojů atd.), jejich samotné spuštění a monitorování. Kromě toho implementuje doprovodné funkce jako je reportování a účtování. k-Dispatch je aktuálně nasazen jako součást pokročilého modelovacího nástroje *k-Plan* pro ultrazvukové plánování, uvedený na trh firmou BrainBox, Ltd.

Automatic scheduling, execution and monitoring of computational workflows on distributed systems

Declaration

Hereby, I declare that this dissertation thesis was prepared as an original author's work under the supervision of doc. Ing. Jiří Jaroš, Ph.D. All the relevant information sources, which were used during the preparation of this thesis, are properly cited and included in the list of references.

.....
Marta Jaroš
May 6, 2023

Acknowledgements

Firstly, I would like to express my deepest gratitude towards my supervisor and husband, Jiri Jaros, for providing invaluable recommendations, research ideas, and hints, for being patient with me during sleepless nights when I had to catch publication deadlines, and for his unwavering support throughout my journey.

I would also like to extend my heartfelt thanks to my colleague from the University College London, Bradley E. Treeby, for his wise advice and support. The head of the Department of Computer Systems, Faculty of Information Technology, Brno University of Technology, Lukas Sekanina, deserves a special mention for his kindness, helpfulness, and guidance.

My colleagues from the Department of Computer Systems have been a great source of support and inspiration. I am especially grateful to Honza Nevoral, Michal Bidlo, Marie Gadorkova, and Vasek Simek, for their unwavering support and encouragement.

My father, Pavel Cuda, has always been my pillar of strength, inspiring me with his extensive knowledge and serving as a debugging duck on numerous occasions. Without his encouragement, I would never have started my PhD studies. I would also like to express my gratitude to other members of my family and my friends for their unwavering emotional support, inspiration, and for bringing joy to my life.

Being a PhD student can be challenging, but I was fortunate enough to be a member of the Girls Gang@FIT that provided stress-relieving regular gatherings. Gabriela Necasova, Janka Puterova, and Dominika Regeciova, I cannot thank you enough for your support and encouragement.

I would not finish writing this thesis without meeting Gabriela and Ondrej Olsak every week on so-called Roasting Days. Thank you for mood-boosting, encouragement, reading my thesis and your help.

Lastly, I must thank my little Welsh Corgi Pembroke, Baddy. Thank you for your contagious smiles, for listening to me during our long walks in all types of weather, and for waking up by my side while stealing my bed and pillow.

This work was supported by numerous research grants and organizations, namely: (1) The Ministry of Education, Youth and Sports of the Czech Republic from the National Programme of Sustainability (NPU II), (2) project IT4Innovations excellence in science - LQ1602, (3) the BUT internal grants FIT-S-17-3994, FIT-S-20-6309, FIT/FSI-J-21-7435 and FIT/FSI-J-22-7980. This project also received funding from the European Union's Horizon 2020 research and innovation programme H2020 ICT 2016-2017 under grant agreement No 732411 and is an initiative of the Photonics Public Private Partnership, and from the European Unions Horizon Europe research and innovation programme under grant agreement No 101071008.

Contents

Chapter 1

Introduction

1.1 Motivation

Personalised medicine is an emerging approach to patient care in which an individual's characteristics guide clinical decisions aiming for the right treatment for the right patient at the right time [30]. Personalised medicine is particularly important in oncology, where increased emphasis is placed on early detection, accurate diagnosis, and precise surgical strategies. Appropriate selection of treatment for patients, to maximise efficacy and minimise toxicity, has long been a fundamental part of routine clinical practice, but until recently, clinicians had limited tools to determine benefits and potential threats.

Cancer is a leading cause of death in both more and less economically developed countries; the burden is expected to grow worldwide due to the growth and ageing of the population, particularly in less developed countries. The adoption of lifestyle behaviours that are known to increase cancer risks, such as smoking, poor diet, physical inactivity, and reproductive changes, have further increased the cancer burden. About 18.1 million new cancer cases and 10 million deaths occurred in 2020 worldwide, and this number is continuing to grow [64]. According to the Czech Society of Oncology,[1] more than 73,000 tumours diseases are newly diagnosed in the Czech Republic every year. Then, almost 27,000 patients die of the disease annually.

The applications of biomedical ultrasound sit at the heart of rapidly emerging cancer diagnosis and treatment procedures. When compared to conventional cancer diagnosis and treatment modalities, such as biopsy, open surgery, radio- and chemo-therapy, ultrasound is non-invasive, non-ionising, and causes fewer complications after treatment. Traditional treatment procedures suffer from severe limitations and side effects (radiation and drug dosage limits, operability, repeatability, long-lasting consequences) that reduce patients' chances of a successful cure. Moreover, most people need to get a combination of these treatments. These methods carry significant morbidity and mortality and may be associated with long in-patient stays and recovery periods. [74]

Non-invasive treatment procedures use high-intensity focused ultrasound (HIFU). HIFU is a medical procedure that uses high-frequency sound waves to heat and destroy targeted tissue without damaging surrounding tissue in the patient's body. Over 250,000 patients throughout the world have been treated using HIFU surgeries with great success.[2] The number of patients being screened by ultrasound is countless. The application of ultra-

[1] https://www.linkos.cz/english-summary/czech-society-for-oncology/

[2] https://www.fusfoundation.org/content/images/pdf/FUSF_State_of_the_Field_Report_2019.pdf

sound in clinics can be used in diagnosis and therapeutic procedures as well. Examples of ultrasound applications are non-invasive cancer tissue ablation [1, 65, 24], targeted drug delivery [51] significantly reducing the drug dosage, diagnostic imaging [38], and neurostimulation (treatment of epilepsy, Alzheimer's or Parkinson's disease) [10].

Ultrasound is relatively inexpensive compared to other modes of investigation, such as computed X-ray tomography or magnetic resonance imaging (MRI). It has no known long-term side effects and rarely causes any discomfort to the patient [46]. It images muscle, soft tissue, and bone surfaces very well and is particularly useful for delineating the interfaces between solid and fluid-filled spaces. Therefore, it can show the structure of organs. On the other hand, this method has problems with penetration in bones (skull, ribs, ...) and calcifications (in the kidney, upper urinary tract, ...). The depth of the ultrasound penetration may be limited by the frequency of imaging as well as the patient's body structure (obese patients). For the whole summary see [2, 13, 49].

In order to adapt therapeutic ultrasound procedures to the patient's needs, complex physical models have to be evaluated prior to the treatment to tailor treatment parameters and estimate the treatment outcome. These models can also be used during the treatment to monitor the procedure progress and after the treatment to evaluate the treatment outcome and predict further disease development.

One physical model widely used in the international community is the open-source k-Wave toolbox designed for the time-domain simulation of acoustic waves propagating in tissues in 1, 2, or 3 dimensions [67]. The toolbox has a wide range of functionality, but at its heart is an advanced numerical model that can account for both linear and nonlinear wave propagation, an arbitrary distribution of heterogeneous material parameters, power law acoustic absorption (thermal model), and the heating induced in tissue (tissue model).

Over the last decade, k-Wave has attracted a lot of interest amongst biomedical physicists, ultrasonographers, neurologists, oncologists and other clinicians. Numerous applications of k-Wave have been reported, including in photoacoustic breast screening [38], transcranial brain imaging [43], or small animal imaging [50]. k-Wave has also been used for exciting applications of HIFU, including treatment planning of kidney [1, 65], liver [24] and prostate tumour ablations [66], ultrasound neurosurgery and targeted drug delivery [51], and neurostimulation [10].

Since the model computations are very complex and intensive, they generally cannot be performed using desktop computers or small servers. Thus, it is essential to offload the computational work to the cloud or High-Performance Computing (HPC) clusters. Each application can be computationally described as a set of either independent or mutually dependent tasks. A set of such tasks reflecting a real-world phenomenon is called a *workflow* implemented as a Task Graph [53]. Workflows may consist of several levels of tasks with differing computational and memory demands. The structure of a workflow also reveals concurrency and mutual task dependencies.

Unfortunately, the composition of the processing workflow is complex and requires advanced knowledge. Moreover, running workflows in the cloud or HPC clusters represents quite a big administrative overhead every day since advanced knowledge in computer science is required. These computing facilities differ in architecture, libraries, tools and policies implemented. Thus, the industry and medical practice show there is a great demand for a middle-ware standing between clouds/HPCs and user applications and performing automatic tasks and workflows scheduling and execution.

Over the last decades, *-as-a-service [52] solutions, acting as a middle-ware layer, have become very popular. The history of this concept has its roots in the early days of computing

when mainframe computers were shared among users in large organizations. It allowed them to access computing resources on-demand and pay for only what they used. A big boom then came with huge companies like Google, Amazon Web Services, Apple and Microsoft, that offered their computing infrastructures, software, cloud, and streaming solutions as a service. The main advantages of *-as-a-service solutions are that (1) they eliminate the need to invest in hardware, software, and infrastructure, (2) are easily scalable and efficiently respond to changes in demands, (3) internet connection provides huge flexibility in accessing data and applications, and (4) faster deployment and security features.

1.2 Thesis Outline

This Thesis focuses on the computational aspects of complex biomedical workflows, with a particular emphasis on cluster management systems and workflow execution planning. The ultimate goal is to apply multi-objective optimization techniques to enable safe and automated execution of workflows in large, heterogeneous environments.

The Thesis comprises a collection of selected papers by the author and is organized as follows. This chapter introduced the research area, revealed real-world applications, and provided the motivation for research and development. Chapter 2 surveys job scheduling and execution heuristics in heterogeneous environments, including cloud-based tools and High-Performance Computing (HPC) facilities, as well as the automation of executions. Chapter 3 lists the challenges in the research area and defines the aims and objectives of this Thesis. Chapter 4 provides a high-level overview of the designed system for workflow execution planning, while Chapter 5 details its implementation and presents experimental results. Chapter 6 introduces selected research papers and summarizes their findings. Additionally, this chapter includes a list of other publications, research projects, fellowships, and awards. Finally, Chapter 7 concludes by summarizing the challenges, aims, and outcomes of this thesis, highlighting the deployment of software used for planning neurostimulation procedures, and suggesting future research directions.

Chapter 2

State of the Art

This chapter brings fundamental definitions and a research overview of areas involved in this Thesis. Attention is particularly pointed to (1) Workflow definition and description, (2) Processing frameworks and workflow management systems, (3) Workflow execution planning and scheduling, and (4) Cluster simulators. Various paradigms, their history, state-of-the-art tools, their advantages and disadvantages, and challenges are discussed.

To prevent misunderstanding, terms *job* and *task* are defined and differentiated here. Both terms may be considered interchangeable since both specify what shall be executed and how. However, a task decidedly reveals dependencies and the position in a task graph. Tasks are used in an abstract way but may hold some execution details. On the other hand, a *job* rather serves as a resource request submitted to the particular remote computing machine and its batch scheduler. Jobs may or may not reveal dependencies since this may be a subject of submission command. Jobs are often defined using machine-specific shell scripts, so-called a *job (submission) script*.

2.1 Workflow Definition

Workflows represent a standardized way to define processes and activities and may be used in a variety of contexts. A well-designed workflow can help improve productivity, reduce errors, and ensure that projects are completed on time and within budget. Effective workflows are designed to be reliable and easy to follow. They often involve the use of specialized tools and software, such as workflow management systems, or automation software. [6]

Workflow structure is defined as a *Directed Acyclic Graph (DAG)*, also referred to as a *Task Graph* [53]. DAG G is defined as $G = (V, E)$ where V is a set of v nodes in the graph (representing tasks), and E is a set of e edges (representing data flow, dependency relationship, and constraints).

There are multiple notations and languages using which workflows can be described and shared amongst users and tools. We can also distinguish between graphical and text description approaches. Examples of tools providing a graphical interface are, e.g., Kepler [37] and Taverna [70]. Such tools may also implement procedures that transform the graphical representation into a text form. Contrary, many tools rely on scripting languages such as Python or R to define workflows. The selection of the notation or language depends on the specific needs of the organization or project as well as the tools and systems being used. [23]

Graphical format transforms DAGs into a diagram such as Unified Modelling Language (UML) diagram or a flowchart. UML is a general-purpose modelling language providing

a standardized notation for creating visual representations. Workflow diagrams are very powerful in expressing the aspect of process models. In a very small space, they convey tasks, sequencing, decisions, participation, and information. [36]

Business Process Model and Notation (BPMN)[1] is a graphical language used to describe business processes and workflows, and provide a standardized notation for creating visual representations of processes, including the flow of tasks, decisions, and information. Workflow Management Coalition (WfMC) Standard[2] is a set of standards providing a common framework for workflow modelling and execution. It includes standards for workflow definition, workflow execution, and workflow interchange.

JSON and YAML can be used to represent workflows, their tasks, inputs and outputs. But they rather make a base for different languages and notations. Distributed Analytics eXtensions (DAX) is an XML-based description of DAGs. DAX is used by Pegasus to define workflows including a number of built-in data processing tasks, such as file transfers, data transformations, and executable commands. [14]

A popular programming language for describing and automating scientific workflows is Workflow Description Language (WDL)[3] that is portable across different computing environments. Common Workflow Language (CWL)[4] is a JSON-based open standard for describing workflows and tools.

2.2 Processing Frameworks and Workflow Management Systems

This section discusses the most commonly used processing frameworks and workflow management system that have been developed and actively used during last decade.

2.2.1 Processing Frameworks

The purpose of processing frameworks is to simplify the development process, reduce development time, and increase the quality and maintainability of the developed software.

Widely used data processing frameworks, especially for big data analytics, include Hadoop [59], a MapReduce-based system for parallel data processing, Apache Spark [73], a system for concurrent processing of heterogeneous data streams, Apache Storm,[5] for real-time streaming data processing, and HTCondor,[6] for managing compute-intensive jobs. These tools, however, do not allow intertask dependencies to be specified. Sometimes, such frameworks are implemented within more general workflow management systems (for example HTCondor/DAGMan[7] and Pegasus [14]) to schedule and offload the tasks.

2.2.2 Scientific Workflow Management Systems

Workflow management systems (WMSs) first emerged in the 1990s as a key technology for supporting the coordination of complex scientific and business processes. These systems were designed to automate the execution of workflows and to address a phenomenon called

[1] https://www.bpmn.org/
[2] https://wfmc.org/
[3] https://openwdl.org/
[4] https://www.commonwl.org/
[5] https://storm.apache.org/
[6] https://research.cs.wisc.edu/htcondor/
[7] https://research.cs.wisc.edu/htcondor/

workflow decay. Workflow decay [40] refers to the poor reproducibility of workflows designed to solve complex scientific problems and accelerate scientific progress. However, users, i.e., scientists in this case, often find it difficult to reuse others' workflows. [61]

Over the last more than a decade, there have been developed manifold middle-ware projects focusing on running computational tasks on high performance and cloud resources to automate and accelerate scientific progress. For example, Globus [19] or gLite[8] rank amongst grid [20] frameworks. They serve scientists to share computing power, databases, tools, and screen resource management out of users. Workflow tools offer a formal way to define, automate, and repeat multi-step computational procedures. Such tools usually provide services for resource monitoring and management, security and file management. Workflows supported by those tools are usually defined as DAGs and serve to specify the tasks that have to be performed during a specific in-silico experiment. For instance, as a reaction to this problem, a framework for facilitating the reproducibility of scientific workflows at the task level by integrating execution environment specifications into scientific workflow systems was proposed in [41]. Scientists were given complete control over the execution environments of the tasks in their workflows and integration of the execution environment specifications into scientific workflow systems. [41] However, a regular researcher might not have enough background knowledge to configure and tune the system to their needs. Insufficient computing background (not the main focus of scientists' work) presents a strong barrier to the adoption and distribution of scientific applications. Therefore, technical details of workflow execution should be delegated to the workflow and resource management system, in such cases.

For example, Taverna [70] brings together a range of features to make it easier for users to find, design and execute complex workflows and share them with other people using a drag-and-drop interface. Therefore, Taverna integrates myExperiment[9] and BioCatalogue [5] and creates an interface to work with these tools. Taverna workflows are built using the Taverna Workflow Language (TWL) and can be executed on various environments, i.e., local desktops (using Taverna Workbench[10]), on Taverna servers, clouds (for example, on Amazon Cloud) and grids, using its own Workflow Management System. Taverna has a huge domain of usage but focuses on bioinformatics and data-intensive science, and supports a wide range of web services, tools and databases.

Similarly to Taverna, Kepler Workbench [37] allows computations over local desktop computers, grid and cloud computing facilities. The user graphical interface helps users to perform complex simulation workflows. Kepler allows building workflows using the Kepler Scientific Workflow System. It offers a library of components, called actors, which are used to build workflows by connecting them through data and control flow. Kepler has a focus on environmental research and ecological science. Both Taverna and Kepler focus on researchers and experienced users.

Probably the most advanced WMS developed during the last two decades is Pegasus [14]. Pegasus executes large-scale scientific workflows and data-intensive applications on grids, and in clouds. Users can create their own workflows using Python API or YAML, and provide input files together with executables using external catalogues, e.g., replica, transformation, and site catalogues. Pegasus also provides a combination of different heuristics and mechanisms for optimizing resource allocation. Some of the commonly used heuristics in Pegasus include backfilling, priority-based scheduling and job clustering packing a bunch

[8] https://glite.web.cern.ch/glite/

[9] https://www.myexperiment.org/home

[10] http://www.taverna.org.uk/download/workbench/

of tiny jobs or such jobs sharing the same feature. Users may adjust and customize these heuristics to meet the specific requirements of different workflows and environments.

FabSim [25] shares functionality with mentioned middleware toolkits such as Globus or gLite. However, FabSim aims at experienced computational scientists and does not provide decision-making in terms of planning and monitoring. Its main goal is to simplify researchers' daily tasks. The only supported interface is a command line which is easy to extend for developers. The key strength of FabSim is its focus on simplifying and accelerating development activities. It simplifies the execution of previously defined workflows as well as creation of the new ones.

Many WMSs focused only on short-running tasks, i.e., single-core applications, usually terminating within a second. For such tasks, it is typical that the time needed for resource allocation creates a significant scheduling overhead. Examples of WMSs addressing this problem include Dask[11] and HyperLoom [12]. These tools usually implement their own scheduling mechanism and heuristics.

Dask handles short-running tasks and can reduce file system usage. However, it does not support native pipelining of third-party applications. Dask offers both high-level (e.g., NumPy objects) and low-level programming user interfaces.

HyperLoom [12] is a platform for defining and executing scientific workflows in large-scale HPC systems. Its goal is to minimize the overall workflow execution time respecting tasks' and the environment's resource constraints. HyperLoom implements an optimized dynamic scheduler that schedules the tasks reactively with a low overhead since the execution time of individual tasks is not known in advance. Moreover, the scheduler respects task dependencies and prioritizes placements that induce the smallest possible inter-node data transfers. Data produced by tasks are kept directly in memory and can be accessed by any other task without additional overhead. HyperLoom allows the chaining and execution of third-party applications. HyperLoom enables users to define and execute workflows using its client application. Although HyperLoom was originally designed to be used within HPC infrastructures, these infrastructures may be unavailable or too expensive especially for small to medium workloads. Therefore, HyperLoom developers started to aim at public cloud providers since the performance of their machines is comparable to those in HPC systems. However, network solutions used in HPC systems offer incomparably higher inter-node throughput. HyperLoom focuses on experienced users.

Many tools classified as HPC as a Service tools, were created and have been used by supercomputing centres to satisfy automated executions of particular classes of tasks. Their robustness and ability to support different HPCs and their schedulers differ. Many of them stand mainly on the programming interface or a command line. They differ in license and some of them require having an active account at the computing facility or a third-party company and having admin privileges for the tool installation. Examples of tools under an active development are: a lightweight Python-based workflow manager Autosubmit[12] used at Barcelona Supercomputing Centre (BSC)[13] in weather and climate research, unifying the access to computational resources with different batch schedulers, and providing job reports; Java and Python-based UNICORE (Uniform Interface to Computing Resources)[14] middleware developed in Juelich supercomputing centre providing unified access to different computational resources, and RESTful API for client integration;

[11] https://www.dask.org/
[12] https://autosubmit.readthedocs.io/en/master/index.html
[13] https://www.bsc.es/
[14] https://www.fz-juelich.de/en/ias/jsc/services/user-support/jsc-software-tools/unicore

FirecREST[15] maintained by the Swiss National Supercomputing Centre and focusing on Slurm only; SuperFacility[16] enabling automated use of all NERSC computational resources encapsulating services like data transfers, data sharing and data discovery; and HEAppE[17] developed by IT4Innovations supercomputing centre, enabling simple and intuitive access to the supercomputing infrastructure and providing client-server interface to control user computations.

2.3 Workflow Execution Planning and Scheduling

Workflow execution planning dates back to the 1970s when early computer systems started to be deployed for scientific and industrial use. By that time, researchers began to recognize the need for automated tools to manage and coordinate the execution of complex tasks, especially in high-performance computing environments. [68]

In the 2000s, with the advent of cloud computing and big data, the demand for efficient workflow execution planning increased significantly. This led to the development of new algorithms and techniques for dynamic scheduling of workflows in large-scale distributed systems, such as grids and clusters. [72]
The concept of HPC as a Service (HPCaaS) was introduced to bring traditional HPC technologies to the era of cloud computing while aiming to easier access to computing facilities and their applications [28].

Together with rapidly growing HPCaaS, there are also different service approaches in HPC and Cloud. The main service approaches according to the National Institute of Standards and Technology (NIST) are: (1) Software as a Service (SaaS) being a model of software delivery where the software application is hosted by a third-party provider and made available to users over the internet, (2) Platform as a Service (PaaS) allowing a user to develop or deploy applications using tools and infrastructure provided by the HPC or cloud service provider, and (3) Infrastructure as a Service (IaaS) providing a complete virtualized infrastructure, including computing resources (e.g., virtual machines, containers), storage, and networking, which users can access and use as needed. [52]

Today, workflow execution planning and scheduling is a widely researched and widely used technology with many organizations relying on it to support their business processes. Research in this area continues to focus on improving the efficiency, scalability, and reliability of workflow execution, as well as developing new techniques for integrating workflows with other technologies, such as machine learning and artificial intelligence.

Effective workflow scheduling is the key issue in computing environments. A workflow modelled as a DAG consists of several tasks that are scheduled for execution. Nevertheless, these tasks differ in execution time, resource demands, dependencies, and software licenses. The task scheduling strongly affects the efficiency and throughput of the HPC facility, as well as the workflow *makespan*, which is defined as the total wall-clock time the workflow stays in the system from the submission of the very first task to the completion of the last one, including the queuing times. The makespan is very crucial for applications that are meant to be used on a daily basis or require a guarantee to be completed within a specified time frame.

The problem of workflow scheduling along with assigning the optimal amount of compute resources to particular tasks is known to be NP-hard [15]. Even more, the task scheduling

[15] https://user.cscs.ch/tools/firecrest/
[16] https://docs.nersc.gov/services/sfapi/
[17] https://heappe.eu/web/

problems aiming for the smallest parallel execution time have been shown to be NP-complete in the strong sense, even for an unbounded number of processors [56].

2.3.1 Workload Managers and Batch Schedulers

In this Thesis, the terms *Workload Manager* and *Batch (Job) Scheduler* are related and both interchangeable. Generally, a *workload manager* is a broader term that refers to a piece of software to manage and schedule executions of jobs in a computing environment. A *batch scheduler* is a specific type of workload manager that is designed for scheduling batch jobs. The typical feature of such jobs is they are non-interactive and long-running. Batch schedulers are often used in HPC environments to manage large-scale computational workloads.

Supercomputing facilities employ commercial or open-source workload managers and batch schedulers that contain job scheduling algorithms such as backfilling, First Come First Served (FCFS), etc. For instance, Portable Batch System (PBS)[18,19] uses a backfilling scheduling algorithm considering user and group priorities, and fair-share cluster policy.[20]

The IT4Innovations[21] supercomputing centre's PBS scheduler first assigns each job an execution priority, which is consequently used to select which job(s) to run. Job execution priority is determined by the queue priority, fair-share priority and eligible time. Queue priority has the biggest impact on job execution priority. Fair-share priority is calculated on the recent usage of resources (projects with higher recent usage have lower fair-share priority). This priority is calculated per project, therefore, all members of the project share the same fair-share priority. The eligible time has the least impact on execution priority. Eligible time is an amount (in seconds) of eligible time a job accrued while waiting to run. Therefore, jobs with higher eligible time gain higher priority. IT4Innovations' clusters, *Barbora* and *Karolina*, use job backfilling scheduling algorithm. Therefore, it is very beneficial to precisely specify the allowed job wall time before submission to enable better scheduling and better resource utilization. *Backfilling* is an FCFS technique improved by increasing the utilization of the system resources and by decreasing the average waiting time in the queue. Backfilling fits smaller jobs in front of the higher-priority jobs if it is possible, in such a way that the higher-priority jobs are not delayed. This allows keeping resources from becoming idle when the top job (job with the highest execution priority) cannot run. [62]

Another widely employed workload manager is Slurm (Simple Linux Utility for Resource Management).[22] Best supercomputers from the top of Top 500[23] chart such as American Frontier, Japanese Fugaku or Finnish Lumi employ it. Slurm is highly scalable and performs always a best-fit algorithm based on a Hilbert curve scheduling or a fat tree network topology in order to determine the best match between the resources required by a job and the available resources in the cluster. By using a best-fit algorithm, Slurm is able to allocate resources in a way that balances the needs of multiple jobs and maximizes the overall utilization of the cluster. Next, Slurm implements a couple of heuristics to improve the performance of parallel computing clusters, e.g., backfilling, fair-share scheduling or topology scheduling. [47]

[18] https://www.openpbs.org/
[19] https://altair.com/pbs-professional
[20] https://www.nas.nasa.gov/hecc/support/kb/how-pbs-schedules-jobs_179.html
[21] https://docs.it4i.cz/
[22] https://slurm.schedmd.com/documentation.html
[23] https://www.top500.org/lists/top500/2022/11/

However, as mentioned before, developers of workflow management systems sometimes implement their own schedulers, e.g., HyperLoom, operating above those used in HPC centres. Another example is the NCSA (National Center for Supercomputing Application at the University of Illinois) scheduler tool[24] designed for Blue Waters (Cray-based) and other high-performance computing systems. If a researcher needs to handle thousands of single-node jobs rather than a single job that can use a thousand nodes, the batch queuing system becomes cumbersome. Furthermore, many HPC facilities limit the number of jobs a user may submit into the queues. The other limitation of standard batch queuing systems is that they do not allow node sharing between applications. NCSA scheduler allows a user to aggregate single-core jobs as a single batch and jobs share the node between applications using a simple configuration file. The scheduler allows queuing jobs and manages efficiently independent single-core jobs, and can bundle OpenMP single-node jobs but cannot bundle MPI jobs.

2.3.2 Autonomous Workflow Scheduling

Based on the flexibility in parallelism, parallel jobs can be divided into four different groups: (1) Rigid, (2) Moldable, (3) Evolving, and (4) Malleable. Although this Thesis deals with the moldable type of jobs, a brief description to make the differences clear is provided here.

Rigid jobs can only be excluded with a single fixed amount of compute resources which cannot be adjusted by the batch scheduler (or even by the user), e.g., a single-threaded job, or a parallel MPI job the user is limited to take exactly 4 computing nodes, etc. As mentioned above, batch schedulers that implement backfilling are, almost exclusively, adapted to deal with rigid jobs. Moldable jobs are flexible in the number of resources they can utilize, which gives the batch scheduler/user some degree of freedom to pick a suitable amount of resources to maximize the cluster utilization or minimize the makespan. However, once launched, the number of resources cannot be adjusted and remains static for the whole time of execution. Malleable and Evolving jobs are even more flexible and allow asking for more resources or releasing some during the job execution. For evolving jobs, changes are application initiated, while the changes in malleable jobs are system initiated. [17, 28]

During the last decade, many papers have focused on the estimation of the rigid workflow execution time and enhancing HPC resource management. For example, Chirkin et al. [11] introduces a makespan estimation algorithm that may be integrated into job schedulers. Robert et al. [35] gives an overview of task graph scheduling algorithms. Menaka et al. [39] gives a thorough overview of workflow scheduling algorithms and strategies used in cloud computing.

Naturally, common HPC batch schedulers allow rigid workflow/task executions meaning that users specify the number of resources together with other execution parameters upon job submission. Those allocated computing units are fixed for the duration of the entire job. In other words, such a rigid job consumes only a specific number of computing units and this cannot be optimized by the batch scheduler in any way. Rigid executions may be used for benchmarking, for example. However, most modern parallel applications are moldable, being able to exploit different levels of parallelism and ask for various amounts of computing resources just before their execution. In those cases, the remote facility performance could be improved. Selecting an appropriate number of resources even for a single job is, however, not an easy task. Users have to consider several factors like actual cluster utilization, and strong and weak scaling of the executed program. The difficulty of this task is rising with

[24] https://github.com/ncsa/Scheduler

a growing number of jobs executed within the workflow and their mutual dependencies. Generally, this task is very time-consuming, tiring and requires advanced knowledge. Some HPC workload managers have already tried to support moldable job submission, e.g., IBM's Load Sharing Facility (LSF).[25] This workload manager allows users to specify a range of required resources instead of a specific amount. The scheduling algorithm is, however, quite simple implementing a greedy algorithm that allocates as many resources as possible at the time of execution. [28]

While the field of rigid workflow optimization has been thoroughly studied, the autonomous optimization and scheduling of moldable workflows have still been an outstanding problem, although first opened two decades ago in [16].

Moldable workflow scheduling is considered a multi-criteria optimization problem. The objective here is to allocate resources to tasks in a workflow so that a set of performance criteria is optimized. These criteria can include objectives like minimizing makespan, maximizing resource utilization, minimizing cost or minimizing energy consumption. Naturally, these objectives often conflict with each other, which makes moldable workflow scheduling a challenging problem that requires the use of sophisticated optimization techniques to find a suitable solution.

Linear programming is a powerful tool for workflow execution scheduling as it allows decision-making to consider multiple conflicting objectives and constraints in a systematic and efficient manner. However, the complexity of the problem and the computational resources required for solving it increase with the size and complexity of the workflow, so it may not always be the most suitable method for large and complex workflows. [60]

Constraint programming is a paradigm for solving optimization problems that involve formulating the problem as a set of constraints on the variables and then finding values for the variables that satisfy the constraints. The constraint programming approach is well-suited for problems with complex constraints and for problems where the objective function is not well-defined or is hard to optimize directly. [54]

Other commonly used optimization methods are, e.g., simulated annealing [34], hill climbing [55] or random search [4]. These methods tend to get stuck in a local optimum and are not very effective in exploring big search spaces. Genetic algorithms (GAs), however, have several advantages over those heuristics. GAs are better suited for global optimization problems, as they explore the entire search space. Moreover, GAs handle large and complex search spaces efficiently and are less computationally expensive. Other advantages are the ability to incorporate various objectives and constraints for scheduling, support of flexible encoding schemes that can easily represent different scheduling policies and trade-offs between objectives, and multi-objective optimization. [45]

The usage of genetic algorithms addressing the task scheduling problems has also been introduced, e.g., a task graph scheduling on homogeneous processors using genetic algorithm, local search strategies [29] and a performance improvement of the used genetic algorithm [45]. However, a handful of works have taken into consideration the moldability and scaling behaviour of particular tasks, their dependencies and the current cluster utilization [7, 15, 71].

A significant complication in moldable workflow scheduling is the necessity to a priori know the execution time for particular tasks under different resource assignments, e.g., complete strong and weak scaling of particular jobs stored in the performance database. Several works addressed this problem [7, 28, 31, 63], however, they are either tightly con-

[25] http://www-03.ibm.com/systems/technicalcomputing/platformcomputing/products/lsf/

nected to an existing HPC cluster and its scheduler, use idealized models of strong scaling and parallel efficiency, or optimize only one criterion such as makespan, cluster throughput, or computational cost. The user tunability of these approaches is thus limited.

In reality, it is often not possible to benchmark the execution time for all possible combinations of the task types, task inputs and execution parameters. If a task has already been executed with given inputs and execution parameters, the execution time can be retrieved from the performance database. However, for unseen combinations, some kind of interpolation or machine learning techniques have to be used.

2.4 Cluster Simulators

Due to many reasons such as the cost of resources, the reliability, the varying background load or the dynamic cluster behaviour, experimental evaluation generally cannot be performed on real systems. Moreover, to obtain reliable results, multiple workflows with various execution parameters need to be performed using the same and controllable conditions that simulate different real-life scenarios which is, however, often not possible. Simple job scheduler simulators often provide a detailed model of the queuing behaviour as the jobs arrive at the system upon submission, wait for available resources, start their execution, and eventually leave the system upon their completion.

For example, PySS [42] is a trace-driven scheduler simulator. It implements a number of scheduling algorithms, including several backfilling ones. The problem with simple simulators is that they do not really model the target HPC system or the runtime behaviour of the applications. PySS takes the job runtime directly from the job trace, although in reality a job's runtime is affected by the specific resources allocated to the job and by the application's runtime behaviour, which can be affected by other jobs running simultaneously [44]. Thus, more sophisticated simulators need to be used instead.

Alea 4 [33] is an event-based grid and cluster scheduling simulator that uses the GridSim toolkit [9]. The simulator is able to deal with common problems related to job scheduling like the heterogeneity of jobs, resources, and dynamic runtime changes such as the arrival of new jobs or resource failures and restarts. The main part of the simulator is a complex scheduler which incorporates several common scheduling algorithms working either on the queue or the schedule (plan) based principle. The latest version of Alea uses a dynamic workload adjustment technique enabling user-to-system interactions to be modelled properly. The input is still a static workload (historical workload traces extracted from the HPC system itself, or from a public workload trace repository) but transformed into a dynamic one afterwards.

Performance Prediction Toolkit (PPT) [44] is a full-scale HPC simulator. It can use synthetic workload models or adopt job traces from existing HPC workload archives. The simulator implements several commonly used scheduling algorithms, however, it does not include backfilling algorithms. Other complex frameworks for studying grids, clouds, HPC or peer-to-peer systems have been developed. However, the majority of these projects seem to be inactive or abandoned. [33]

Chapter 3

Aims and Objectives

Although the workflow execution and scheduling investigation dates back several decades ago, it is still a hot research topic. There is a growing interest in developing novel workflow scheduling algorithms that can address crucial challenges. Different approaches to workflow scheduling have been explored, such as heuristic algorithms, machine learning, optimization techniques, and game theory. Moreover, workflow scheduling may be considered as an interdisciplinary research topic that involves computer scientists, domain experts, and practitioners from different fields which can lead to the development of innovative solutions to workflow scheduling problems. The next section gives a brief overview of open problems and challenges in workflow scheduling. Furthermore, the hypothesis of the work is introduced accompanied by aims and particular objectives.

3.1 Open Problems and Challenges

Workflow scheduling is a complex and challenging problem. The execution of many tasks on distributed and heterogeneous computing systems needs to be precisely coordinated. Here, a list of selected open problems and challenges follows:

- **Heterogeneous computing resources.** Scheduling workflows on heterogeneous computing systems poses many challenges, including load balancing, data transfer optimization, and resource allocation. Different computing platforms (CPUs, GPUs, FPGAs) have different computing power and capacity, data storage, and network connectivity. Naturally, their incorporation affects the workflow makespan and cost.
- **Workflow interoperability.** To enable the reuse and sharing of workflows across different domains and remote computing facilities, workflows have to be compatible with different tools, for example, being able to orchestrate different types of executable binaries and work under different environments. Another challenge is the standardization of workflows description and compositions.
- **Real-time decision making.** When planning the executions of workflows, many factors need to be considered, such as task input data and related parameters, type and amount of requested resources, current availability and utilization of computing facility, time and cost constraints, execution constraints like availability of particular executable binaries for different hardware, and so on. The decision must be made promptly before the situation at the facility has changed.

- **Quality of Service (QoS).** Workflow execution and scheduling are supposed to provide some guarantees about the quality of service including throughput, reliability and fault tolerance. Proper monitoring of executed workflows and remote computing facilities is essential for QoS. Although the process of monitoring itself may be straightforward, the challenge is to properly recognize and handle uncommon and suspicious situations.

- **Security.** Workflow executions have to ensure the security and privacy of workflow data and computations, especially when involving sensitive data, such as personal health information or financial transactions. Some of the challenges include certified binaries, data encryption and access control.

- **Multi-objective optimization.** In order to balance multiple objectives, such as execution time, computational cost, energy consumption, and resource utilization, workflow scheduling algorithms face several challenges. It is usually necessary to build a performance database and employ different optimization algorithms and heuristics to operate over this database, and finally perform, e.g., Pareto optimization or trade-off analysis.

- **Large scale workflows.** Workflows can grow in their size as well as complexity involving hundreds of tasks. The challenges include data replication and transfer, tasks coupling and conforming to execution queues policies, such as the maximum number of submitted jobs, disk quotas, etc. On the other hand, workflows capable of employing a vast number of resources face challenges with long queuing times, starvation, and large execution parameters search space which makes the multi-objective optimization even more complex.

3.2 Challenges from a User's Perspective

Depending on their knowledge and experience, users may struggle with different actions during the whole workflow execution process from constructing the workflow, over the data transfers to the execution and monitoring. When the workflow structure is constructed, its inputs and outputs together with dependencies are specified, accompanied by cluster-specific execution parameters.

Workflows may be constructed fully manually by creating job submission scripts (followed by a correct submission defining dependencies), using a programming language interface or graphical user interface. When using some dedicated tool interface, e.g., a WMS, the definition of workflows may be simplified in some manner. Workflows may or may not be created and stored in a standard format. Assuming no WMS or specialized tool is used, users have to access the cluster using the command line interface, manually transfer files to an intended place and correctly submit workflow tasks to particular computing queues.

Submission is the most crucial point of the execution progress. It affects the workflow makespan and its cost. The submission of every task in the workflow individually based on the input data and the strong and weak scaling of the executed program is very demanding and requires a low-level knowledge of the executed programs. Thus, the majority of users stick with the default parameters of the selected computational queue. However, in practice, the execution is completed in a much shorter time than requested. This leads to unnecessarily long waiting times in queues, potentially higher computational cost, and low

cluster utilization connected with the inefficiency of the batch scheduler. When submitted correctly (although possibly not sensibly), the only next step is to monitor the whole execution and transfer the output data back.

Monitoring may be way more tedious when the submission is done inefficiently. In daily practice, however, users face multiple issues when monitoring arising from (1) their own codes, (2) hitting quotas, (3) used libraries, e.g., MPI processes getting stuck, and (4) cluster itself, e.g., failing hardware, overloaded nodes and storage. Any of these issues may corrupt the completion of the execution and users have to know how to handle them. We may come to the conclusion that users in general have to gain some basic knowledge of the cluster architecture and deployed batch system at least, and their experience plays a significant role in the task submission and handling error situations. Constructing the workflow and its submission is time-consuming, but other steps are crucial in terms of utilization, efficiency and cost. The complexity of the whole procedure gets higher with the number of workflows and their size.

3.3 Hypothesis

Contemporary complex high-performance computing (HPC) systems do not allow users from academia and industry to use them without proper and deep knowledge. The design of a robust and user-friendly interface with workflow execution planning software is supposed to (a) bring HPC and the latest technologies to a much broader user base, (b) increase the processing efficiency by sophisticated workflow scheduling taking into account background HPC workload, (c) save resources, reduce the price of calculation and decrease computation time by selecting appropriate job execution parameters, (d) offer a level of fault tolerance by restarting faulty tasks with respect to dependencies, and (e) enable new methods to emerge and to gain new knowledge by allowing more users to cooperate on complex problems.

3.4 Research Objectives

The ultimate goal of the research presented in this Thesis is to

provide an automated and failure-free execution of workflows.

Based on the open problems and challenges depicted in this chapter, the main and auxiliary objectives are defined in this section. These objectives may be divided into research and implementation ones.

The main objectives of this Thesis are to:

1. Investigate multi-objective optimization methods in order to select a suitable amount of computing resources such as computing cores, nodes and accelerators for particular tasks in the workflow.

2. Collect and create performance datasets and identify experimental use cases to demonstrate the selected optimization method.

3. Provide an effective workflow execution planning with throughput maximization and latency minimization.

4. Create a piece of software providing the execution, data transfers, and monitoring of the workflows on HPC systems. This software parses the input data, automatically assembles a corresponding workflow, sets execution parameters and generates HPC system-specific jobs scripts. Required data is transferred to the target HPC system where the execution is orchestrated. The remote jobs are monitored and obtained result data is transferred back.

5. Experimentally evaluate the implemented optimization methods and software on selected use cases.

6. Asses the implemented solution, its benefits and discuss open issues.

Auxiliary objectives are to (1) design and implement additional features within the developed software, e.g., accounting, authentication and authorization, reporting, and fault tolerance, (2) store performance data after each successful run in order to build a custom performance database, and (3) design and implement a RESTful (Representational State Transfer) API (Application Programming Interface) together with a GUI (Graphical User Interface) application to handle workflow submissions easily.

Chapter 4

Design of the Workflow Execution Planning System

This chapter provides a high-level overview of the designed system for workflow execution planning, orchestration and monitoring. It is demonstrated how the Thesis goals were achieved and what led to the selected methods and solutions. This chapter also describes the impact of the designed system on practice.

4.1 Designed Solution and Research Steps

Together with this Thesis, a workflow management system called k-Dispatch has been developed. k-Dispatch was first developed as a lightweight platform for experiments done in the Thesis. By this time, it has become a stand-alone tool being successfully deployed and used by clinical users and researchers, utilizing remote Czech and UK computational resources. It should be noted, that k-Dispatch has also been commercialized and is being offered by the BrainBox, Ltd. company.[1] Experiments done in this Thesis were performed within k-Dispatch or aside as a module to be integrated into k-Dispatch in the future after additional tuning.

Thus, this section describes k-Dispatch together with performed experiments. It summarizes the research done and crucial decisions. The attached papers cited here provide more details and explanations.

k-Dispatch [mjaros6] represents a software package that provides biomedical workflows (1) execution offloading to remote computing machines, (2) execution planning, and (3) execution monitoring. Together with these key features, supporting mechanisms including accounting, reporting, file transfers and fault tolerance are implemented as well. k-Dispatch's mission is to make high-performance computing (HPC) facilities and cloud computational resources easily accessible as a service (providing HPC as a Service, HPCaaS) to end-users with no prior expertise in computational science. k-Dispatch's goal is to provide automated and failure-free job execution including advanced job execution planning.

k-Dispatch receives an input file including a workflow description tag and accompanying parameters together with the input data such as CT images. Since the development was driven by PAMMOTH,[2] a European H2020 project in the area of photoacoustic breast

[1] https://brainbox-neuro.com/products/k-plan
[2] https://www.pammoth-2020.eu/

imaging, and is further routinely used in CITRUS,[3] an EC EU - Horizon Europe project in the area of ultrasound neurostimulation, the k-Wave toolbox was employed in workflows. After the input file is parsed, a task graph corresponding to the workflow is constructed. This task graph serves as a template since execution details are not assigned to each task yet. Thereafter, compute allocation, i.e., assigned computational units such as core hours, together with the remote computing machine are selected and for each individual task in the workflow, and execution details are filled in. The execution details include an executable binary, its execution parameters, hardware selection (CPU, GPU, ...), number of computing units (i.e., nodes, cores, MPI processes), wall-clock time (i.e., execution time) and memory requirements. Such an evaluated task graph is called a specific execution plan. Since multiple execution plans may be generated, a quality function considering the execution plan with respect to the estimation of overall execution time (makespan) and computational cost is defined. Based on this quality function, a single execution plan is finally selected, submission job scripts are created and together with the input data transferred to the particular remote computing machine. Computing jobs processing particular tasks are submitted to particular computing queues. Their execution is controlled via individual job scripts. k-Dispatch then monitors submitted jobs and handles situations when jobs need to be altered or restarted. After the computation is finished, k-Dispatch downloads the results to its own storage cleans the remote space and performs reporting, accounting and performance database updates.

Since k-Dispatch's parts will be often referenced, a brief high-level description of its architecture follows here. The overall architecture of k-Dispatch is shown in Fig. 4.1. k-Dispatch consists of three main modules: *Web server*, *Dispatch database* and *Dispatch core*. The user applications, e.g., a stand-alone medical GUI or a web application, communicate with the Web server using the secured HTTPS protocol and REST API. The Dispatch database holds all the necessary information about the users, groups of users, submitted workflows, particular jobs, computational resources, available executable binaries for different hardware resources, types of tasks, and permissions. This database is crucial in processes that perform workflow execution planning, HPC selection, and accounting. Its structure is depicted in Fig. B.1 in the appendix and for better orientation, tables in the figure are coloured. The tables in green define workflows together with remote jobs, their possible dependencies and the files used. The tables in red are essential for executions. They define remote computational facilities, allowed codes, and their particular implementations, together with internal resource allocations used in submissions. The performance database is included in the Dispatch database as well. A single purple table stands for the performance database for k-Wave source codes. It is assumed to have multiple performance tables similar to this one when adding computing codes to capture their scaling under specific domain parameters. The tables in orange are used by both the accounting and the workflow execution planning systems when acquiring free resources. Lastly, tables in yellow define groups, users and software licenses. A brief description of individual tables can be found in Table B.1.

The Dispatch core is the most crucial part, which is responsible for the following key features: (1) planning, (2) execution and (3) monitoring of submitted workflows. The planning is a complex process addressed by this Thesis's research. The communication with HPC and cloud facilities is done via SSH and RSYNC protocols.

[3] https://mpbmt.meduniwien.ac.at/en/forschung/projekte/citrus/

From the application HPCaaS deployment point of view, we define three modules: (1) a treatment planning module (TPM), running on a local PC under the control of the end user, (2) a dispatch server module (DSM), running on a remote server under the control of the administrative team, this is equivalent to k-Dispatch, and (3) a simulation execution module (SEM), running on a remote computing machine (HPC, cloud, ...) under the control of its provider.

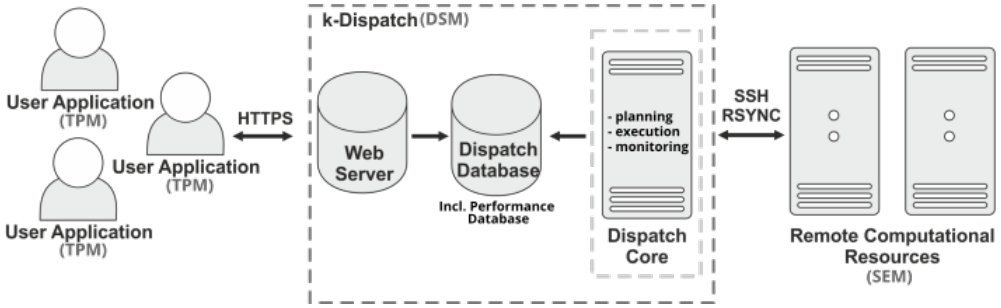


Figure 4.1: Simplified architecture of k-Dispatch (DSM) showing its three essential modules - Web Server, Dispatch Database including Performance Database, and Dispatch Core, and their connection to user applications (TPM) and computational resources (SEM). The key features, i.e., workflow execution planning, the execution itself and monitoring, are implemented within Dispatch Core.

4.2 Workflow Executions and Limitations

Workflows are formally defined as task graphs, see Paper II for details. Nodes represent executed tasks together with execution details while edges define task dependencies and data transfers. The k-Wave toolbox-based workflows are directed acyclic graphs (DAGs). They usually consist of many heavy computational tasks spreading over several compute nodes alternated with lightweight data processing tasks. These heavy computational tasks are meant to be *moldable*. It means that a task can be executed on various numbers of nodes as shown in Fig. 4.2.

Ideally, let us assume we have two vectors of 8 numbers and we want to add one vector to the other one. Such a task can be evenly distributed over 1, 2, 4, and 8 processing units. It can be simply shown that employing 8 processing units yields 8 times faster execution while the overall computational cost remains constant, no matter how many units are employed. This is called perfect strong scaling since the parallel efficiency stays at 100%. But this usually does not happen. Parallel algorithms almost always introduce some overhead caused by additional work to aggregate partial results, communication between processing units, and inherently sequential parts causing idleness of other units [3]. A typical example would be summing a vector of 8 numbers into a single value. A single processing unit would need 8 steps, while a parallel algorithm would need $\log_2 P$ steps, where P is the number of processing units employed. The speedup given by the quotient of sequential and parallel execution time reaches only $S = T_s/T_p = 8/3 = 2.66$, parallel efficiency $E = S/P = 2.66/8 = 33\%$, and the computational cost $C = P \times T$ grows from 8 and 21.28 for a single and 8 processing units, respectively.

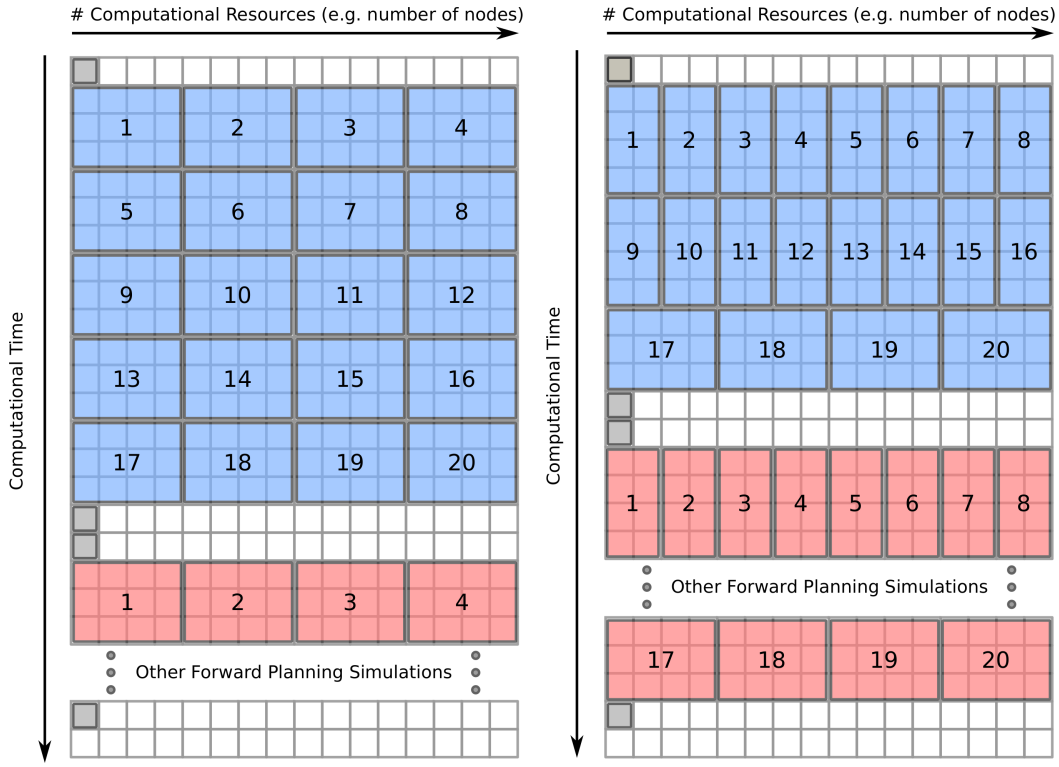


Figure 4.2: Example of two different execution plans of the neurostimulation workflow on a 16-node cluster. On the left, every job was optimized independently neglecting the queuing times caused to the subsequent jobs. On the right, the complete workflow was optimized together, which led to different resource allocations for particular jobs and shorter overall computation time. This figure shows how the number of resources assigned per task affects the execution plan and its quality.

Figure 4.3 shows a practical example of the MPI implementation of the k-Wave toolbox [32, 67] (SEM module) simulating (non)-linear propagation on ultrasound wave through a heterogeneous absorbing medium. The strong scaling of the execution time and cost for one specific problem instance on the Barbora cluster with 36 processor cores is depicted. In this case, a domain of 1024^3 grid points is partitioned into 2D slabs and distributed over various numbers of compute nodes (1 to 32). The red curve shows the execution time per one simulation time step (the whole simulation usually executes tens of thousands of time steps). The strong scaling curve looks generally very good but several sudden drops in the execution time caused by reaching a well-balanced workload distribution can be observed. Well-balanced (the number of 2D slabs and the number of MPI ranks are commensurable) and poor-balanced (some MPI ranks has more work than others) workload distribution are also reflected in the computational cost since there is an indirect proportion between the parallel efficiency and the related cost. The blue curve shows several local minima and maxima in the computational cost which provide suitable execution parameters or those that should be avoided, respectively. As already mentioned before, this plays a crucial role in submission and finding the execution parameters.

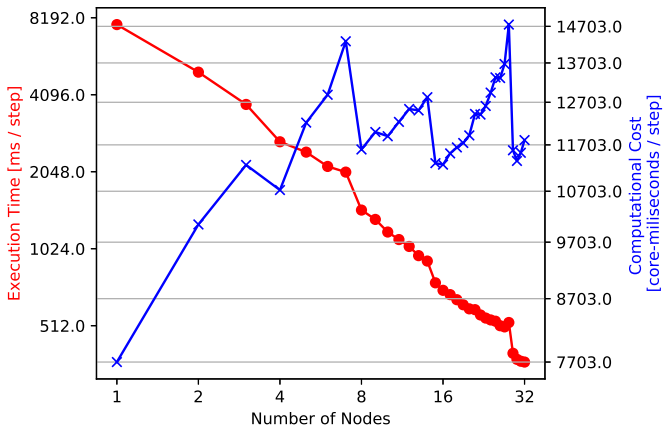


Figure 4.3: Red line shows the strong scaling of the k-Wave code measured for a domain size of 1024^3 grid points on the Barbora cluster. The blue line shows the evolution of the computational cost when more nodes are added.

As depicted in Fig. 4.4, a static acyclic execution model and both task- and data-driven workflows are supported. After the workflow is constructed, evaluated and submitted, no conditional behaviour, such as dynamic task generation or loops with an unknown number of iterations, is allowed. Since workflows may contain subgraphs that may be either omitted or repeated multiple times, this has to be determined during the planning phase while the final workflow is being constructed.

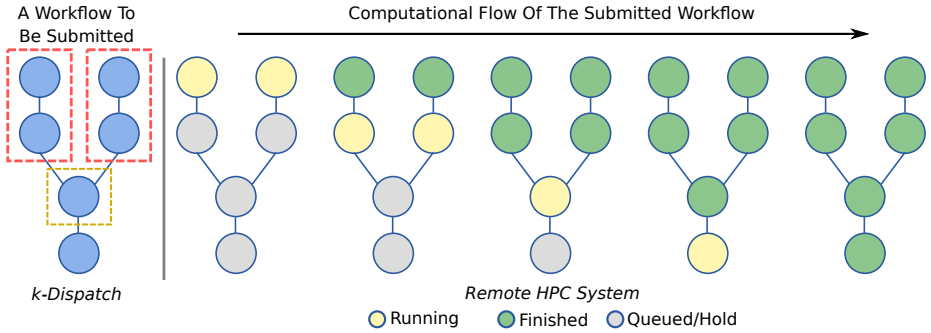


Figure 4.4: The execution model of workflows. The first blue workflow reveals the concurrency and dependencies between tasks. Tasks in separate red rectangles can be executed concurrently, in other words, there are no dependencies between other red rectangles. However, tasks within the red rectangle have to follow the dependencies. Tasks within a red rectangle are usually pipelined, i.e., one task waits for the previous task to finish. A task in a yellow rectangle has to wait till all red rectangles finish. Workflows on the right side of the dividing line show an example of computational flow on a remote computational machine. One may see the order in which tasks may be executed.

4.3 Investigated Workflows and Performance Datasets

Although the workflows used here have evolved a bit during the research, two typical biomedical ultrasound workflows are shown. These workflows are applied in ultrasound neurostimulation and photoacoustic imaging, see Fig. 4.5. Both workflows consist of two types of tasks. The simulation tasks (ST) executing the k-Wave MPI solver represent heavy parallel jobs running for a few hours. The k-Wave solver is based on the distributed fast Fourier transform (FFT) [21] which has a known communication bottleneck and $O(n \log n)$ time complexity. Moreover, the time to compute an FFT is highly sensitive to the factorization of the domain size, working well only for small prime numbers such as 2, 3, 5 and 7. [26]

In experiments, the STs were limited to use between 1 and 32 nodes (i.e., 36–1152 cores). The data processing tasks (PT) perform data pre-processing, post-processing, aggregation, etc. The PTs have a linear time complexity and almost perfect scaling. Since their runtime is in the order of minutes, only one or two nodes depending on the amount of memory requested are used.

The first workflow begins with a single PT generating input files for a set of STs. Consequently, a few independent series of ST-PT-STs are executed. Finally, the results from all series are aggregated using a parallel reduction tree composed of only PTs. The second workflow begins by running a few STs operating on the same input file but with different parameters. The results are aggregated into a single output file using a parallel tree reduction. But this time, the result is used by the following set of STs. In practice, this workflow is repeated in a loop until some error metric calculated by the last PT is satisfied.

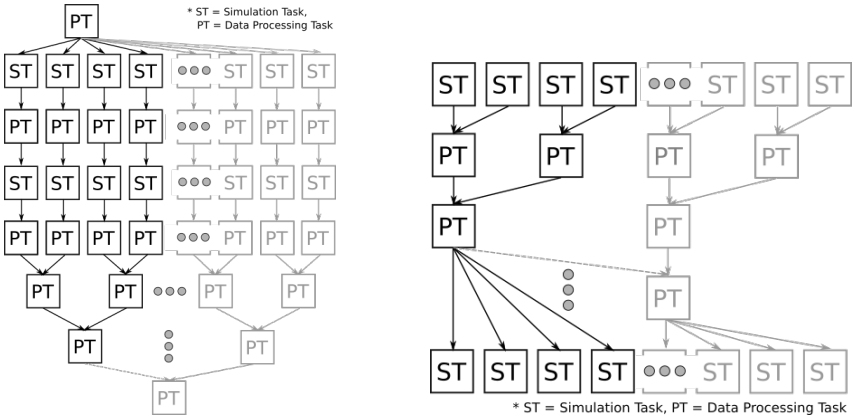


Figure 4.5: The structure of investigated workflows. The heavy simulation tasks are interlaced with light data-processing tasks. The parts highlighted in black show the minimal workflow structure consisting of 20 and 11 tasks, respectively. The parts displayed in grey show how the workflow structure can grow.

In order to explore various kinds of task scaling and assess the sensitivity of the execution planning to the scaling anomalies, both artificial (ranging from ideal scaling to extreme cases with peaks and bumps) and real performance datasets were used. Currently used performance datasets may slightly differ from those ones presented in already published papers, especially the real ones, due to continuing optimization of the underlying executable binaries. Additional information can be found in Appendix A.

4.4 Steps to Generality

Currently deployed version of k-Dispatch aims mainly to clinical users by supporting a subset of predefined biomedical workflows. Nevertheless, k-Dispatch was designed to be simply extensible to support other applications and different computing resources.

k-Dispatch is based on modular design (see details in Paper II) enabling a so called *plug and execute* approach. It allows easy extensions by adding (1) new computational workflows, (2) task execution planning strategies and algorithms, and (3) computing resources. Since k-Dispatch provides a simple and well-documented interface in the form of Python virtual classes, new functionality can be added by means of inheritance and polymorphism. Moreover, the implementation of extensions (1) and (3) is equivalent to filling out a form.

To add a new computational workflow, one has to define (a) how to parse the input file and what information to collect for advanced execution planning if required, and (b) job script templates together with commands to be executed on remote machines. Executable binaries together with required software modules may be added to the k-Dispatch database.

Adding new planning strategies and algorithms may be more complicated since they are supposed to be evaluated and tested before integration. When ready, the integration within k-Dispatch stands on the re-implementation of several virtual classes and adding new records into the factory design pattern. In order to be able to plan moldable task execution adaptively based on the input data and selected computational resources, performance data has to be stored in the database. k-Dispatch makes this data accessible for additional post-processing, filtering, mining, etc. Selected execution parameters can then be forwarded to the submission module which incorporates them into job execution scripts.

Defining a new computational resource is very similar to adding a new computational workflow. Essential information about the resource, e.g., name, address, batch scheduler commands and location of SSH keys, has to be specified. To support the new functionality, updates in both DSM and SEM modules need to be done.

Chapter 5

Implementation and Results

This chapter describes the implementation of the execution planning strategy and algorithms described in Chapter 4. This chapter also addresses the performed experiments and achieved results.

Execution planning is a process carried out by HPC batch schedulers. It can be described as a mapping of tasks within the workflow to free time slots and computational resources, see Eq. (5.1):

$$Q \to (T' \times R'), T' \subseteq T \wedge R' \subseteq R, \tag{5.1}$$

where Q is a set of all tasks in the workflow, T and T' are finite sets of all and available time slots, respectively, and R and R' are finite sets of all and idle computation resources at given time slots, respectively. Based on the scheduling policy, each scheduler attempts to maximize cluster utilization while guaranteeing the quality of service at some level.

The execution planning implemented in k-Dispatch follows Algorithm 1. The presumptions of the algorithm define necessary conditions and functions to (1) select an active resource allocation and an executable binary for each task, and (2) evaluate the quality of candidate workflows in terms of the makespan and cost. The workflow cannot be executed if no suitable resource allocation is found or the set of binaries for the particular task type is empty. The algorithm steps are performed in a loop until a suitable machine-specific workflow (best-evaluated workflow) is found. Points 3 and 4 of Algorithm 1 may be implemented by various optimization algorithms, e.g., genetic algorithms, simulated annealing, hill climbing algorithms, etc.

Algorithm 1: Workflow execution planning algorithm

Presumptions:

1 Let $G = (V, E)$ be a workflow where V is a set of tasks and $E \subseteq V \times V$ is a set of task dependencies.

2 Let C be a set of active resource allocations with enough resources to satisfy the workflow G. It holds $C \subseteq A$, where A is a set of all allocations the user has got access to.

3 All executable binaries for supported task types available in a given allocation $a \in A$ are defined as $D \in (B_1, B_2, \ldots, B_N)$, where N is the number of task types within the workflow G, and $B_i = \{b_1, b_2, \ldots, b_M\}$ is the set of available binaries for a given task type. B_i may be an empty set.

4 Let $p : G \times C \times D \to \mathbb{R}^+$ be a price function returning the aggregated computational cost of the workflow G.

5 Let $t : G \times C \times D \to \mathbb{R}^+$ be a function returning the aggregated execution time of the workflow G. This value is calculated as a critical path through the workflow considering both the net execution time e and the queuing time q.

6 Let workflow evaluation f serving as quality metric be defined as $f = \alpha \cdot p + (1 - \alpha) \cdot t$, where α is an optional ratio prioritizing the computational cost or the execution time.

Algorithm:

1 Create a workflow $G = (V, E)$ from the workflow template and input data.

2 Select a set of candidate allocations $C = \{c \in A^+ \mid c.status == active \wedge c.hours\_left > 0.0\}$.

3 Search for appropriate execution parameters for all tasks and evaluate the workflow G for all combinations of candidate allocations C and binary executables D.

4 Return the best parameters for a given workflow G as $\text{argmin}_{(c \in C, d \in D)} f(G)$.

The implementation of the execution planning in k-Dispatch is divided into four performance modules: *Optimizer* (see Paper III and V), *Estimator* (see Paper IV and V), *Evaluator* (see Paper III and V) and *Collector* (see Paper II). Optimizer traverses the search space and seeks for suitable execution parameters for particular tasks and their input data. If the collected performance database is sparse and incomplete, Estimator mines measured performance database and estimates execution time for the requested amount of resources. Once a candidate workflow evaluation is created, the Evaluator simulates the workflow execution and assesses its quality metrics. Finally, Collector updates the performance database after each successful workflow execution. Performance modules and their interactions are depicted in Fig. 5.1.

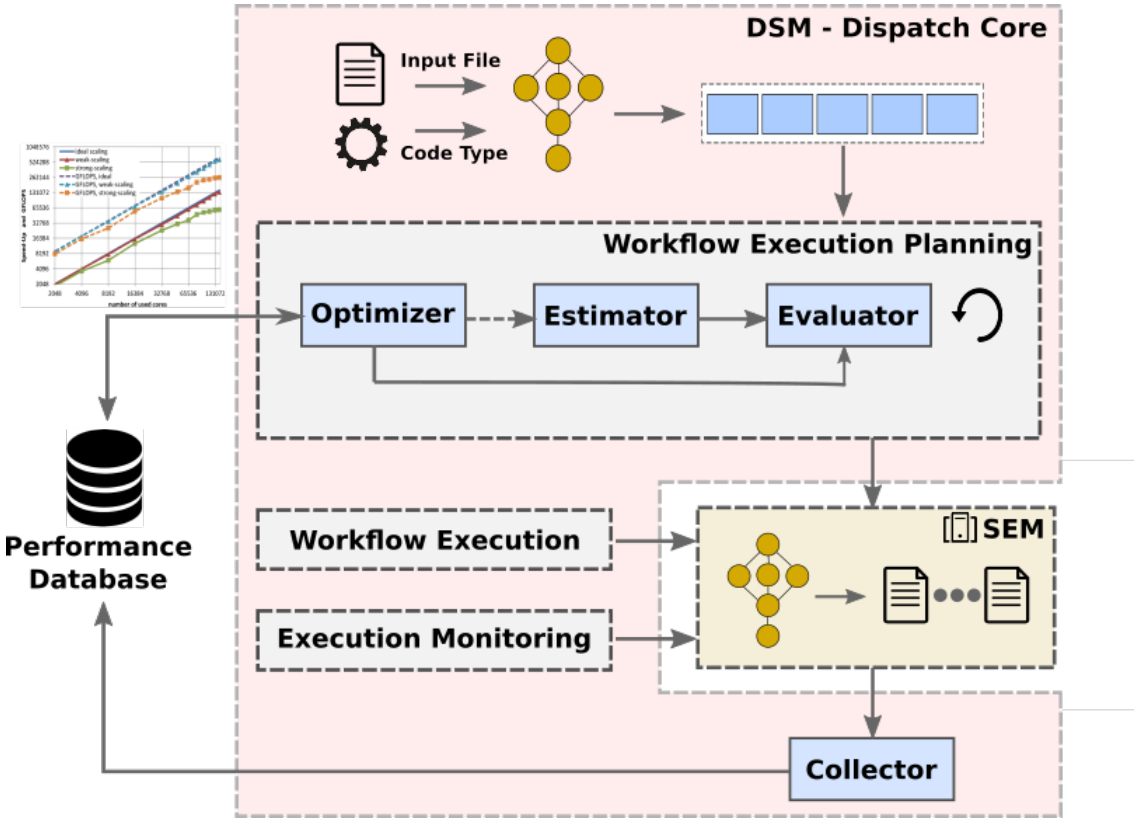


Figure 5.1: The illustration of the key features (grey) of Dispatch Core (pink), performance modules (blue) and remote computation (SEM, yellow). A workflow is created based on the input file and code types. For the workflow execution planning process, the workflow structure is transformed into a vector where each element represents the execution parameters of the particular task. The workflow execution planning is an iterative process managed by three performance modules, Optimizer, Estimator, and Evaluator. Optimizer selects suitable execution parameters based on the data retrieved from the performance database. Optimizer can invoke Estimator if the retrieved performance data is not complete, and regularly calls Evaluator to evaluate candidate workflow based on the predefined constraints and optimization criteria. The output of this grey box is an evaluated workflow in which computation, orchestrated using a set of job scripts, can be offloaded to the selected remote computational resource (Workflow Execution). The remote process of computation is then carefully watched by Execution Monitoring. When the computation is completed, the last performance module called Collector updates the Performance Database.

5.1 Optimizer: Execution Planning

Based on the research in the related work presented in Sec. 2.3, Genetic Algorithms (GAs) were chosen as the optimization. GAs are very effective and robust in the solutions of combinatorial problems and scheduling, well scalable and simply parallelizable. The crucial part of the designed approach is the fitness function, which can be simply employed by different space searching or machine learning methods. GAs use various operators and parameters to control the search. Selected operators and parameters used in the proposed solutions are summarized in Table 5.1.

Table 5.1: Selected Control Parameters of the Genetic Algorithm

Control parameter	Details
Selection method	Steady-state (sss), roulette wheel (rws), rank, tournament
Crossover method	Uniform
Crossover probability	0.7
Mutation method	Random
Gene mutation probability	0.1%, 0.5%, 1%, and 5%
Elitism	5% of the best individuals copied to the next population.
Population size	25, 50, 100, 150
Maximum number of generations	1000

The GA traverses the search space and seeks good solutions by applying genetic manipulations and selection strategies on the population of individuals/chromosomes. Each individual encodes a candidate assignment of execution parameters and binaries to particular tasks in the workflow. The quality of all individuals is evaluated by the fitness function. First, the execution time for every task is calculated based on the task type, execution parameters encoded in the individual, input data size, and known parallel efficiency/strong scaling behaviour. Next, the tasks are submitted to Evaluator (implemented using a cluster simulator, such as Alea or Tetrisator) that draws up an execution schedule and calculates the makespan as the critical path through the workflow including queuing times and computational cost. The output of the optimization is a set of best execution parameters for individual tasks minimizing given criteria implemented by the fitness function. The optimization process is depicted in Fig. 5.1.

From the vast number of existing implementations, PyGAD [22], an open-source Python library for building genetic algorithms, has been chosen. PyGAD supports different types of crossover, mutation, and parent selection operators. It allows different types of problems to be optimized using the genetic algorithm by customizing the fitness function. Since the PyGAD operators and fitness functions can be easily customized, we found this tool suitable to be integrated into our code which is also implemented in Python.

5.1.1 Workflow Encoding

To solve the workflow optimization problem using GAs, it is necessary to transform the workflow into a template for candidate solutions (chromosomes/individuals) I. The workflow's task graph is traversed in a breath-first manner producing a vector of N tasks. Every gene i corresponds to a single task and holds the number of resources R_i assigned to that task i, see Eq. (5.2).

$$I = (R_1, R_2, \ldots, R_N) \tag{5.2}$$

The amount of resources assignable per task is naturally constrained by 1 from the bottom and the size of the computing system from the top. However, acceptable values

may be further limited by the scaling behaviour of the task and the actual input data size. These constraints are imposed during the fitness function evaluation.

5.1.2 Fitness Functions

There may be designed many approaches to evaluate the quality of the workflow schedules and the suitability of chosen execution parameters to particular tasks. This thesis investigates two different approaches, the second of which is in two distinct variants.

The first approach, referred to as *local task optimization fitness function*, treats each task individually and independently trying to find the best execution parameters to optimize the objective functions (makespan or cost). The idea supporting this approach is that the workflow execution will be optimal if each task will have optimal execution parameters.

The second approach, referred to as *global workflow optimization fitness function* considers the dependencies amongst tasks and their interaction in computational queues, e.g., one long task employing a large number of compute nodes may delay many independent small tasks due to an insufficient amount of free resources, which can prolong the critical path and the makespan. This approach is examined in two variants designed for (1) on-demand allocation where users only pay for really consumed resources, and (2) static allocations where a portion of the computational facility is reserved for a given user/workflow, which may lower the latency but the user pays for all reserved resources, no matter of being idle.

These approaches are further described in the following Sec. 5.1.3 and 5.1.4.

5.1.3 Local Task Optimization

The local task optimization approach searches for suitable execution parameters of particular tasks independently considering only one optimization criterion, e.g., execution time or computational cost, while neglecting others, see Eq. (5.3) and (5.4).

$$fitness = t = \sum_{i=1}^{N} t_i(R_i) \tag{5.3}$$

where t is the aggregated net execution time of N tasks in the workflow, each of which running on R_i compute cores/accelerators/nodes for time t_i. An analogous fitness function may be written for the computational cost as

$$fitness = c = \sum_{i=1}^{N} (t_i(R_i) \cdot R_i) \tag{5.4}$$

which computes the cost of the task as a product of the execution time and the amount of resources used.

This fitness function sums the execution time of all tasks and calculates the total time consumed on the computing facility by the workflow. This fitness function relies on the batch job scheduler to assemble a good execution schedule allowing to run as many tasks as possible concurrently and minimize queuing time, which naturally contributes to the makespan but is not considered in this fitness function. This fitness function is suitable for large HPC clusters with hundreds of nodes and workflow composed of a few tasks employing low tens of nodes.

If this assumption forfeits its validity, the quality of the schedule may be degraded. A simple example is shown in Fig. 5.2 where the shortest execution time for tasks <0,6>

and <8,14> is reached with 36 nodes, but the computing facility only has 60 nodes. This prevents executing two tasks concurrently and increases makespan. It might have been better to use suboptimal execution parameters for those tasks, e.g., 30 nodes, to reduce the makespan to almost one-half.

From a practical point of view, this fitness function is the fastest one. The time complexity of the fitness function calculation is linear with the size of the workflow and does not involve running the cluster simulator.

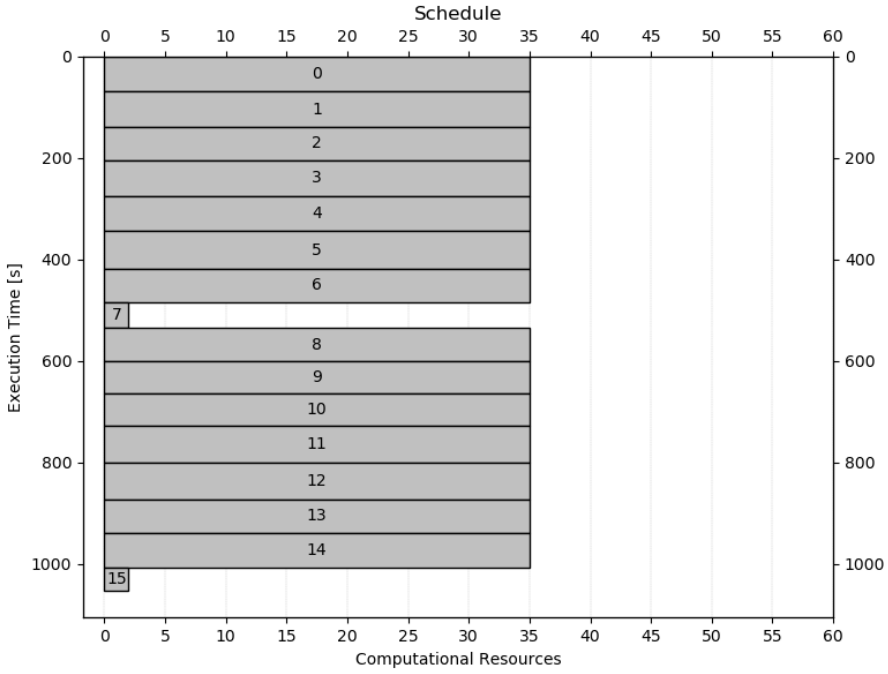


Figure 5.2: Execution plan performing the local task optimization for the workflow of 16 tasks. Each task is optimized for the shortest execution time independently resulting in workflow completion in 20 minutes. The workflow structure used can be seen in Fig. A.2.

5.1.4 Global Workflow Optimization

Unlike local task optimization, the global workflow approaches take into account the interaction amongst tasks and competition for resources. This requires a cluster simulator to be engaged to calculate the critical path in the workflow, see Sec. 5.3. This approach can produce better results, however, the optimization complexity significantly increases because of a more complex fitness function and the dependencies between tasks causing the search space growing exponentially with the number of tasks [57].

Table 5.2 shows the time necessary to evaluate a single generation of workflow of various sizes, ranging from 7 to 64 tasks, and different population sizes.

Table 5.2: The execution time of the evolution process for various population sizes and workflows with dependant tasks, evaluated by global workflow fitness functions on the Salomon cluster at IT4Innovations using a single computing core. The evolution runtimes for workflow without dependencies are approximately three times smaller.

Workflow Size / Population Size	**Runtime per a Single Generation in Seconds**							
	7	8	15	16	31	32	63	64
25	0.004	0.005	0.010	0.010				
50	0.007	0.009	0.019	0.021	0.040	0.043	0.112	0.120
100	0.013	0.018	0.037	0.043	0.077	0.088	0.227	0.220
150					0.110	0.132	0.382	0.335

Global workflow Optimization on systems with on-Demand Allocations (GODA)

This fitness function minimizes the makespan [27], the overall execution time t of the workflow given by the sum of the execution times of the tasks along the critical path in the workflow graph, together with the computational cost c given by a sum of the computational cost of all tasks in the workflow, see Eq. (5.7). No delays caused by sitting in computational queues while waiting for free resources are considered, but can simply be added by the cluster simulator by reading the time the first task was submitted and the time the last task left the system.

As we know, the optimization criteria may go against each other making the optimization more challenging. Here, makespan and cost are such parameters. Therefore, a trade-off coefficient α to prioritize either makespan (bigger α) or cost (smaller α) is introduced. This coefficient can be set by the user when the workflow is submitted to k-Dispatch. In order to balance between proportionally very different criteria, a kind of normalization is introduced. The makespan is normalized by the maximum total execution time of the workflow t_{max}, which is considered to be the sum of the execution times of all N tasks executed by only a single compute node in a sequential manner. The cost is normalized by the minimum computational cost which is the cost of the workflow computed by a single node in a sequential manner, see Eq. (5.5). This presumption is valid for typical parallel algorithms with sublinear scaling, i.e., parallel efficiency as a function of the number of nodes is always smaller than 1.

The algorithm cannot perform a true multi-objective optimization because there is no further feedback from the user that could select the preferred solution from the Pareto frontier. Instead, the most suitable solution has to be chosen autonomously and submitted to the cluster as soon as possible (before the cluster background workload changes significantly).

$$c_{min} = t_{max} = \sum_{i=1}^{N} t_i(1) \tag{5.5}$$

$$c_i = t_i(R_i) \cdot R_i \tag{5.6}$$

$$fitness = \alpha \cdot \sum_{j \in M} \left(\frac{t_j(R_j)}{t_{max}} \right) + (1 - \alpha) \cdot \sum_{i=1}^{N} \left(\frac{c_i(R_i)}{c_{min}} \right), \quad \text{where } M = \{i | i \in \text{CriticalPath}\} \tag{5.7}$$

This fitness function suits best the workflow being executed in environments with shared resources where only truly consumed resources are paid for, e.g., shared HPC systems. Figure 5.3 depicts two execution plans for the same workflow (see Fig. 4.5) and demonstrates the differences in the makespan and computational cost while different trade-off coefficients are used.

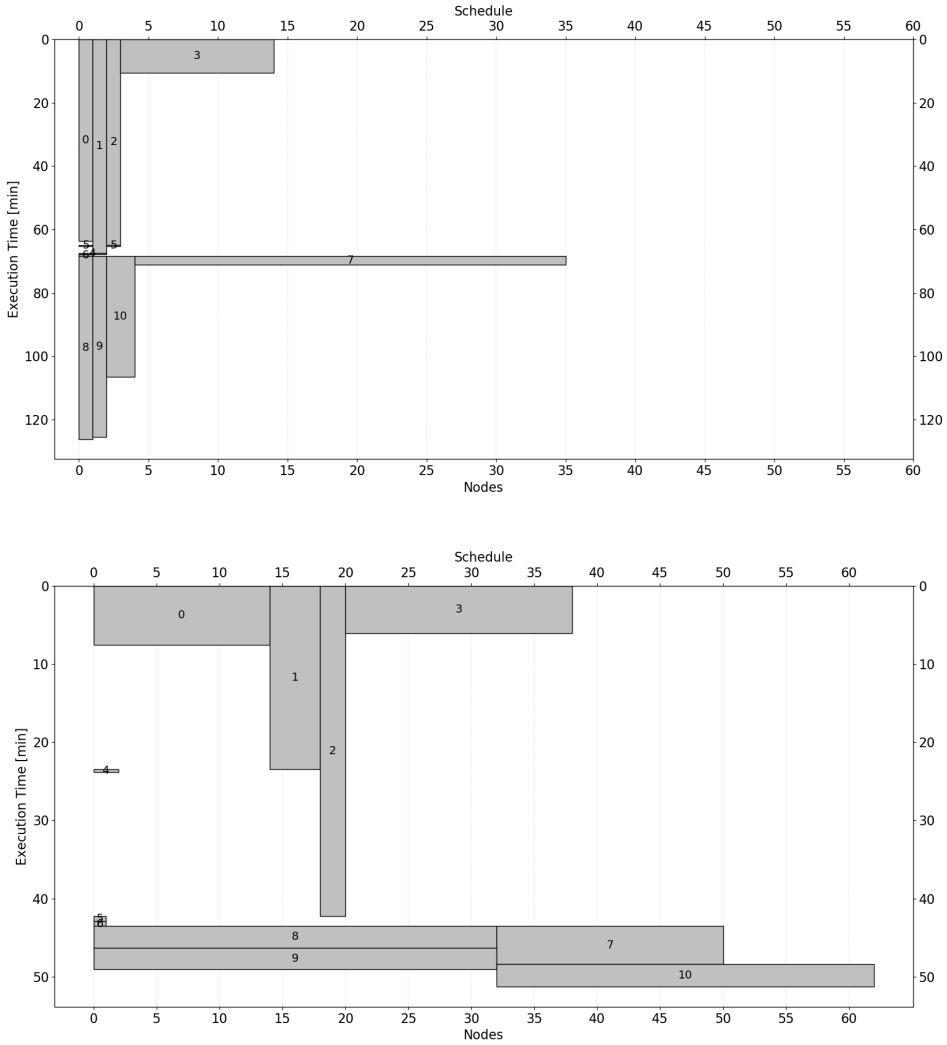


Figure 5.3: Execution plans designed under the GODA fitness function for the workflow of 11 tasks and different α parameters. The trade-off parameter of 0.05 (top) pushes down the overall execution cost by allocating fewer resources to tasks. This plan completes in 126 minutes and costs 592 node minutes. On the other hand, the trade-off parameter of 0.95 (bottom) prioritizes the makespan over cost. It is visible that tasks demand many more compute nodes. This plan completes in 51 minutes and costs 747 node minutes.

Global workflow Optimization on systems with Static Allocations (GOSA)

The last fitness function, described by Eq. (5.9), also minimizes the workflow makespan, but the computational cost now considers also idling nodes. The computational cost, the

user will be accounted for, equals the size of the allocation multiplied by the makespan, no matter whether some nodes are not being used for the whole duration of the workflow execution. Therefore, the fitness function attempts to shake down the tasks to minimize the amount of idling resources while still minimizing the makespan. The computational cost is then normalized by the highest possible cost in the dedicated system where only one node works.

Although these allocations may be more expensive, they usually reduce the queuing time. Since the makespan and cost are directly proportional, no trade-off coefficient is needed and only the makespan is considered, as demonstrated in Paper III.

$$t_{max} = \sum_{i=1}^{N} t_i(1) \tag{5.8}$$

$$fitness = \sum_{j \in M} \left(\frac{t_j(R_j)}{t_{max}} \right), \tag{5.9}$$

$$\text{where } M = \{i | i \in \text{CriticalPath}\}$$

Similarly to the previous case, t is the overall execution time of the workflow, and t_{max} is the maximum overall execution time obtained for serial scheduling of sequential tasks. The number of nodes statically allocated to the workflow is denoted by P. The number of nodes assigned per task i is R_i.

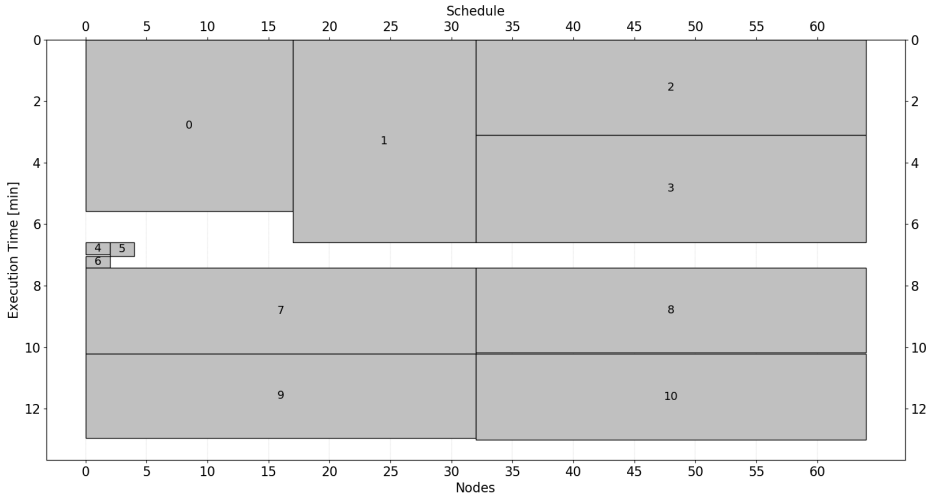


Figure 5.4: The execution plan designed under the GOSA fitness function for the workflow of 11 tasks. This plan completes in 13 minutes with only 8.45 % of allocated resources idling. When only a single workflow is submitted, the obtained numbers are highly dependent on the workflow structure used and how well the tasks can utilize free resources. The cost of such a solution is 832 node-minutes.

5.1.5 Investigation of GA Control Parameters

This section summarizes the process of finding suitable control parameters for the genetic algorithm, more details can be found in Appendix A. These experiments use a custom

performance database consisting of both artificial and real scaling data obtained for the acoustic k-Wave toolbox [32]. The experiments are performed for both local task and global workflow optimization. The experiments cover workflows with and without dependencies (embarrassingly parallel), however, the presented results only show workflows with dependencies. Investigated workflows contain between 8 and 64 tasks. Bigger workflows were not taken into account since many HPC facilities limit the number of currently submitted tasks to low tens, e.g. 100, at IT4Innovations where the experimenters were conducted.

For the local tasks optimization, we can conclude that the best selection strategy driving the GA through the search spaces appears to be steady-state selection, although the difference between the rank and tournament selections was marginal. Uniform crossover produced the best results and the influence of the crossover probability was not statistically significant. The optimal gene mutation ratio seems to be around 0.5 %. As expected, the better the task scaling, the faster the convergence and the higher the success rate of finding the optimal resource assignment. The number of generations to be evaluated before the GA finds an optimal schedule stays relatively constant close to 200. The execution time of the evolution appears to grow linearly. This growth can be attributed to a product of increasing population size which rises the number of fitness function evaluations, and the linearly growing time complexity of the fitness function evaluation. Nevertheless, an execution time of 14 s on a single core with a 95 % success rate for the workflow containing 64 tasks is an excellent result. For the workflows omitting dependencies, the runtime is roughly 20 % shorter. The results were obtained from 20 independent runs of GA on the IT4Innovations' Salomon supercomputer.

For GODA fitness function, steady-state selection and the random mutation probability of 0.1 % led to the best results. For workflows counting less than 32 tasks included, however, rank selection provided the best results. Other control parameters remained the same as for the local optimization. Small populations counting about 50 individuals seemed to be sufficient for benchmarks with good-looking strong scaling. Otherwise, a population of 150 individuals had to be employed to find sensible solutions.

Our very first experiments with GOSA fitness function, published in Paper III, employed a trade-off coefficient between the makespan and cost, as well. However, experiments showed that the fitness function can be simplified. This gave us a different perspective on the problem, we modified the fitness function and published the latest result in Paper V. Nevertheless, we were still able to find solutions that reduced unused resources to 29 % but balancing the different optimization criteria could not be well controlled. When task dependencies are considered, the ability to reduce idling resources also stands on the structure and size of the particular workflow. Recommended settings of the GA are roulette wheel selection using a 1 % random gene mutation for workflows counting more than 64 tasks, and steady-state selection with a 0.5 % random gene mutation for smaller workflows. Uniform crossover of a 0.7 rate together with a 5 % elitism remains the same as for other methods.

5.2 Estimator: Incomplete Performance Data Handling

When planning the task execution, we may face the following situations: (1) The performance dataset measured for given tasks is complete and no estimation is required. Such a dataset contains execution times measured for all possible amounts of computational resources such as nodes. Moreover, when having multiple measurements, their median value can be taken to smooth out anomalies. (2) The performance dataset for a given input

data is incomplete and sparse, i.e., the execution time for some amounts of resources is not known. Missing values need to be estimated. (3) No performance data is available for a given input data and the full performance dataset has to be estimated. In other words, execution times for all possible amounts of resources have to be estimated from performance data collected on similar input data. This Thesis investigated cases (2) and (3).

5.2.1 Interpolation

Interpolation is commonly used in various fields such as engineering, computer graphics, and scientific computing, where it is used to estimate values of a function at points where it is not explicitly defined. More precisely, it is a mathematical method of estimating or calculating the value of a function at a point within the range of the function, based on the known values of the function at nearby points.

A thorough investigation of interpolation techniques was performed in Papers IV and V. Finally, two different interpolation techniques, linear and quadratic interpolation, from Python's scipy package [69] were used to estimate missing execution time for a particular task, input data size, and the number of resources (compute nodes in this case).

Although interpolation methods are very straightforward and easy to implement, they may suffer from the inability to accurately estimate the execution time of tasks with poor scaling. Such scaling may be characterized by drops caused by poor/good workload and distribution, complex communication patterns and parallel overhead.

5.2.2 Evaluation Metrics

As the measure of the interpolation quality, a mean relative error was used, see Eq. (5.10).

$$meanError = \frac{1}{N}\sum_{i=0}^{N}\left(\frac{|a_i - b_i|}{a_i}\right) \tag{5.10}$$

where a denotes the measured execution time, b the estimated execution time, and N is the total number of the compute nodes.

5.3 Evaluator: Workflow Quality Evaluation

The Evaluator module is responsible for calculating the makespan and cost for global workflow optimization fitness functions. As such, it is directly called from within the fitness function and can possibly calculate additional characteristics of the workflow schedule, such as average queuing time, node idle time, etc. In order to do so, Evaluator simulates the execution of individual tasks in the workflow following the constraints of the target HPC cluster and its batch job scheduler.

Naturally, an instance of the batch scheduler (PBS or Slurm) running in a Docker container was considered to take charge of the Evaluator. This container would have been executed for each evaluation of the fitness function. Although the execution and processing of data coming to and from this Docker container was quite straightforward in Python, this approach was not finally used due to the high time complexity of an accurate HPC system simulation. Moreover, too low-level understanding of the target system and the factors that impact its performance and efficiency were required to get reliable outputs.

On the other hand, the key parameters of the cluster simulator that shall be taken into account when modelling the target batch scheduler and the HPC cluster are listed below:

- number and type of compute nodes,
- computational queues,
- resource allocation policies,
- job scheduling and submission policies.

Since all of these parameters have a significant impact on the accuracy of the makespan evaluation, they need to be set properly to accurately reflect the target system. Thus, a simplified model of the batch scheduler was implemented in the Python language with the emphasis put on the fast evaluation. This model is called Tetrisator (see Paper V). Tetrisator is a PBS-based simulator that also implements backfilling, and allows specification of initial background workload. As implemented, Tetrisator can simulate the execution of workflows coming to the system in sequential order.

Experiments with a more robust and complex Alea simulator were performed as well. Paper I confirmed it as a possible candidate for Evaluator but evaluated it as far too complex to be used with the GA. Assuming that a workflow of 64 tasks with dependencies may be evolved in 140 s using 67,500 evaluations of the fitness function (i.e., 450 generations and 150 individuals), a single evaluation takes 0.002 s in Paper III. In Paper I we confirmed that Alea is able to simulate the same-sized workflow under 2.5 s. Employing Alea in the developed approach would have made the optimization time 1,000 times longer, which is enormous. We affirmed that although our custom cluster simulator (Tetrisator) has some limitations, it can provide reasonable estimations and works accurately for static allocations.

Nevertheless, Alea is still a live project that is being worked on. This Java-based tool was extended to support dependencies in workflows, enables switching among different schedulers and scheduling policies, and can simulate the execution of multiple workflows at the same time.

5.4 Experimental Results

This section briefly discusses the quality and accuracy of the developed schedules when using the performance database (1) containing all data, (2) only a subset, and (3) no data for particular task inputs using the GOSA fitness function as presented in Paper V. Realistic k-Wave workflows with various number highly parallel ST tasks and short PT tasks were used for the evaluation.

Table 5.3 summarizes these experiments in two parts. The left part investigated a workflow with tasks spreading over a domain size of 1024^3 grid points where missing strong scaling values were completed by a linear interpolation. Performance datasets mentioned in this part of the table are depicted in Fig. 5.5 where *Dataset 1* is on the right while *Dataset 2* on the left. These datasets differ in the number of known values on the strong scaling curve, i.e., 8 and 16, respectively, while the rest are estimated. The right part of the table investigates a workflow running over a domain size of 810^3, where the execution time was fully interpolated using a quadratic interpolation, see Fig. 5.6. The table states the minimum, maximum and average times of achieved makespans in minutes obtained from 20 independent runs of the GA. The first row reveals reference values measured for fully known performance datasets. Column *Diff.* in other rows describes a percentage difference between the achieved makespan on the full performance dataset and interpolated

datasets. This difference is given by an interpolation error and performance fluctuations of the cluster's nodes.

Table 5.3: The results show fitness function GOSA applied on k-Wave workflows with the domain of 1024^3 grid points on the left and 810^3 grid points on the right. Experiments were performed using (1) the full performance dataset without interpolation, (2) the partial performance dataset of 8 and 16 known values, respectively, completed using linear interpolation, and (3) the complete performance dataset estimated using quadratic interpolation. The table depicts the average (Avg), minimum (Min) and maximum (Max) obtained values of the makespan in minutes. The percentage difference between experiments with partial and full performance datasets is also depicted.

1024 x 1024 x 1024		40 Tasks Makespan [min]	40 Tasks Diff. [%]	80 Tasks Makespan [min]	80 Tasks Diff. [%]	810 x 810 x 810		40 Tasks Makespan [min]	40 Tasks Diff. [%]	80 Tasks Makespan [min]	80 Tasks Diff. [%]
GOSA	**Avg**	**29.70**	-	**58.31**	-	**GOSA**	**Avg**	**14.82**	-	**30.05**	-
with no	**Min**	27.75	-	55.74	-	**with no**	**Min**	14.07	-	28.32	-
interp.	**Max**	35.10	-	61.07	-	**interp.**	**Max**	16.88	-	31.76	-
GOSA with	**Avg**	**29.19**	**1.72**	**59.23**	**1.57**	**GOSA**	**Avg**	**17.08**	**15.25**	**33.11**	**10.18**
linear interp.	**Min**	27.29	1.65	55.27	0.84	**with quadratic**	**Min**	15.44	9.70	31.27	10.41
(Dataset A1)	**Max**	33.25	5.27	65.47	7.21	**interpolation**	**Max**	18.85	11.64	36.67	15.44
GOSA with	**Avg**	**26.74**	**9.98**	**51.06**	**12.44**						
linear interp.	**Min**	24.87	10.36	49.05	12.00						
(Dataset A2)	**Max**	30.33	13.58	56.46	7.55						

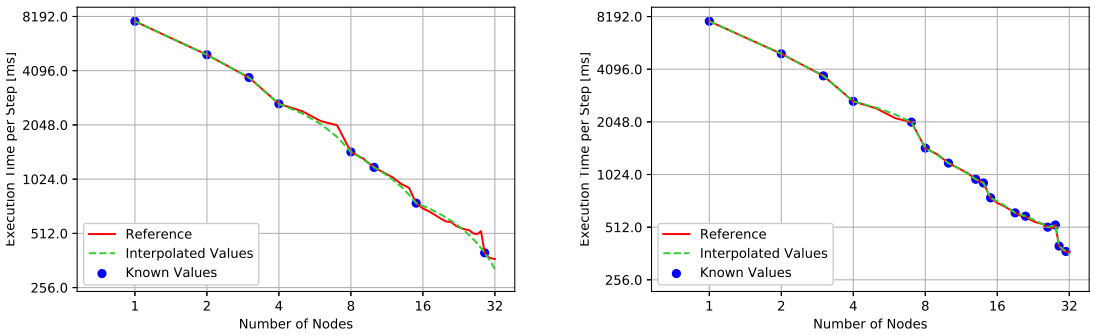


Figure 5.5: Reference and interpolated strong scaling of ST tasks for a domain size of 1024^3 grid points with a linear interpolation calculated from 8 and 16 known values, respectively. In the left figure, values in peaks were selected intentionally to see how much the value would be underestimated.

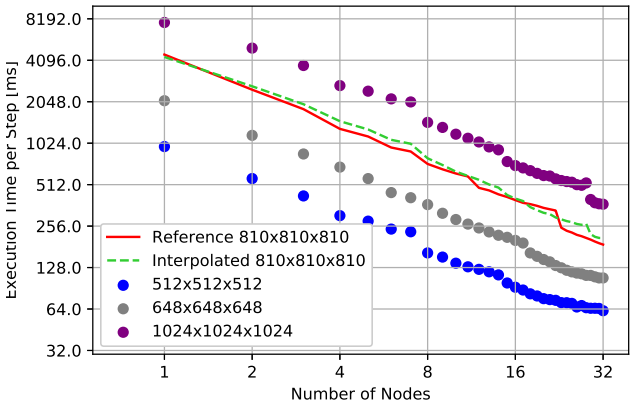


Figure 5.6: Reference and interpolated strong scaling of ST tasks for an unknown domain size of 810^3 grid points with quadratic interpolation.

We may conclude that the number of missing values in the strong scaling has a significant impact on the accuracy of the makespan prediction. On the other hand, having a difference of less than 10 % or 13 %, for 40 and 80 tasks respectively, while only 8 from 32 values in the performance dataset were known is a very good result. When 16 from 32 values are known, the difference drops below 2 %. It has been demonstrated that linear interpolation works very well in situations where the input data has been seen before and the task has already been executed using a few execution parameter configurations. For yet unseen inputs (domain sizes) where the whole performance dataset has to be estimated based on similar inputs, the difference is below 15% and 10 %, respectively. This affirmed that a quadratic interpolation works sufficiently well for our codes having $O(n \cdot \log n)$ time complexity with respect to the domain edge size and P^2 communication complexity with respect to the number of MPI ranks (cores used). For codes with different time complexity, higher polynomial interpolations may produce better results. However, when the whole domain is estimated, there are several variables that portray and impact the achieved accuracy, i.e., characteristics of other collected performance data (similarities in strong scaling vs. anomalies, domain size, medium parameters, and so forth). We may say that overall achieved results are very promising.

The evolved schedules were cross-validated with the real schedules created by the PBS job scheduler on the IT4Innovations' Barbora cluster. This showed a good general match, see Fig. 5.7. It can be seen that the evolved schedule finishes slightly earlier than the real one. This is caused by a tiny lag of the PBS scheduler connected with the orchestration of the job execution (see thicker lines in the plot, e.g., between tasks no. 10 and 11).

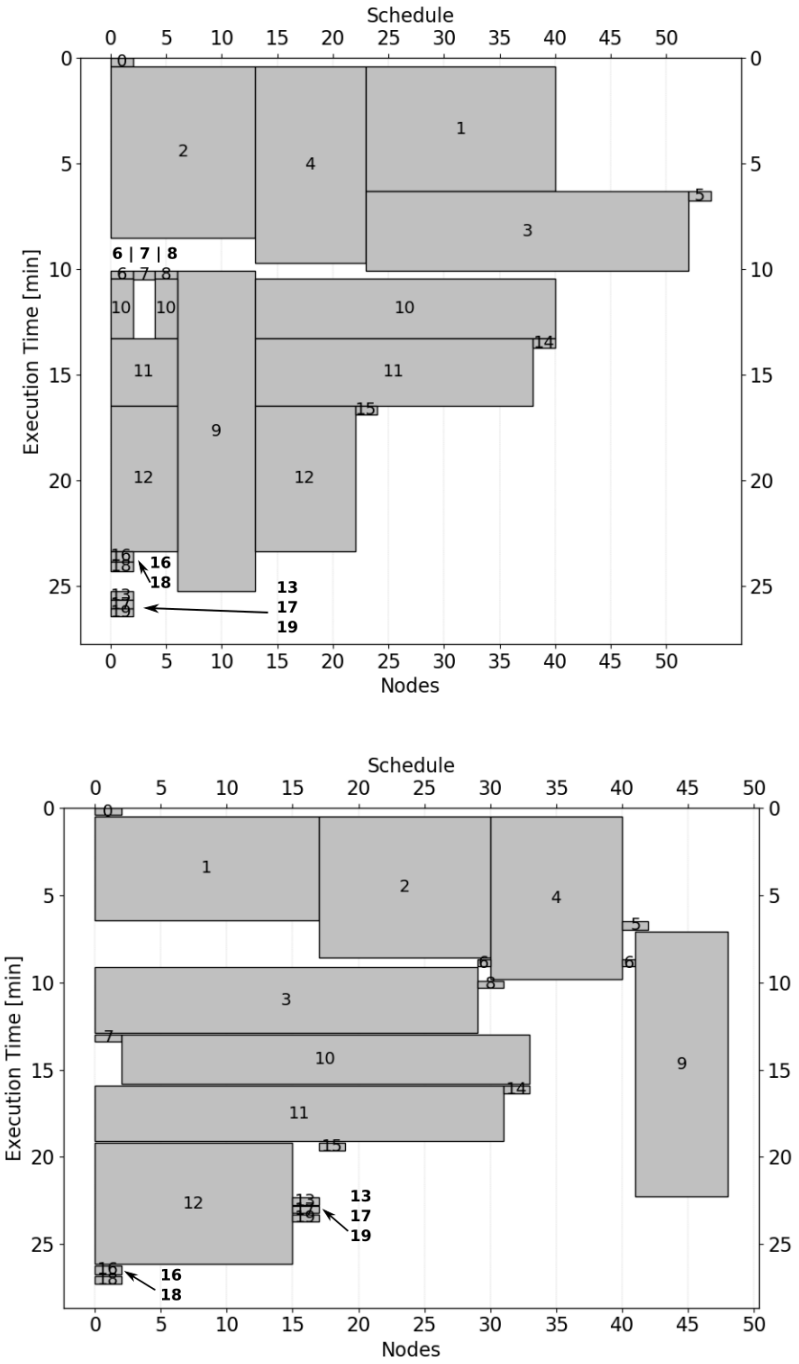


Figure 5.7: Simulated execution schedule finishing in 26.4 minutes (top), and the corresponding real workflow execution on Barbora (IT4Innovations) finishing in 27.3 minutes (bottom).

5.5 Discussion

This chapter has provided a high-level overview of the optimization of the execution parameters and workflow scheduling implemented in the k-Dispatch software.

The workflows used stand on the k-Wave toolbox and reflect real biomedical use cases. When incorporating new codes, it is recommended to analyse them first and provide basic benchmarking on target computational machines and typical input data to pre-fill the performance database. Understanding used algorithms, communication and decomposition techniques and typical workload is key for selecting appropriate optimization heuristics.

As for k-Wave codes, there are many parameters inside the input data, such as the domain size, heterogeneity of the media, linearity/non-linearity and lossless/absorbing wave propagation. These variables influence the computation time, and obviously the cost as well. We performed a correlation analysis and experimentally verified how much these parameters affect the computation time. A strong correlation was found for heterogeneity, where heterogeneous media required 40 % longer execution, absorption causing an additional increase of 30 %, while non-linearity had negligible impact. Based on this knowledge, the size of the performance database was reduced because the impact of some parameters can be calculated on the fly.

Recently, we have started experiments employing symbolic regression to capture the task strong scaling, which will be deployed in further versions of Estimator. Early experiments showed very promising results reaching the maximum error of 7.3 %. Since this topic is still under investigation and has not been published yet, its description, experimental results together with future research steps was added to Appendix C.

Chapter 6

Research Summary

The research outcomes consist of five original publications published at leading HPC-related conferences and workshops ordered by their release date. For each publication, an abstract, the author's contribution, and other relevant information are provided.

Paper I [mjaros4], summarized in Sec. 6.1.1, answers the question of how to optimise workflow scheduling based on historical performance data and demonstrates this concept using the Alea simulator. Paper II [mjaros6], summarized in Sec. 6.1.2, introduces k-Dispatch as a workflow management system (WMS), ranks k-Dispatch amongst other WMSs, and reveals both development and deployment details. Big progress in workflow optimization is depicted in Paper III [mjaros2] which was summarized in Sec. 6.1.3. In this paper, a genetic algorithm-based scheduling is introduced together with local task-based and global workflow-based quality assessment functions. To satisfy contradictory objectives, a trade-off coefficient is established. Paper IV [mjaros5], summarized in Sec. 6.1.4, overcomes another challenge in the execution optimization, i.e., how to deal with incomplete performance data and yet unseen inputs. This paper serves mainly as a proof of concept comparing several interpolation methods in different but realistic scenarios. Finally, Paper V [mjaros3], summarized in Sec. 6.1.5, follows up the previous paper, integrates the selected methods to the optimization algorithm, and demonstrates the simulated results on a cluster. All papers discuss and compare the achieved results, and propose future research directions.

The full texts of these publications are available in Appendices I–V of this Thesis. Lastly, a list of other author's publications, research projects, fellowships and awards is provided at the end of this chapter in Sec. 6.4.

6.1 Publications

6.1.1 Publication I

JAROS Marta, KLUSACEK Dalibor and JAROS Jiri. **Optimizing Biomedical Ultrasound Workflow Scheduling Using Cluster Simulations.** In: Job Scheduling Strategies for Parallel Processing. JSSPP 2020. Lecture Notes in Computer Science (including subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics), vol. 12326. New Orleans: Springer Nature Switzerland AG, 2020, pp. 68-84. ISBN 978-3-030-63170-3. Available from: https://link.springer.com/chapter/10.1007/978-3-030-63171-0_4

- Publication type: conference paper
- Conference ranking: B (ERA), B1 (Qualis), B (CORE2020, CORE2021)
- Author's participation: 70%

Author's Contribution

Collection of historical performance data on the HPC cluster and its post-processing. Preparing and running experiments on the cluster, and evaluation of results. Preparing experiments executed using the Alea simulator and the evaluation of the results. Creating figures and contributing to writing the paper parts regarding k-Dispatch.

Abstract

Therapeutic ultrasound plays an increasing role in dealing with oncological diseases, drug delivery and neurostimulation. To maximize the treatment outcome, thorough pre-operative planning using complex numerical models considering patient anatomy is crucial. From the computational point of view, the treatment planning can be seen as the execution of a complex workflow consisting of many different tasks with various computational requirements on a remote cluster or in cloud. Since these resources are precious, workflow scheduling plays an important part in the whole process. This paper describes an extended version of the k-Dispatch workflow management system that uses historical performance data collected on similar workflows to choose suitable amount of computational resources and estimates execution time and cost of particular tasks. This paper also introduces necessary extensions to the Alea cluster simulator that enable the estimation of the queuing and total execution time of the whole workflow. The conjunction of both systems then allows for finegrain optimization of the workflow execution parameters with respect to the current cluster utilization. The experimental results show that this approach is able to reduce the computational time by 26%.

6.1.2 Publication II

JAROS Marta, TREEBY Bradley E., GEORGIOU Panayiotis and JAROS Jiri. **k-Dispatch: A Workflow Management System for the Automated Execution of Biomedical Ultrasound Simulations on Remote Computing Resources.** In: Proceedings of the Platform for Advanced Scientific Computing Conference, PASC 2020. New York: Association for Computing Machinery, 2020, pp. 1-10. ISBN 978-1-4503-7993-9. Available from: https://dl.acm.org/doi/pdf/10.1145/3394277.3401854

- Publication type: conference paper
- Conference ranking: none
- Author's participation: 70%

Author's Contribution

Design of k-Dispatch, its core, database and partially REST API used for communication with user applications. Implementation and testing of all these modules. Creating definitions of workflows and their execution model. Deployment of k-Dispatch on a remote server. Creating figures and writing the paper.

Abstract

Therapeutic ultrasound is increasingly being used for applications in oncology, drug delivery, and neurostimulation. In order to adapt the treatment procedures to patient needs, complex physical models have to be evaluated prior to the treatment. These models, however, require intensive computations that can only be satisfied by cloud and HPC facilities. Unfortunately, employing these facilities and executing the required computations is not straightforward even for experienced developers. k-Dispatch is a novel workflow management system aimed at modelling biomedical ultrasound procedures using the open-source k-Wave acoustic toolbox. It allows ultrasound procedures to be uploaded with a single click and provides a notification when the result is ready for download. Inside k-Dispatch, there is a complex workflow management system which decodes the workflow graph, optimizes the workflow execution parameters, submits jobs to remote computing facilities, monitors their progress, and logs the consumed core hours. In this paper, the architecture and deployment of k-Dispatch are discussed, including the approach used for workflow optimization. A key innovation is the use of previous performance data to automatically select the utilised hardware and execution parameters. A review of related work is also given, including workflow management systems, batch schedulers, and cluster simulators.

6.1.3 Publication III

JAROS Marta and JAROS Jiri. **Performance-Cost Optimization of Moldable Scientific Workflows.** In: Job Scheduling Strategies for Parallel Processing. Lecture Notes in Computer Science (including subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics), vol. 12985. Portland, Oregon USA: Springer International Publishing, 2021, pp. 149-167. ISBN 978-3-030-88223-5. Available from: https://link.springer.com/book/10.1007%2F978-3-030-88224-2

- Publication type: conference paper
- Conference ranking: B (ERA), B1 (Qualis), B (CORE2020, CORE2021)
- Author's participation: 80%

Author's Contribution

Design and implementation of the presented algorithm, solution encoding and fitness function. Preparing and running experiments on the cluster, results post-processing and evaluation. Creating figures and writing paper parts regarding the algorithm, experiments, their evaluation and conclusions.

Abstract

Moldable scientific workflows represent a special class of scientific workflows where the tasks are written as distributed programs being able to exploit various amounts of computer resources. However, current cluster job schedulers require the user to specify the number of resources per task manually. This often leads to suboptimal execution time and related cost of the whole workflow execution since many users have only limited experience and knowledge of parallel efficiency and scaling. This paper proposes several mechanisms to automatically optimize the execution parameters of moldable workflows using genetic algorithms. The paper introduces a local optimization of workflow tasks, a global optimization of the workflow on systems with on-demand resource allocation, and a global optimization for systems with static resource allocation. Several objectives including the workflow makespan, computational cost and the percentage of idling nodes are investigated together with a trade-off parameter putting stress on one objective or another. The paper also discusses the structure and quality of several evolved workflow schedules and the possible reduction in makespan or cost. Finally, the computational requirements of the evolutionary process together with the recommended genetic algorithm settings are investigated. The most complex workflows may be evolved in less than two minutes using the global optimization while in only 14s using the local optimization.

6.1.4 Publication IV

JAROS Marta, SASAK Tomas, TREEBY Bradley E. and JAROS Jiri. **Estimation of Execution Parameters for k-Wave Simulations.** In: High Performance Computing in Science and Engineering. HPCSE 2019. Lecture Notes in Computer Science (including subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics). Cham: Springer Nature Switzerland AG, 2021, pp. 116-134. ISBN 978-3-030-67076-4. Available from: https://link.springer.com/chapter/10.1007/978-3-030-67077-1_7

- Publication type: conference paper
- Conference ranking: none
- Author's participation: 70%

Author's Contribution

Post-processing of collected performance data, providing experiments with different interpolation methods and different inputs. Evaluation of experiments. Creating figures, plots and writing the paper.

Abstract

Estimation of execution parameters takes centre stage in the automatic offloading of complex biomedical workflows to the cloud and high-performance facilities. Since ordinary users have no or very limited knowledge of the performance characteristics of particular tasks in the workflow, the scheduling system has to have the capability to select the appropriate amount of compute resources, e.g., compute nodes, GPUs, or processor cores and estimate the execution time and cost.

The presented approach considers a fixed set of executables that can be used to create custom workflows and collects performance data of successfully computed tasks. Since the workflows may differ in the structure and size of the input data, the execution parameters can only be obtained by searching the performance database and interpolating between similar tasks. This paper shows it is possible to predict the execution time and cost with high confidence. If the task parameters are found in the performance database, the mean interpolation error stays below 2.29%. If only similar tasks are found, the mean interpolation error may grow up to 15%. Nevertheless, this is still an acceptable error since the cluster performance may vary in order of percent as well.

6.1.5 Publication V

JAROS Marta and JAROS Jiri. **Optimization of Execution Parameters of Moldable Workflows under Incomplete Performance Data.** In: Klusáček, D., Julita, C., Rodrigo, G.P. (eds) Job Scheduling Strategies for Parallel Processing. JSSPP 2022. Lecture Notes in Computer Science, vol 13592. Springer, Cham. https://doi.org/10.1007/978-3-031-22698-4_8.

- Publication type: conference paper
- Conference ranking: B (ERA), B1 (Qualis), B (CORE2020, CORE2021)
- Author's participation: 80%

Author's Contribution

Preparing and running experiments on the cluster and their evaluation. Implementation of improvements in Tetrisator, the cluster simulator. Creating figures, and plots and writing the paper parts about the problem description, experiments done, obtained results together with their evaluation, conclusions and future work.

Abstract

Complex ultrasound workflows calculating the outcome of ultrasound procedures such as neurostimulation, tumour ablation or photoacoustic imaging are composed of many computational tasks requiring high-performance computing or cloud facilities to be computed in a sensible time. Most of these tasks are written as moldable parallel programs being able to run across various numbers of compute nodes. The number of compute nodes assigned to particular tasks strongly affects the overall execution and queuing times of the whole workflow (makespan) as well as the total computational cost. This paper employs a genetic algorithm searching for a good resource distribution over particular tasks, and a cluster simulator evaluating the makespan and cost of the candidate execution schedules. Since the exact execution time cannot be measured for every possible combination of the task, input data size, and assigned resources, several interpolation techniques are used to predict the task duration for a given amount of compute resources. The best execution schedules are eventually submitted to a real cluster with a PBS scheduler to validate the whole technique. The experimental results confirm the proposed cluster simulator corresponds to a real PBS job scheduler with sufficient fidelity. The investigation of the interpolation techniques showed that incomplete performance data can be successfully completed by linear and quadratic interpolations making a maximum mean error below 10%. Finally, the paper shows it is possible to implement a user-defined parameter which instructs the genetic algorithm to prefer either the makespan or cost or find a suitable trade-off.

6.2 Other Publications

2022

- **JAROS Marta**, TREEBY Bradley E. and JAROS Jiri. **k-Plan: from the Hospital to the Cluster and back.** 6th Users' Conference of IT4Innovations. Ostrava, CZ, 2022.
  - Publication type: poster
  - Author's participation: 50%
- **JAROS Marta**, TREEBY Bradley E. and JAROS Jiri. **k-Dispatch's Performance Modules for Advanced Workflow Submission**. High Performance Computing in Science and Engineering 2022, Hotel Soláň, CZ, 2022.
  - Publication type: poster
  - Author's participation: 80%

2021

- **JAROS Marta**, TREEBY Bradley E. and JAROS Jiri. **Adaptive Execution Planning in Biomedical Workflow Management Systems.** The Platform for Advanced Scientific Computing (PASC) Conference 2021. Geneva, CH, 2021.
  - Publication type: poster
  - Author's participation: 80%
- **JAROS Marta** and JAROS Jiri. **k-Dispatch's Performance Modules for Advanced Workflow Submission.** 5th Users' Conference of IT4Innovations. Ostrava, CZ, 2021. Available from: https://events.it4i.cz/event/119/contributions/410/
  - Publication type: poster
  - Author's participation: 80%

2020

- KUKLIS Filip, **JAROS Marta** and JAROS Jiri. **Accelerated Design of HIFU Treatment Plans Using Island-based Evolutionary Strategy.** In: Applications of Evolutionary Computation. Lecture Notes in Computer Science (including subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics), vol. 12104. Cham: Springer International Publishing, 2020, pp. 463-478. ISBN 978-3-030-43721-3. Available from: https://link.springer.com/chapter/10.1007/978-3-030-43722-0_30
  - Publication type: conference paper
  - Conference rating: B5 (Qualis), B (CORE2021)
  - Author's participation: 15%

2019

- **JAROS Marta**, TREEBY Bradley E. and JAROS Jiri. **Adaptive Execution Planning in Biomedical Workflow Management Systems.** 3rd Users‘ Conference of IT4Innovations, Ostrava, CZ, 2019.
  - Publication type: poster
  - Author's participation: 75%
- **JAROS Marta**. **Adaptive Execution Planning in Workflow Management Systems.** In: Pocitacove architektury a diagnostika 2019. Doksy: Academic and Medical Conference Agency, CZ, 2019, pp. 23-26. ISBN 978-80-88214-20-5.
  - Publication type: conference paper
  - Conference ranking: none
  - Author's participation: 100%
- **JAROS Marta**, JAROS Jiri and TREEBY Bradley E. **Adaptive Execution Planning in Workflow Management Systems.** SuperComputing 2019, Denver, Colorado, US, 2019.
  - Publication type: poster
  - Author's participation: 75%
- **JAROS Marta**, TREEBY Bradley E. and JAROS Jiri. **Scientific workflow management framework**. High Performance Computing in Science and Engineering 2019, Hotel Soláň, CZ, 2019.
  - Publication type: poster
  - Author's participation: 75%

2018

- **CUDOVA Marta**, TREEBY Bradley E. and JAROS Jiri. **Design of HIFU Treatment Plans using Evolutionary Strategy**. In: GECCO 2018 Companion - Proceedings of the 2018 Genetic and Evolutionary Computation Conference Companion. Kyoto: Association for Computing Machinery, 2018, pp. 1568-1575. ISBN 978-1-4503-5764-7. Available from: https://dl.acm.org/citation.cfm?doid=3205651.3208268
  - Publication type: conference paper
  - Conference ranking: none
  - Author's participation: 50%
- **JAROS Marta**, TREEBY Bradley E. and JAROS Jiri. **Design of HIFU Treatment Plans using an Evolutionary Strategy**. 2nd Users Conference of IT4Innovations, Ostrava, CZ, 2018.
  - Publication type: poster
  - Author's participation: 75%

- **JAROS Marta**. **Scientific Workflows Management**. In: Pocitacove architektury & diagnostika PAD 2018. Pilsen: University of West Bohemia in Pilsen, CZ, 2018, pp. 25-28. ISBN 978-80-261-0814-6.
  - Publication type: conference paper
  - Conference ranking: none
  - Author's participation: 100%

2017

- **CUDOVA Marta** and JAROS Jiri. **Framework for Planning, Executing and Monitoring Cooperating Computations.** 1st Users Conference of IT4Innovations, Ostrava, CZ, 2017.
  - Publication type: poster
  - Author's participation: 80%
- **CUDOVA Marta**. **Framework for Planning, Running and Monitoring Cooperating Computations**. In: Pocitacove architektury & diagnostika PAD 2017. Bratislava: Slovak University of Technology in Bratislava, SK, 2017, pp. 20-23. ISBN 978-80-972784-0-3.
  - Publication type: conference paper
  - Conference ranking: none
  - Author's participation: 100%

6.3 Publications Being Prepared

- BUCHTA Martin, **JAROS Marta** and JAROS Jiri. **Using Symbolic Regression in Incomplete Performance Data for Optimization of Workflow Scheduling**.
  - Publication type: conference paper
  - To be submitted at: WORKS 2023 (workshop at Supercomputing) or JSSPP 2024 (workshop at IPDPS)
  - Author's participation: 30%
- TREEBY Bradley E., **JAROS Marta** and JAROS Jiri. **k-Plan: Advanced Modelling Tool for Ultrasound Planning**.
  - Publication type: journal paper
  - Author's participation: 20%

6.4 Fellowships, Memberships, Awards and Others

- A year fellowship as a Research Assistant at the Department of Medical Physics and Biomedical Engineering, University College London, London, UK, 2021.
- Three-year membership of the faculty senate, Faculty of Information Technology, Brno University of Technology, CZ, 2019–2022.
- Presentation at the PRACE Birds of Feathers at ISC High Performance, DE 2021.
- Invited talk at the University College London at the Social Tech event, UK, 2021.
- Invited talks at the Girls Summer School at the Faculty of Information Technology, Brno University of Technology, CZ, 2020–2021.
- Contribution at the Women in HPC (WHPC) blog about k-Dispatch, 2017. Accessible from https://womeninhpc.org/hpc/framework-planning.
- Poster at Women in HPC (WHPC) workshop at Supercomputing 2017, Denver, USA. See https://womeninhpc.org/women-in-hpc-events/whpc-reflections.
- Talk at the PRACE booth at Supercomputing 2017, Denver, USA.
- Winner of the PRACE Summer of HPC Ambassador Award, 2016. See https://prace-ri.eu/about/prace-awards/sohpcprize/sohpcprize-2016/.
- Two-month internship PRACE Summer of HPC at the supercomputing center EPCC, University of Edinburgh, UK, 2016. Working on Parallelising of Scientific Python Applications on Archer, supervised by Dr. Neelofer Banglawala. See https://youtu.be/__oNjB6DIi8.
- The PAD (Počítačové architektury a diagnostika) workshop co-organization, CZ, 2016.

6.5 Research Projects and Grants

2023

- Application-specific HW/SW architectures and their applications, BUT, FIT-S-23-8141, 2023-2025, running, start: 2023-03-01, end: 2025-12-31.

2022

- Closed-loop Individualized image-guided Transcranial Ultrasonic Stimulation, EC EU - HORIZON EUROPE, 101071008, 101071008, 2022-2026, running, start: 2022-10-01, end: 2026-09-30
- Acceleration of selected evolutionary computing techniques for solving global optimization problems., BUT, FIT/FSI-J-22-7980, 2022, completed, start: 2022-03-01, end: 2022-12-31.

2021

- Acceleration of Selected Evolutionary Communication Techniques for Solving combinatoric NP-complete tasks, BUT, FIT/FSI-J-21-7435, 2021, completed, start: 2021-03-01, end: 2021-12-31, **principal investigator**.
- Analysis of the applicability of Angle of Arrival technology in practice, ADWITECH system s.r.o., 2021-2022, completed, start: 2021-10-01, end: 2022-02-28

2020

- Design, Optimization and Evaluation of Application Specific Computer Systems, BUT, FIT-S-20-6309, 2020-2022, completed, start: 2020-03-01, end: 2022-12-31.

2017

- Advanced parallel and embedded computer systems, BUT, FIT-S-17-3994, 2017-2019, completed, start: 2017-03-01, end: 2019-12-31.
- Moderní a otevřené studium techniky (MOST), MŠMT CR, 2017-2022, completed, start: 2017-09-01, end: 2022-12-31.
- Photoacoustic/Ultrasound Mammoscopy for evaluating screening-detected lesions in the breast, EC EU, PAMMOTH, PAMMOTH, 2017-2021, completed, start: 2017-01-01, end: 2021-06-30.

2016

- IT4Innovations excellence in science, MŠMT CR, LQ1602, 2016-2020, completed, start: 2016-01-01, end: 2020-12-31.

Chapter 7

Conclusions

This chapter summarizes and discusses the contributions and outcomes of this Thesis for the research community and industry. Finally, future research directions are proposed.

The main objectives of this Thesis were:

1. *Investigate multi-objective optimization methods in order to select a suitable amount of resources, e.g., nodes, for particular tasks in workflows.*

   Multi-objective optimization methods were thoroughly studied together with approaches adopted by high-performance computing (HPC) centres, workload managers, processing frameworks and workflow management systems. The summary may be found in Sec. 2 and other details are provided in related papers attached.

2. *Collect and create performance datasets and identify experimental use cases to demonstrate the selected optimization method.*

   Biomedical workflows employing the state-of-the-art k-Wave toolbox were selected as experimental use cases. The selection was motivated by PAMMOTH, the European Horizon 2020 project, requiring routine execution of ultrasound simulations to reconstruct photoacoustic images of the breast, and several other research projects in the areas of ultrasound-based neurostimulation, and is further developed in CITRUS, an EC EU - Horizon Europe project in the area of ultrasound neurostimulation. The k-Wave performance data was collected on various IT4Innovations clusters. Detailed description can be found in Sec. 4.3. Experimental workflows, their computational demands on different clusters and collected performance data are also explained in the following related papers: Paper I gives a detailed description of computational demands of different code types, measured for the IT4Innovations' Anselm cluster, executed within the neurostimulation workflow and shows their strong scaling characteristics. Paper II presents structures of two real-world biomedical workflows. Paper III introduces two experimental workflows together with the performance data collected on the IT4Innovations' Barbora cluster for the distributed MPI version of the k-Wave toolbox. Paper IV reveals performance data for the shared memory OpenMP-based k-Wave toolbox simulations measured using IT4Innovations' Anselm cluster. Finally, Paper V depicts differences between k-Wave performance data obtained for Salomon and Barbora clusters and describes experimental datasets together with experimental workflows.

3. *Provide an effective workflow execution planning with the throughput maximization or the latency minimization.*

   After a thorough investigation of related solutions in Sec. 2, the genetic algorithms-based approach was selected and implemented. The crucial part of multi-objective optimization is the fitness function. We designed three different fitness functions providing: (1) Local task-level optimization selecting optimal execution parameters for each individual task in the workflow and considering only one optimization criterion. This approach minimizes the execution time per each task but the overall execution time (makespan) may not be the fastest one due to longer waiting times for free resources. (2) Global workflow-level optimization using on-demand resource allocations selecting execution parameters with respect to the whole workflow. The selected parameters may, thus, be suboptimal for individual tasks. This fitness function optimizes two contrary optimization criteria, i.e., workflow makespan and cost. Therefore, a trade-off coefficient to prioritize between them was introduced. Using this approach, time or cost-constrained solutions can be found as well as differently balanced solutions. Finally, (3) Global (workflow-level) optimization using static resource allocations finding solutions that better utilize allocated resources and minimizes latency. Unlike (2), no trade-off coefficient is used.

   The fitness function is encapsulated by the genetic algorithm. The fitness function is, however, universal enabling the genetic algorithm to be easily replaced by different optimization methods. The encapsulating method represents a black box inside which the execution parameters are selected based on the task input and collected performance data in history. The optimization heuristics and fitness functions are thoroughly described in Sec. 5.1 and Papers III and V.

   Since the collected performance data may be incomplete or missing for a given input, interpolation methods were incorporated. They estimate the missing values using similar known values in their neighbourhood. Section 5.2 and Papers IV and V deal with this topic.

   Moreover, this Thesis introduced early experiments where symbolic regression is employed and gave very promising results. This is, however, the subject of future research. A description of this approach together with the provided experiments can, however, be found in Appendix C.

4. *Create a piece of software providing the execution and monitoring of the workflows on HPC systems, and data transfers. This software parses the input data, automatically assembles a corresponding workflow, sets execution parameters and generates HPC system-specific jobs scripts. Required data is transferred to the target HPC system where the execution is orchestrated. The remote jobs are monitored and obtained result data is transferred back.*

   For this purpose, k-Dispatch providing HPC as a Service was designed, developed and successfully deployed. k-Dispatch currently utilizes Czech and UK computational resources and is used by users from both academia and industry. It should be noted, that k-Dispatch has also been commercialized and is being offered by the BrainBox, Ltd. company. k-Dispatch and its functionality is described in Sec. 4.1 and published in Paper II. Its deployment and real usage are then defined in Sec. 7.1.

5. *Experimentally evaluate the implemented optimization method and software on selected use cases.*

   The genetic algorithm-based optimization method was experimentally evaluated (1) in the custom-created cluster and batch scheduler simulator called Tetrisator (see Paper III and V), (2) in Alea, the complex and general cluster simulator (see Paper I), and (3) on real IT4Innovations' HPC clusters and static resource allocations (see Paper III and V). The experimental results were summarized in Sec. 5.4.

   k-Dispatch was tested and evaluated in various aspects: inputs processing and decoding, data transfers, remote job script generation, executions orchestration, job and cluster monitoring, status reporting, RESTful API communication, and fault tolerance. A set of Python unit tests was created to test the key low-level features. Typical and more complex scenarios covering a submission and orchestration of executions, monitoring, detection of error states, etc., were tested manually and experimentally evaluated using the IT4Innovations' HPC clusters.

6. *Asses the implemented solution, its benefits discuss open issues.*

   We designed, developed and deployed a piece of software to automatize and offload computational workflows to remote computing facilities. We also designed and developed optimization heuristics to select a suitable amount of computational resources (e.g., nodes) for individual tasks in the workflow. We designed several fitness functions to satisfy different optimization criteria (e.g., overall execution time, cost, computational throughput). The execution plans show how workflows are executed using particular remote computing resources. The time needed to create an execution plan is dependent on the size of the workflow. We managed to create these plans in under two minutes with a success rate higher than 90% for workflows consisting of up to 64 tasks. The evaluation of these plans answers how well the optimization criteria would be met. The presented solution provides automated and failure-free execution of workflows.

Auxiliary objectives were to

1. *design and implement additional features within the developed software, e.g., accounting, authentication and authorization, reporting, and a level of fault tolerance,*
2. *store performance data after each successful run in order to build a custom performance database,*
3. *design and implement a RESTful (Representational State Transfer) API (Application Programming Interface) together with a GUI (Graphical User Interface) application to handle workflow submissions easily.*

All auxiliary objectives were fulfilled. k-Dispatch, as an HPC as a Service tool, shares features with other state-of-the-art workflow management tools. It deploys its own database of users, allocated resources, remote computing facilities and available executable binaries. This database is used for execution planning, accounting, authentication and authorization. Designed RESTful API enables unified communication with various user applications, and can read from and write to the database. To provide a quality of service, mechanisms providing fault tolerance and reporting are implemented. To provide adaptive and optimized workflow executions, the performance database is being collected. New records are added after each successful workflow completion. Details can be found in Chpt. 6 and Paper II.

The approaches presented in this text are applicable to anybody who is interested in automated task or workflow execution and scheduling. This Thesis gives ideas and offers means how to (1) automate daily tasks, (2) improve workflow executions and better utilize computational resources, and (3) handle constrained executions. Steps needed to obtain a general solution are depicted in Sec. 4.4. Designed fitness functions can be used generally by various optimization and machine learning methods. This approach may be adjusted or extended, e.g., to optimize different or multiple variables alongside. It also lists open problems and challenges (see Sec. 3.1 and 3.2) in this area that may help new researchers and developers to react to nowadays issues. Presented techniques build a bridge between users and computing facilities that helps to employ them in industry, academia and various research fields. Moreover, a middle-ware software k-Dispatch introduced has been successfully deployed in clinics and represents an HPC as a Service tool.

7.1 k-Dispatch Development and Deployment

k-Dispatch is a Python-based tool. Due to clinical usage, its development was performed under strict rules, properly tested, and versioned in a GitLab repository. Each new functionality or improvement was described within the GitLab's Issue system and supplied with a log of changes. Every code update or new feature has to be properly tested before being merged into the main development branch. Strict rules also concern used binaries and data. Due to clinical usage, only certified binaries can be employed in the workflows. All data is anonymized and stored temporarily only for the necessary period of time.

A container-based approach was used to simplify the process of deployment, maintenance, data safety and fault tolerance. We used Docker [8], an open-source platform based on Linux containers, which has become very popular for modern software development and deployment. The biggest advantage of this solution is the isolation of k-Dispatch and its dependencies into self-contained units that can run anywhere. k-Dispatch is split into four Docker containers and three volumes connected via a Docker network deployed by the docker-compose tool, as depicted in Fig. 7.1. These containers individually encapsulate the Dispatch database (depicted in yellow), the Dispatch core together with the k-Dispatch's application server Web Server (depicted in pink), and additionally, an Nginx-based web server [18] acting as an entry point for user requests (depicted in green). The Dispatch database stores all persistent data in a dedicated volume. The remaining volumes store k-Dispatch Python source codes and SSH credentials to remote computational facilities, respectively. These volumes are shared between the Dispatch core and the application server to enable easy data updates. The application web server employing the Flask framework cannot be run in the production version without another gateway, e.g., GUnicorn[1] or other uWSGI[2] hosting services, since they only offer HTTP communication. The Nginx container thus adds the required security by providing HTTPS communication. Moreover, the website API is secured using the Let's Encrypt[3] certificates.

[1] https://gunicorn.org/
[2] https://flask.palletsprojects.com/en/1.0.x/deploying/uwsgi/
[3] https://letsencrypt.org/

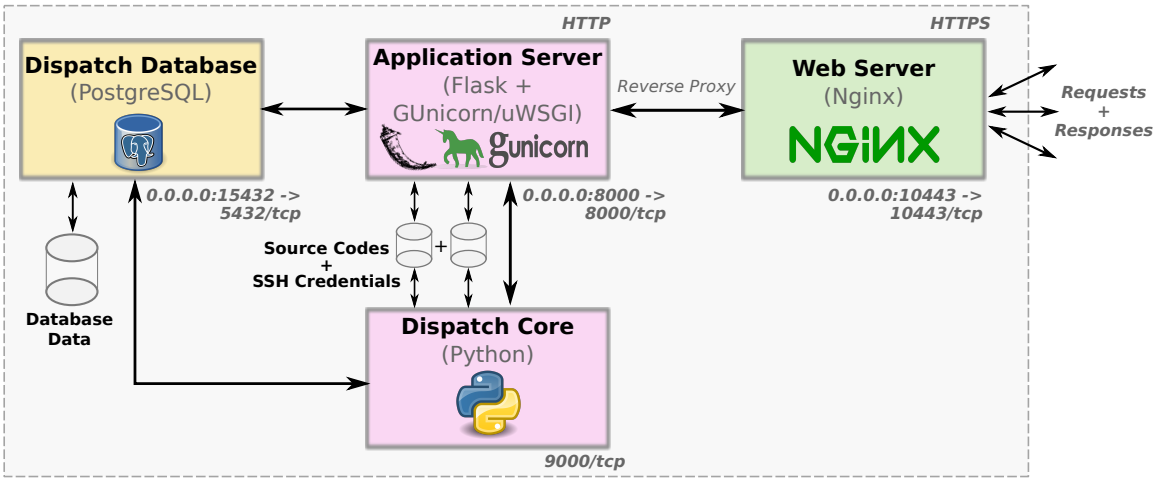


Figure 7.1: k-Dispatch architecture wrapped into Docker containers and volumes. The text descriptions below individual rectangles show network port mappings used in the current solution.

7.1.1 k-Plan: Advanced Modelling Tool for Ultrasound Planning

k-Plan[4] is an advanced modelling tool for planning transcranial ultrasound stimulation (TUS) procedures. k-Plan uses a streamlined and intuitive workflow that allows users to select an ultrasound device, position the device using a template or medical image, and specify the sonication parameters. k-Dispatch here plays a key role in allowing access to high-performance computing resources in the cloud enabling users to run high-resolution planning simulations with a single click. k-Dispatch optimises the computing resources needed for every simulation and monitors the running simulations. Their states are automatically refreshed and displayed in the plan browser. Additional services like accounting, data transfers, fault tolerance, etc., are performed under the hood as well.

Today, k-Plan has 35 active users from both academia and industry that simulate their problems regularly using this service. Over the last six months, users have submitted over 1400 workflows with typical sizes ranging from $37\times69\times69$ to $1095\times954\times704$, and consumed over 3000 node hours on both Barbora and Karolina supercomputers at IT4Innovations.

Development and exhaustive testing could be possible owing to the national supercomputing center IT4Innovations. Required node hours were gained in periodical open access grant competitions.[5] This work was supported by the following projects: OPEN-17-6, OPEN-22-47, OPEN-25-43, OPEN-25-25, OPEN-25-26 and OPEN-27-69.

7.2 Future Work

Science and research define humanity, transform our society for the better, and push our intellectual borders even further. Of course, the work presented here is not at its end. This work could not answer all of the found open problems and implement all ideas and approaches due to their complexity. Each day gives us new opportunities to follow up with the research done here.

[4] https://brainbox-neuro.com/products/k-plan
[5] https://www.it4i.cz/en/for-users/open-access-competition

Some ways of further research have already been indicated in this text. We may split the future work into two groups, which are research goals, and application improvements and extensions.
Research goals mainly follow up with the commenced research here and aim at their (commercial) usage in a broader community:

- Complete and publish experiments with genetic programming and symbolic regression to estimate task execution time for unknown input data and/or amount of computing resources. Symbolic regression allows to better capture the scaling behaviour, and thus, allows better predictions.
- Include the information about the actual remote facility utilization. This is particularly handy in on-demand allocations where users compete with each other. In such a case, tasks and their character coming to the remote system are unpredictable. Together with a changing workload, user priorities change as well. Waiting times in computational queues are directly affected by these priorities. Waiting for more resources usually implies waiting in queues for a longer time. Therefore, it is beneficial to ask for fewer resources and try to fill in the currently idling resources in the schedule. This solution may not be so price advantageous because the execution time gets longer, however, the throughput is better. This is crucial for time-constrained workflows.
- Update Evaluator to allow new scheduling strategies. Since different computing facilities based on different technologies are to be used, it is essential to employ a tool that can dynamically simulate the target system. It is also crucial to follow a modular design to flexibly react to any change that the future may bring.

On the other hand, application improvements and extensions react to users' needs and try to keep pace with nowadays technologies and requirements:

- Integrate the latest advanced planning methods introduced in this Thesis and enable users to prioritize one optimization criterion over another on their own for on-demand allocations. This may be implemented using a graphical slider, internally represented as a trade-off coefficient.
- Enable custom workflows together with a user-friendly workflow editor. This step helps k-Dispatch to keep pace with traditional workflow management systems. Creating custom workflows and their sharing in a standard format is, however, a great opportunity for (slightly) advanced users to optimize their routine tasks.
- Enable user binary executables. This step also helps k-Dispatch to keep pace with traditional workflow management systems. Allowing user binaries may be considered a security hazard but running them encapsulated, e.g., in a Docker container may solve this issue. Unlike in the current k-Dispatch, users would be responsible for their binaries, their reliability and efficiency.
- Employ Slurm-based computing facilities and clouds, e.g., Amazon Cloud.[6]

[6] https://aws.amazon.com/

Bibliography

[1] Abbas, M. A., Coussios, C. C. and Cleveland, R. O. Patient specific simulation of HIFU kidney tumour ablation. In:. July 2018, vol. 2018, p. 5709–5712. DOI: 10.1109/EMBC.2018.8513647. ISSN 1558-4615.

[2] Aldrich, J. E. Basic physics of ultrasound imaging. *Critical Care Medicine*. 2007, vol. 35, p. S131–S137.

[3] Amdahl, G. M. Validity of the Single Processor Approach to Achieving Large Scale Computing Capabilities. In: *Proceedings of the April 18-20, 1967, Spring Joint Computer Conference*. New York, NY, USA: Association for Computing Machinery, 1967, p. 483–485. AFIPS '67 (Spring). DOI: 10.1145/1465482.1465560. ISBN 9781450378956.

[4] Bergstra, J. and Bengio, Y. Random Search for Hyper-Parameter Optimization. *Journal of Machine Learning Research*. 2012, vol. 13, no. 10, p. 281–305. DOI: 10.5555/2188385.2188395. ISSN 1532-4435.

[5] Bhagat, J., Tanoh, F., Nzuobontane, E., Laurent, T., Orlowski, J. et al. BioCatalogue: a universal catalogue of web services for the life sciences. *Nucleic Acids Research*. may 2010, vol. 38, suppl_2, p. W689-W694. DOI: 10.1093/nar/gkq394. ISSN 03051048.

[6] Bharathi, S., Chervenak, A., Deelman, E., Mehta, G., Su, M.-H. et al. Characterization of scientific workflows. In:. IEEE, November 2008, p. 1–10. DOI: 10.1109/WORKS.2008.4723958. ISBN 978-1-4244-2827-4.

[7] Bleuse, R., Hunold, S., Kedad Sidhoum, S., Monna, F., Mounie, G. et al. Scheduling Independent Moldable Tasks on Multi-Cores with GPUs. *IEEE Transactions on Parallel and Distributed Systems*. september 2017, vol. 28, no. 9, p. 2689–2702. DOI: 10.1109/TPDS.2017.2675891.

[8] Boettiger, C. An Introduction to Docker for Reproducible Research. *ACM SIGOPS Operating Systems Review*. ACM. 2015, vol. 49, no. 1, p. 71–79. DOI: 10.1145/2723872.2723882.

[9] Buyya, R. and Murshed, M. GridSim: A Toolkit for the Modeling and Simulation of Distributed Resource Management and Scheduling for Grid Computing. *Concurrency Computat.: Pract. Exper.* Chichester, UK: John Wiley & Sons, Ltd. october 2002, vol. 14, 13-15, p. 1175–1220. DOI: 10.1002/cpe.710. ISSN 1532-0626.

[10] Chaplin, V., Phipps, M. and Caskey, C. A random phased-array for MR-guided transcranial ultrasound neuromodulation in non-human primates. *Physics in Medicine and Biology*. december 2017, vol. 10, p. 105016. DOI: 10.1088/1361-6560/aabeff.

[11] Chirkin, A. M., Belloum, A. S., Kovalchuk, S. V., Makkes, M. X., Melnik, M. A. et al. Execution time estimation for workflow scheduling. *Future Generation Computer Systems*. 2017, vol. 75, p. 2017. DOI: 10.1016/j.future.2017.01.011. ISSN 0167-739X.

[12] Cima, V., Böhm, S., Martinovič, J., Dvorský, J., Janurová, K. et al. HyperLoom: A platform for defining and executing scientific pipelines in distributed environments. In: ACM. *Proceedings of the 9th Workshop and 7th Workshop on Parallel Programming and RunTime Management Techniques for Manycore Architectures and Design Tools and Architectures for Multicore Embedded Computing Platforms*. 2018, p. 1–6. DOI: 10.1145/3183767.3183768. ISBN 9781450364447.

[13] Cobbold, R. S. C. *Foundantions of Biomedical Ultrasound*. Oxford University Press, 2007. ISBN 9780195168310.

[14] Deelman, E., Vahi, K., Rynge, M., Mayani, R., Silva, R. Ferreira da et al. The Evolution of the Pegasus Workflow Management Software. *Computing in Science Engineering*. IEEE. 2019, vol. 21, no. 4, p. 22–36. DOI: 10.1109/MCSE.2019.2919690.

[15] Dutot, P.-F., Netto, M. A. S., Goldman, A. and Kon, F. Scheduling Moldable BSP Tasks. In: Feitelson, D., Frachtenberg, E., Rudolph, L. and Schwiegelshohn, U., ed. *Job Scheduling Strategies for Parallel Processing*. Berlin, Heidelberg: Springer Berlin Heidelberg, 2005, p. 157–172. ISBN 978-3-540-31617-6.

[16] Feitelson, D. G. and Rudolph, L. Toward convergence in job schedulers for parallel supercomputers. In: *Lecture Notes in Computer Science*. 1996, p. 1–26. DOI: 10.1007/BFb0022284.

[17] Feitelson, D. G., Rudolph, L., Schweigelshohn, U., Sevcik, K. and Wong, P. Theory and Practice in Parallel Job Scheduling. In: Feitelson, D. G. and Rudolph, L., ed. *Job Scheduling Strategies for Parallel Processing*. Springer-Verlag, 1997, vol. 1291, p. 1–34. Lecture Notes in Computer Science. DOI: 10.1007/3-540-63574-2_14. ISBN 978-3-540-70710-3.

[18] Fjordvald, M. and Nedelcu, C. *Nginx HTTP Server - Fourth Edition: Harness the Power of Nginx to Make the Most of Your Infrastructure and Serve Pages Faster Than Ever Before*. 4thth ed. Packt Publishing, 2018. ISBN 978-1788623551.

[19] Foster, I. Globus Toolkit Version 4: Software for Service-Oriented Systems. *Journal of Computer Science and Technology*. july 2006, vol. 21, no. 4, p. 513–520. DOI: 10.1007/s11390-006-0513-y.

[20] Foster, I. and Kesselman, C. Computational Grids. In: Palma, J. M. L. M., Dongarra, J. and Hernández, V., ed. *Vector and Parallel Processing — VECPAR 2000*. Berlin, Heidelberg: Springer Berlin Heidelberg, 2001, p. 3–37. ISBN 978-3-540-44942-3.

[21] Frigo, M. and Johnson, S. G. The design and implementation of FFTW3. *Proc. IEEE*. 2005, vol. 93, no. 2, p. 216–231.

[22] Gad, A. F. *PyGAD: An Intuitive Genetic Algorithm Python Library*. 2021.

[23] Garzón, W., Benavides, L., Gaignard, A., Redon, R. and Südholt, M. A taxonomy of tools and approaches for distributed genomic analyses. *Informatics in Medicine Unlocked*. 2022, vol. 32, p. 101024. DOI: 10.1016/j.imu.2022.101024. ISSN 2352-9148.

[24] Grisey, A., Yon, S., Letort, V. and Lafitte, P. Simulation of high-intensity focused ultrasound lesions in presence of boiling. *Journal of Therapeutic Ultrasound*. 2016. DOI: 10.1186/S40349-016-0056-9. ISSN 20505736.

[25] Groen, D., Bhati, A. P., Suter, J., Hetherington, J., Zasada, S. J. et al. FabSim: Facilitating computational research through automation on large-scale and distributed e-infrastructures. *Computer Physics Communications*. Elsevier B.V. 2016, vol. 207, p. 375–385. DOI: 10.1016/j.cpc.2016.05.020. ISSN 00104655.

[26] Gu, L. and Li, X. DFT Performance Prediction in FFTW. In: Gao, G. R., Pollock, L. L., Cavazos, J. and Li, X., ed. *Languages and Compilers for Parallel Computing*. Berlin, Heidelberg: Springer Berlin Heidelberg, 2010, p. 140–156. ISBN 978-3-642-13374-9.

[27] Hejazi, S. and Saghafian, S. Flowshop-scheduling problems with makespan criterion: a review. *International Journal of Production Research*. Taylor & Francis. Jul 2005, vol. 43, no. 14, p. 2895–2929. DOI: 10.1080/00207540500056417. ISSN 1366-588X.

[28] Huang, K.-C., Huang, T.-C., Tsai, M.-J. and Chang, H.-Y. Moldable Job Scheduling for HPC as a Service. In: Park, J. J. J. H., Stojmenovic, I., Choi, M. and Xhafa, F., ed. *Future Information Technology*. Berlin, Heidelberg: Springer Berlin Heidelberg, 2014, p. 43–48. ISBN 978-3-642-40861-8.

[29] Izadkhah, H. Learning based genetic algorithm for task graph scheduling. *Applied Computational Intelligence and Soft Computing*. 2019, vol. 2019, p. 2019. DOI: 10.1155/2019/6543957.

[30] Jackson, S. E. and Chester, J. D. *Personalised cancer medicine*. 2015. DOI: 10.1002/ijc.28940. ISSN 10970215.

[31] Jansen, K. and Land, F. Scheduling Monotone Moldable Jobs in Linear Time. In: *2018 IEEE International Parallel and Distributed Processing Symposium (IPDPS)*. IEEE, May 2018, p. 172–181. DOI: 10.1109/IPDPS.2018.00027. ISSN 1530-2075.

[32] Jaros, J., Rendell, A. P. and Treeby, B. E. Full-wave Nonlinear Ultrasound Simulation on Distributed Clusters with Applications in High-Intensity Focused Ultrasound. *International Journal of High Performance Computing Applications*. SAGE Publications. 2015, vol. 30, no. 2, p. 137–155. DOI: 10.1177/1094342015581024. ISSN 1741-2846.

[33] Klusacek, D., Toth, S. and Podolnikova, G. Complex Job Scheduling Simulations with Alea 4. In: *Proceedings of the 9th EAI International Conference on Simulation Tools and Techniques*. Brussels, BEL: ICST (Institute for Computer Sciences, Social-Informatics and Telecommunications Engineering), 2016, p. 124–129. SIMUTOOLS'16. ISBN 9781631901201.

[34] Laarhoven, P. J. M. van and Aarts, E. H. L. *Simulated Annealing: Theory and Applications*. Dordrecht: D. Reidel, 1987. ISBN 90-277-2513-6.

[35] Lamprecht, A.-L. and Turner, K. J. Scientific workflows. *International Journal on Software Tools for Technology Transfer*. Nov 2016, vol. 18, no. 6, p. 575–580. DOI: 10.1007/s10009-016-0428-z.

[36] Long, J. Process Modeling Style. In: Long, J., ed. *Process Modeling Style*. Boston: Morgan Kaufmann, 2014, p. 14–16. DOI: 10.1016/B978-0-12-800959-8.00004-3. ISBN 978-0-12-800959-8.

[37] Ludäscher, B., Altintas, I., Berkley, C., Higgins, D., Jaeger, E. et al. Scientific workflow management and the Kepler system. *Concurrency and Computation: Practice and Experience*. 2006, vol. 18, no. 10, p. 1039–1065. DOI: 10.1002/cpe.994.

[38] Manohar, S. and Dantuma, M. Current and future trends in photoacoustic breast imaging. *Photoacoustics*. Elsevier. december 2019, vol. 16. DOI: 10.1016/j.pacs.2019.04.004. ISSN 2213-5979.

[39] Menaka, M. and Kumar, K. S. S. Workflow scheduling in cloud environment – Challenges, tools, limitations and methodologies: A review. *Measurement: Sensors*. Elsevier Ltd. december 2022, vol. 24. DOI: 10.1016/j.measen.2022.100436. ISSN 26659174.

[40] Meng, H. and Thain, D. Facilitating the reproducibility of scientific workflows with execution environment specifications. *Procedia Computer Science*. Elsevier. 2017, vol. 108, p. 705–714. DOI: 10.1016/j.procs.2017.05.116. International Conference on Computational Science, ICCS 2017, 12-14 June 2017, Zurich, Switzerland.

[41] Meng, H. and Thain, D. Facilitating the Reproducibility of Scientific Workflows with Execution Environment Specifications. *Procedia Computer Science*. 2017, vol. 108, p. 705–714. DOI: 10.1016/j.procs.2017.05.116. ISSN 1877-0509. International Conference on Computational Science, ICCS 2017, 12-14 June 2017, Zurich, Switzerland.

[42] Mens, T., Decan, A. and Spanoudakis, N. A method for testing and validating executable statechart models. *Software & Systems Modeling*. 2018. DOI: 10.1007/s10270-018-0676-3.

[43] Mohammadi, L., Behnam, H., Tavakkoli, J. and Avanaki, M. R. Skull's photoacoustic attenuation and dispersion modeling with deterministic ray-tracing: Towards real-time aberration correction. *Sensors (Switzerland)*. 2019. DOI: 10.3390/s19020345. ISSN 14248220.

[44] Obaida, M. A. and Liu, J. Simulation of HPC job scheduling and large-scale parallel workloads. In: IEEE. *2017 Winter Simulation Conference (WSC)*. 2017, p. 920–931. DOI: 10.1109/WSC.2017.8247843. ISSN 1558-4305.

[45] Omara, F. A. and Arafa, M. M. Genetic algorithms for task scheduling problem. *Journal of Parallel and Distributed Computing*. 2010, vol. 70, no. 1, p. 13–22. DOI: 10.1016/j.jpdc.2009.09.009. ISSN 0743-7315.

[46] Organization, W. H. *Training in Diagnostic Ultrasound: essentials, principles and standards: Report of a WHO Study Group*. World Health Organization, 1998. ISBN 9789241208758.

[47] Pascual, J. A., Navaridas, J. and Miguel Alonso, J. Job Scheduling Strategies for Parallel Processing. In: Frachtenberg, Eitan Schwiegelshohn, U., ed. *JSSPP 2009. Lecture Notes in Computer Science, vol. 5798*. Berlin, Heidelberg: Springer, 2009. DOI: 10.1007/978-3-642-04633-9_8. ISBN 978-3-642-04633-9.

[48] Poli, R., Langdon, W. and Mcphee, N. *A Field Guide to Genetic Programming*. January 2008. ISBN 978-1-4092-0073-4.

[49] Postema, M. *Fundamentals of Medical Ultrasonics*. CRC Press, 2007. ISBN 9780429176487.

[50] Poudel, J., Lou, Y. and Anastasio, M. A. A survey of computational frameworks for solving the acoustic inverse problem in three-dimensional photoacoustic computed tomography. *Physics in Medicine and Biology*. may 2019. DOI: 10.1088/1361-6560/ab2017. ISSN 13616560.

[51] Pouliopoulos, A., Wu, S.-Y., Burgess, M., Karakatsani, M., Kamimura, H. et al. A Clinical System for Non-invasive Blood-Brain Barrier Opening Using a Neuronavigation-Guided Single-Element Focused Ultrasound Transducer. *Ultrasound in Medicine and Biology*. october 2019, vol. 46, p. 73–89. DOI: 10.1016/j.ultrasmedbio.2019.09.010.

[52] Quinn, M. and Cleary, P. Intellectual capital and business performance: An exploratory study of the impact of cloud-based accounting and finance infrastructure. *Journal of Intellectual Capital*. march 2016, vol. 17. DOI: 10.1108/JIC-06-2015-0058.

[53] Robert, Y. Task Graph Scheduling. In: Padua, D., ed. *Encyclopedia of Parallel Computing*. Boston, MA: Springer US, 2011, p. 2013–2025. DOI: 10.1007/978-0-387-09766-4_42. ISBN 978-0-387-09766-4.

[54] Rossi, F., Beek, P. van and Walsh, T. *Handbook of Constraint Programming*. Elsevier, 2006. ISBN 9780080463803.

[55] Russell, S. J. and Norvig, P. *Artificial Intelligence: A Modern Approach*. 2nd ed. Upper Saddle River, New Jersey: Prentice Hall, 2003. 111–114 p. ISBN 0-13-790395-2.

[56] Sarkar, V. *Partitioning and Scheduling Parallel Programs for Multiprocessors*. MIT Press, Cambridge, 1989. 101–154 p. ISBN 0-262-69130-2.

[57] Sarkar, V. *Partitioning and Scheduling Parallel Programs for Multiprocessors*. Cambridge, MA, USA: MIT Press, 1989. 101–154 p. ISBN 0262691302.

[58] Schuchart, J., Hackenberg, D., Schone, R., Ilsche, T., Nagappan, R. et al. The Shift from Processor Power Consumption to Performance Variations: Fundamental Implications at Scale. *Computer Science - Research and Development*. Springer. 2016, vol. 31, no. 4, p. 197–205. DOI: 10.1007/s00450-016-0327-2.

[59] Shvachko, K., Kuang, H., Radia, S. and Chansler, R. The Hadoop Distributed File System. In: IEEE Computer Society. *Proceedings of the 2010 IEEE 26th Symposium on Mass Storage Systems and Technologies (MSST)*. Washington, DC, USA: [b.n.], 2010, p. 1–10. DOI: 10.1109/MSST.2010.5496972. ISBN 978-1-4244-7153-9.

[60] Sierksma, G. and Zwols, Y. *Linear and Integer Optimization: Theory and Practice*. 3rd ed. CRC Press, 2015. 1 p. ISBN 978-1498710169.

[61] Silva, R. F. da, Filgueira, R., Pietri, I., Jiang, M., Sakellariou, R. et al. A characterization of workflow management systems for extreme-scale applications. *Future Generation Computer Systems*. 2017. DOI: 10.1016/j.future.2017.02.026. ISSN 0167739X.

[62] Singh, P., Quadri, Z. and Kumar, A. Comparative Study of Parallel Scheduling Algorithm for Parallel Job. *International Journal of Computer Applications*. 2016, vol. 134, no. 10, p. 10–14. DOI: 10.5120/ijca2016908061.

[63] Srinivasan, K. and Sadayappan. A robust scheduling technology for moldable scheduling of parallel jobs. In: *Proceedings IEEE International Conference on Cluster Computing CLUSTR-03*. IEEE, 2003, p. 92–99. DOI: 10.1109/CLUSTR.2003.1253304. ISBN 0-7695-2066-9.

[64] Sung, H., Ferlay, J., Siegel, R. L., Laversanne, M., Soerjomataram, I. et al. Global Cancer Statistics 2020: GLOBOCAN Estimates of Incidence and Mortality Worldwide for 36 Cancers in 185 Countries. *CA: A Cancer Journal for Clinicians*. 2021, vol. 71, no. 3, p. 209–249. DOI: 10.3322/caac.21660.

[65] Suomi, V., Jaros, J., Treeby, B. and Cleveland, R. Nonlinear 3-D simulation of high-intensity focused ultrasound therapy in the Kidney. In:. IEEE, Aug 2016, p. 5648–5651. DOI: 10.1109/EMBC.2016.7592008. ISBN 978-1-4577-0220-4.

[66] Suomi, V., Treeby, B., Jaros, J., Makela, P., Anttinen, M. et al. Transurethral ultrasound therapy of the prostate in the presence of calcifications: A simulation study. *Medical physics*. september 2018, vol. 45, p. 4793–4805. DOI: 10.1002/mp.13183.

[67] Treeby, B. E. and Cox, B. T. K-Wave: MATLAB toolbox for the simulation and reconstruction of photoacoustic wave-fields. *Journal of Biomedical Optics*. Mar–Apr 2010, vol. 15, no. 2, p. 021314. DOI: 10.1117/1.3360308.

[68] Van Der Aalst, W. and Van Hee, K. M. *Workflow management - models, methods and systems*. January 2004. ISBN 978-0262720465.

[69] Virtanen, P., Gommers, R., Oliphant, T. E. et al. SciPy 1.0: Fundamental Algorithms for Scientific Computing in Python. *Nature Methods*. Nature Publishing Group. 2020, vol. 17, p. 261–272. DOI: 10.1038/s41592-019-0686-2.

[70] Wolstencroft, K., Haines, R., Fellows, D., Williams, A., Withers, D. et al. The Taverna workflow suite: designing and executing workflows of Web Services on the desktop, web or in the cloud. *Nucleic Acids Research*. may 2013, vol. 41, W1, p. W557–W561. DOI: 10.1093/nar/gkt328. ISSN 0305-1048.

[71] Ye, D., Chen, D. Z. and Zhang, G. Online scheduling of moldable parallel tasks. *Journal of Scheduling*. Dec 2018, vol. 21, no. 6, p. 647–654.

[72] Ye, X., Liang, J., Liu, S. and Li, J. A Survey on Scheduling Workflows in Cloud Environment. In:. January 2015, p. 344–348. DOI: 10.1109/ICNISC.2015.91.

[73] Zaharia, M., Chowdhury, M., Franklin, M. J., Shenker, S. and Stoica, I. *Spark: Cluster Computing with Working Sets*. UCB/EECS-2010-10. UC Berkeley, 2010.

[74] Zhou, Y.-F. High intensity focused ultrasound in clinical tumor ablation. *World Journal of Clinical Oncology*. 2011, vol. 2, no. 1, p. 8–27. DOI: 10.5306/wjco.v2.i1.8. ISSN 2218-4333.

Publications

[mjaros1] Jaros, M. Adaptive Execution Planning in Workflow Management Systems. In: *Počítačové architektury a diagnostika 2019*. Academic and Medical Conference Agency, 2019, p. 23–26. ISBN 978-80-88214-20-5. Available at: https://www.fit.vut.cz/research/publication/12018.

[mjaros2] Jaros, M. and Jaros, J. Performance-Cost Optimization of Moldable Scientific Workflows. In: *Job Scheduling Strategies for Parallel Processing*. Springer International Publishing, 2021, vol. 12985, p. 149–167. Lecture Notes in Computer Science (including subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics). DOI: 10.1007/978-3-030-88224-2_8. ISBN 978-3-030-88223-5. Available at: https://www.fit.vut.cz/research/publication/12298.

[mjaros3] Jaros, M. and Jaros, J. Optimization of Execution Parameters of Moldable Workflows under Incomplete Performance Data. In: *Job Scheduling Strategies for Parallel Processing. JSSPP 2022*. Springer Nature Switzerland AG, 2023, vol. 13592, p. 152–171. Lecture Notes in Computer Science. DOI: 10.1007/978-3-031-22698-4_8. ISBN 978-3-031-22697-7. Available at: https://www.fit.vut.cz/research/publication/12691.

[mjaros4] Jaros, M., Klusacek, D. and Jaros, J. Optimizing Biomedical Ultrasound Workflow Scheduling Using Cluster Simulations. In: *Job Scheduling Strategies for Parallel Processing. JSSSP 2020*. Springer Nature Switzerland AG, 2020, vol. 12326, p. 68–84. Lecture Notes in Computer Science (including subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics). DOI: 10.1007/978-3-030-63171-0_4. ISBN 978-3-030-63170-3. Available at: https://www.fit.vut.cz/research/publication/12299.

[mjaros5] Jaros, M., Sasak, T., Treeby, E. B. and Jaros, J. Estimation of Execution Parameters for k-Wave Simulations. In: *High Performance Computing in Science and Engineering. HPCSE 2019*. Springer Nature Switzerland AG, 2021, p. 116–134. Lecture Notes in Computer Science (including subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics). DOI: 10.1007/978-3-030-67077-1_7. ISBN 978-3-030-67076-4. Available at: https://www.fit.vut.cz/research/publication/12137.

[mjaros6] Jaros, M., Treeby, E. B., Georgiou, P. and Jaros, J. K-Dispatch: A Workflow Management System for the Automated Execution of Biomedical Ultrasound Simulations on Remote Computing Resources. In: *Proceedings of the Platform for Advanced Scientific Computing Conference, PASC 2020*. Association for Computing Machinery, 2020, p. 1–10. DOI: 10.1145/3394277.3401854.

ISBN 978-1-4503-7993-9. Available at: https://www.fit.vut.cz/research/publication/12125.

[mjaros7] Jaroš, M. Scientific Workflows Management. In: *Počítačové architektúry & diagnostika PAD 2018*. University of West Bohemia in Pilsen, 2018, p. 25–28. ISBN 978-80-261-0814-6. Available at: https://www.fit.vut.cz/research/publication/11768.

[mjaros8] Kuklis, F., Jaros, M. and Jaros, J. Accelerated Design of HIFU Treatment Plans Using Island-Based Evolutionary Strategy. In: *Applications of Evolutionary Computation*. Springer International Publishing, 2020, vol. 12104, p. 463–478. Lecture Notes in Computer Science (including subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics). DOI: 10.1007/978-3-030-43722-0_30. ISBN 978-3-030-43721-3. Available at: https://www.fit.vut.cz/research/publication/12043.

[mjaros9] Čudová, M. Framework for Planning, Running and Monitoring Cooperating Computations. In: *Počítačové architektúry & diagnostika PAD 2017*. Slovak University of Technology in Bratislava, 2017, p. 20–23. ISBN 978-80-972784-0-3. Available at: https://www.fit.vut.cz/research/publication/11475.

[mjaros10] Čudová, M., Treeby, E. B. and Jaroš, J. Design of HIFU treatment plans using an evolutionary strategy. In: *GECCO 2018 Companion - Proceedings of the 2018 Genetic and Evolutionary Computation Conference Companion*. Association for Computing Machinery, 2018, p. 1568–1575. DOI: 10.1145/3205651.3208268. ISBN 978-1-4503-5764-7. Available at: https://www.fit.vut.cz/research/publication/11696.

Appendices

Appendix A

Evaluation and Tuning of Genetic Algorithms Optimizer

In order to evaluate the capabilities of the genetic algorithm and tune genetic operators and control parameters, a performance database containing tasks with various parallel scaling characteristics was created. This database was consequently used to build several test workflows covering a wide range of typical use cases.

When creating the performance database, the number of nodes assigned to each task was limited to the interval <1, 36>. Figure A.1 shows generated strong scaling curves consisting of 6 different examples, three of which are purely artificial and three of which are realistic.

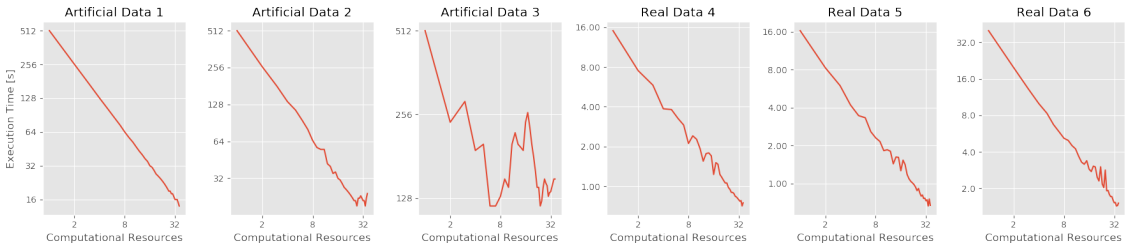


Figure A.1: Investigated strong scaling curves generated for both artificial data and a realistic application. Execution time is related to a single simulation step.

The artificial scalings were created to capture specific corner cases. The first plot represents perfect scaling, which is typical for embarrassingly parallel applications. The second one shows good scaling with some minor artifacts at the end typical for memory-bound applications. The last example represents an extreme type of bad scaling containing many local extremes caused by unbalanced workload distribution or high communication overhead. The realistic scalings were measured on the distributed MPI version of the acoustic k-Wave toolbox [32], which is our target application. In this case, k-Wave simulated the ultrasound wave propagation in tissue-realistic media with grid resolutions of 500^3, 512^3 and 544^3 grid points. The pseudo-spectral solver heavily depends on 3D fast Fourier transforms which are highly sensitive to the highest prime factor of the domain sizes, which are 5, 2, and 17 for the domain sizes of 500^3, 512^3 and 544^3 grid points, respectively. The toolbox was compiled using the GCC compiler 8.2, OpenMPI 4.0 and FFTW 3.3.8, and measured on the Barbora supercomputer at IT4Innovations.

The size of the experimental workflows varied between 8 and 64 tasks. Bigger workflows were not taken into account since many HPC facilities limit the number of currently submitted tasks to low tens, e.g. 100, at IT4Innovations where the experimenters were conducted. In order to bring some variability into the workflows, the execution time of each task was randomly altered by up to 25%, which typically happens in integrative algorithms with an error-based termination condition. Moreover, since the performance variability between nodes in the cluster may attribute to about 5 % deviations in the execution time [58], we added this variation to the execution time as well. For the sake of brevity, this section will only present experimental results obtained on Artificial Datasets 2, 3 and Real Dataset 4, and the workflow structure depicted in Fig. A.2.

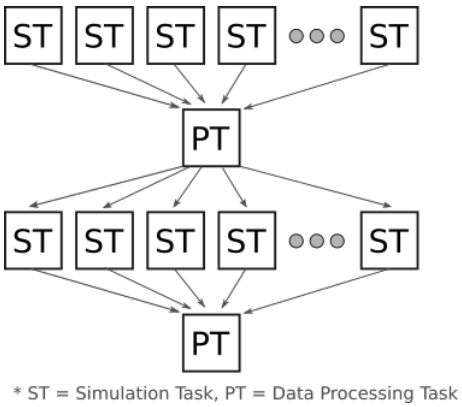


Figure A.2: The structure of the investigated workflow where computationally heavy Simulation Tasks (STs) produce data and less demanding Processing Tasks (PTs) imply barriers between ST stages.

The proposed experiments investigated (1) control parameters and operators used in the GA and summarized in Table A.1, (2) local task optimization fitness function and (3) global workflow optimization fitness function utilizing either shared allocation (GOSA) or on-demand allocation (GODA) resources. The experiments employ fitness functions defined in Eq. (5.3), Eq. (5.7) and Eq. (5.9), and they were evaluated using an artificial cluster composed of 64 computational nodes. To get statistically relevant results, 20 independent runs for each configuration were carried out.

Table A.1: Control Parameters and Operators of the Genetic Algorithm

Control parameter	Details
Selection method	Steady-state (sss), roulette wheel (rws), rank, tournament
Crossover method	Single-point, uniform, arithmetic
Crossover probability	0.7, 0.9
Mutation method	Random
Gene mutation probability	0.1%, 0.5%, 1%, 5%, and 10%
Elitism	5% of the best individuals copied to the next population.
Population size	25, 50, 100, 150
Maximum number of generations	1000

Figures A.3 and A.4 show experiments performing local tasks optimization on Artificial Datasets no. 2 and 3. While all runs of the GA with almost any combination of control parameters found the solution for the perfect scaling, the Artificial Dataset no. 3 usually led to a sub-optimal solution. We managed to find the optimal solution only for small workflows of 8 tasks using steady-state selection, the uniform crossover of 0.7 probability with 28 % average success rate and 20 % median success rate. It may be calming that such scaling is really extreme and it is rather unlikely for real runs.

A comparison of various control parameters can be seen in Fig. A.3. The plots show the generation in which the GA converged (the dashed line stands for the optimal solution) together with the execution time of the best solution found. The colour of the dots denotes the particular parameter combination. It is visible that the mutation rate of 5 % seems the most promising. Thus, Fig. A.4 investigates other control parameters for the experiment with Artificial Dataset no. 2 while using a 5 % mutation rate fixed.

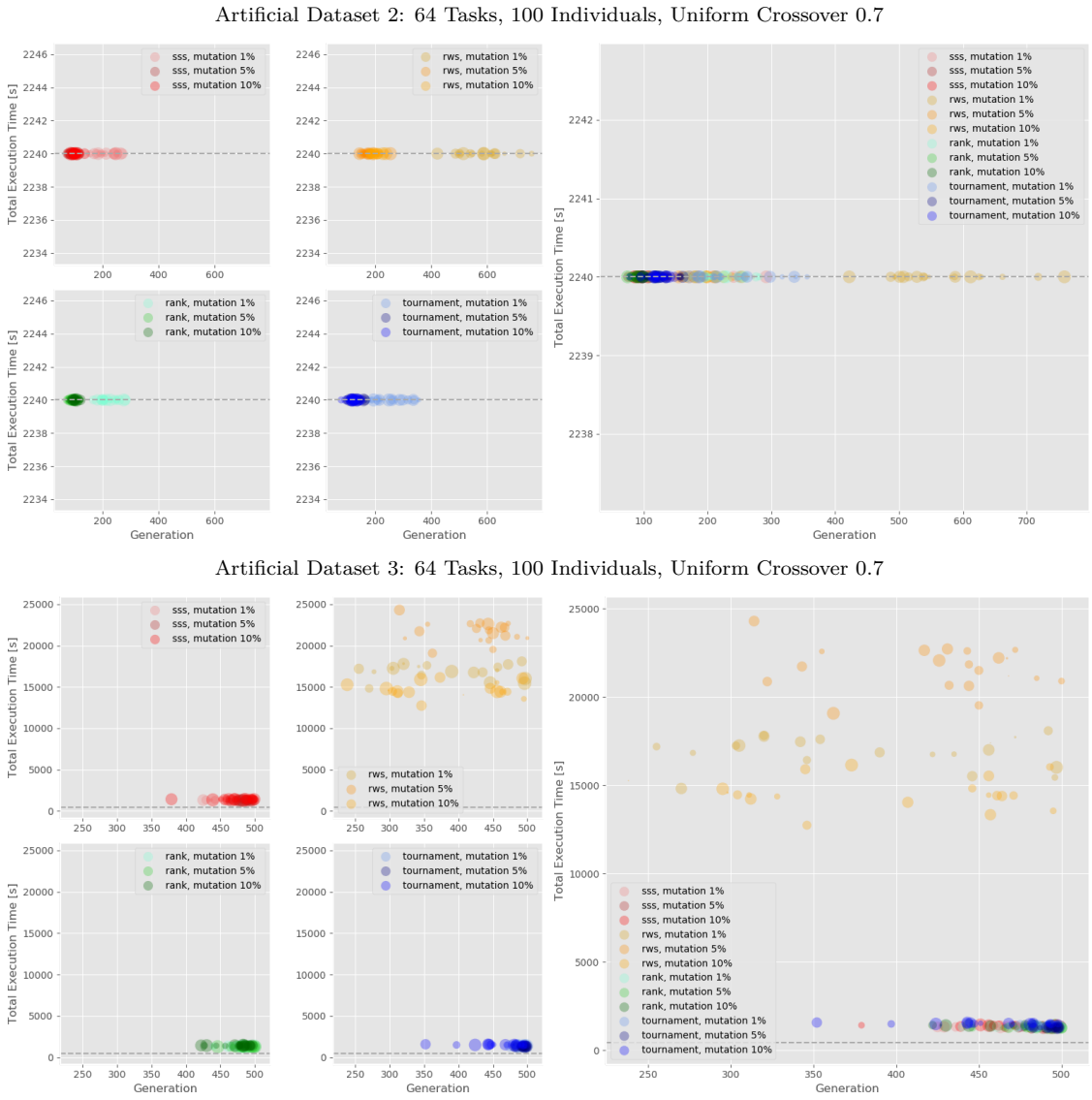


Figure A.3: Investigation of the GA control parameters and operators shown on Artificial Dataset no. 2 (top) and Artificial Dataset no. 3 (bottom). The plots illustrate the number of generations needed to find the optimal solution (dashed line). The y-axis shows the makespan.

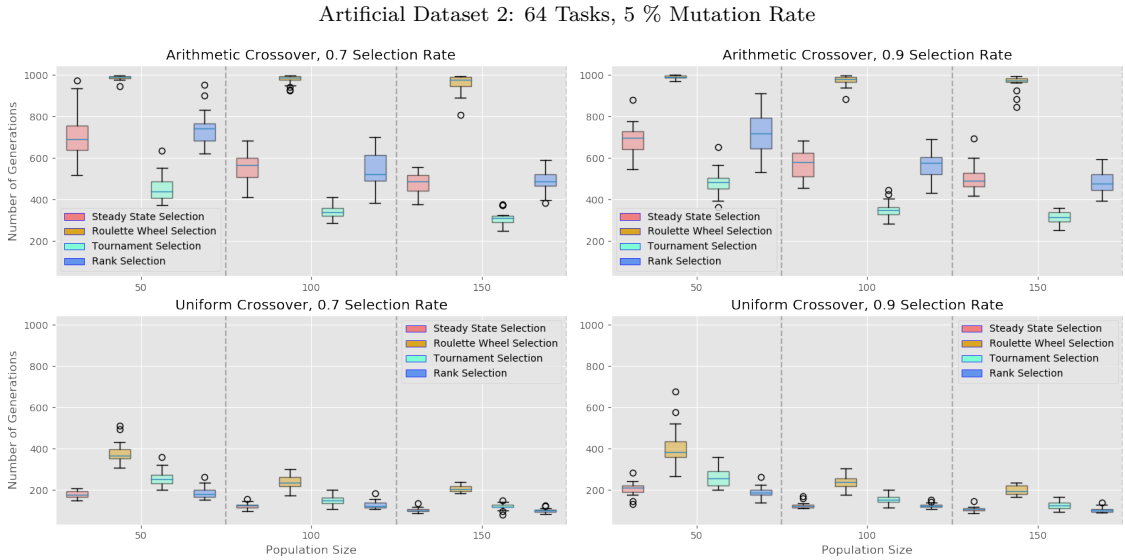


Figure A.4: Evaluation of the necessary number of generations to reach convergence for various control parameters.

Local task optimization shows very good capabilities in optimizing particular tasks. From 20 independent runs of the GA, more than 90% trials always found the best possible solution. The following Table A.2 summarises the average evolution runtime of the GA in seconds and the median number of generations to achieve the best solution for various sizes of the population while considering steady-state selection, the uniform crossover of 0.7 probability, random gene mutation of 1 % probability, and 5 % elitism. The evolution runtimes were obtained for Real Dataset no. 4. The table reveals that the necessary population size linearly grows with the size of the workflow from 25 up to 150 individuals, but still stays quite small. This is natural behaviour since bigger workflows require longer chromosomes which in turn requires larger populations to keep promising building blocks of the solution. On the other hand, the execution time appears to grow quadratically. This growth can be attributed to a product of increasing population size which rises the number of fitness function evaluations, and the linearly growing time complexity of the fitness function evaluation. Nevertheless, an execution time of nearly 14 s on a single computing core with 90 % success rate for the biggest workflow is an excellent result.

Table A.2: Evolution time for Real Dataset no. 4 to find the best solution for different workflows size. Various sizes of populations were investigated. Values in brackets stand for the generation number (median) when the solution was found. Solutions highlighted in bold determine the minimum number of generations needed to reach a 90 % success rate.

Population Size Number of Tasks	**25**	**50**	**100**	**150**
8	**0.45 s (225)**	**0.30 s (102)**	**0.28 s (48)**	
16	1.29 s (431)	**1.00 s (431)**	**1.66 s (151)**	
32		3.33 s (370)	**3.51 s (196)**	**4.75 s (176)**
64		10.8 s (598)	12.2 s (360)	**13.2 s (260)**

We may conclude that the uniform crossover produced the best results. The influence of the crossover probability was not statistically significant. The best selection strategy that drives the GA through the search spaces is hard to definitely determine. The difference amongst steady-state, rank and tournament selection methods is marginal but significantly higher when compared to Roulette Wheel selection. However, experiments with Real Dataset no. 4 showed steady-state selection as the best one followed by rank selection. The optimal mutation ratio seems to be around 0.5 %. As expected, the better the scaling the faster the convergence and the higher the success rate of finding the optimal resource assignment.

When performing a global workflow optimization, the GA does not converge to a single optimal solution but to many incomparable solutions. These solutions may be illustrated as a Pareto frontier. Figure A.5 shows the Pareto frontier for a workflow consisting of 64 tasks using Real Dataset no. 4 and performing the global optimization using on-demand allocation, together with dominated solutions illustrated using a line. Figure depicts evolved solutions while various trade-off coefficients α are employed. It can be seen, the solutions form clusters proving that using the trade-off coefficient, different solutions meeting different optimization criteria can be found. These clusters may slightly overlap. It can also be seen that although the fitness function may still push down the execution time, the computational cost remains the same from some point. Figure A.6 depicts suitable GA parameters for GODA when a trade-off coefficient of 0.95 is used. Steady-state selection and random gene mutation of 1.0 % rate defeats other methods. Both Rank and Tournament selection methods provide mostly identical results. We also observed that for smaller workflows counting 32 tasks included, Rank selection gives better results than steady-state selection, see Paper III.

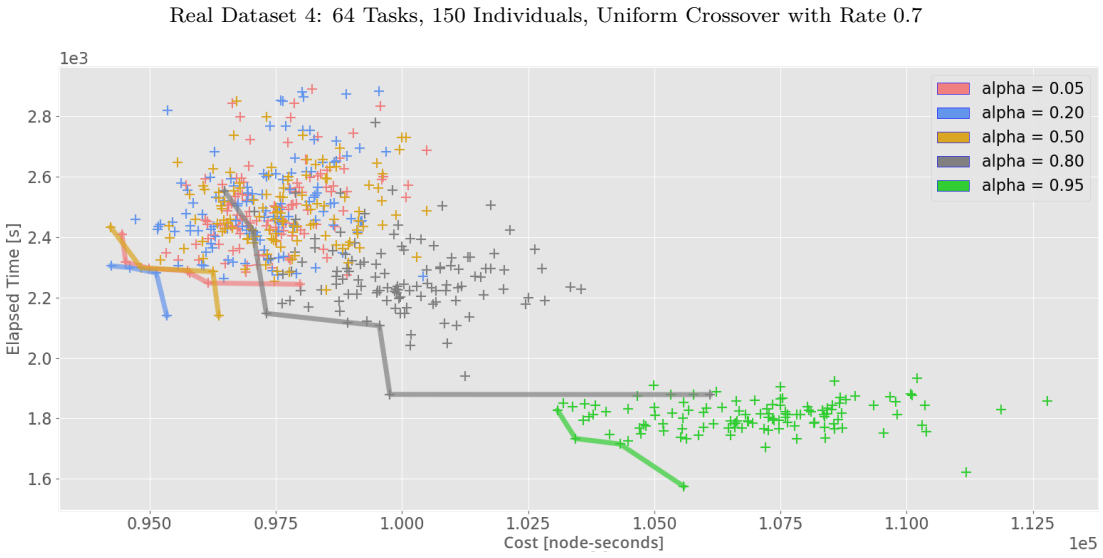


Figure A.5: Pareto frontier and dominated solutions obtained by multiple GA runs using the GODA fitness functions for the workflows without dependencies and various values of the α parameter. The computational cost on the x-axis is computed as the execution time multiplied by the number of computational resources, while the elapsed time on the y-axis stands for the makespan.

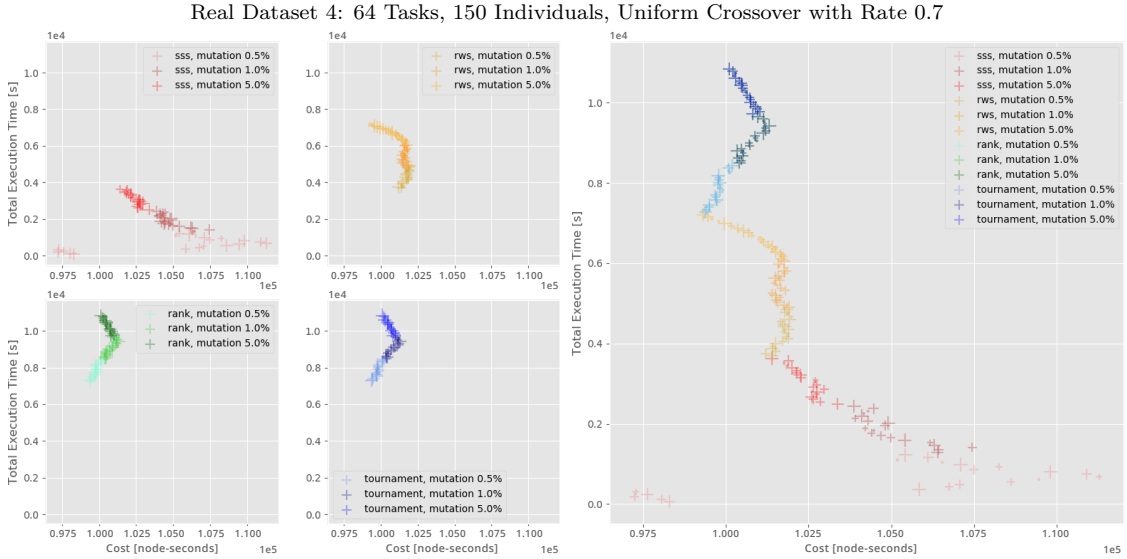


Figure A.6: Exploration of suitable GA parameters for GODA employing the trade-off coefficient of 0.95. It can be seen that the steady-state selection method gives the most promising results.

In the case of GOSA shown in Fig. A.7, solutions for different trade-off coefficients highly overlap. A smaller α parameter pushes the GA to find solutions with a smaller amount of unused resources (up to 29 %) but the range of found solutions is very high. When considering a simple case of a single workflow with task dependencies, the feasibility of the request to minimize idling resources may also be limited by the workflow structure. Figure A.8 shows that Roulette Wheel selection with 1.0 % random gene mutation rate is the most appropriate method for this kind of problem. Next, steady-state selection with 0.5 % random gene mutation seems to be able to give good results but without additional tuning and experiments cannot compete with roulette wheel selection. In this configuration, red points form a line which indicates that two optimization criteria are correlated and it is not feasible to prioritize between them. On the other hand, we found out that steady-state selection works better for smaller workflows counting less than 32 tasks included. Other selection methods are considered unsatisfying. The selection example uses a population of 100 individuals and a uniform crossover of a 0.7 rate. The runtimes of GODA and GOSA are shown in Table A.3 and Table A.4, respectively. The workflow containing 64 tasks was evolved on a single processor core within 2 and 4.5 minutes, respectively.

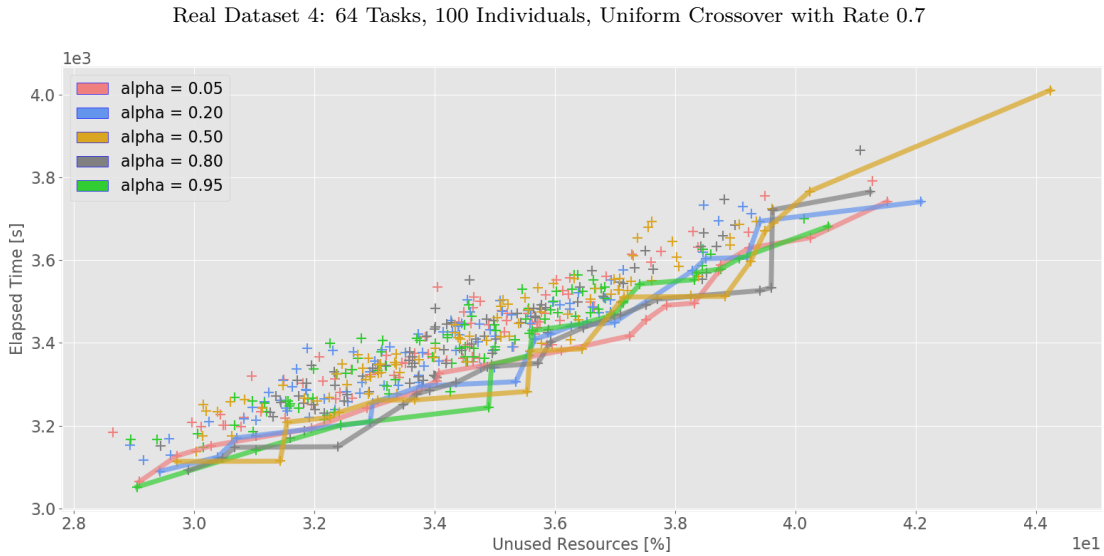


Figure A.7: Pareto frontier and dominated solutions obtained by multiple GA runs using the GOSA fitness function for the workflows with dependencies and various values of the α parameter. The computational cost on the x-axis is computed as a sum of tasks' execution times multiplied by the number of computational resources used, while elapsed time on the y-axis stands for the makespan.

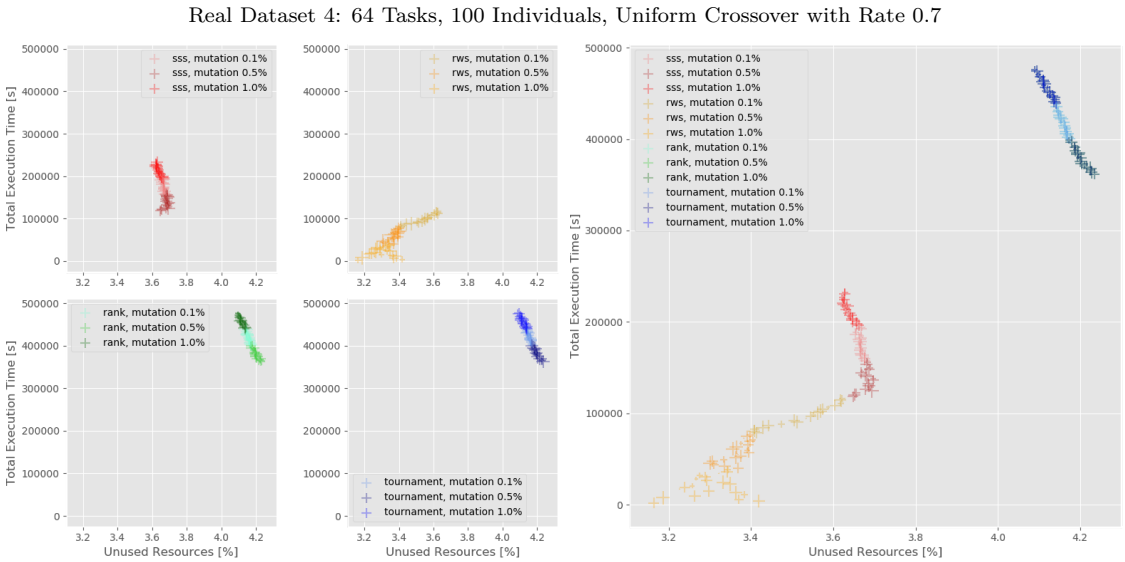


Figure A.8: Graph showing the search for suitable GA parameters for GOSA fitness function employing the trade-off coefficient of 0.05%. It is evident that only the Roulette Wheel selection gives reasonable results. The steady-state selection using 0.5 % random mutation provided acceptable results too but did not achieve as good values as the roulette wheel. Both rank and tournament selection methods provided the same unsatisfying results.

Table A.3: Evolution time for the GODA fitness function measured for Real Dataset no. 4 to find the best solution for different workflow sizes. Various sizes of populations were investigated. Values in brackets stand for the generation number (median) when the solution was found.

Population Size / Number of Tasks	**25**	**50**	**100**	**150**
8	0.68 s (135)	0.56 s (62)	0.51 s (28)	
16	1.60 s (160)	1.79 s (85)	2.58 s (60)	
32		6.36 s (148)	8.80 s (100)	11.88 s (90)
64		30.60 s (460)	62.7 s (285)	120.6 s (360)

Table A.4: Evolution time for the GOSA fitness function measured for Real Dataset no. 4 to find the best solution for different workflow sizes. Various sizes of populations were investigated. Values in brackets stand for the generation number (median) when the solution was found.

Population Size / Number of Tasks	**25**	**50**	**100**	**150**
8	1.03 s (205)	1.53 s (170)	4.41 s (245)	
16	3.40 s (340)	4.20 s (200)	13.33 s (310)	
32		24.1 s (560)	44.9 s (510)	73.26 s (555)
64		102.6 s (855)	189.2 s (860)	274.7 s (820)

Appendix B

Database Structure

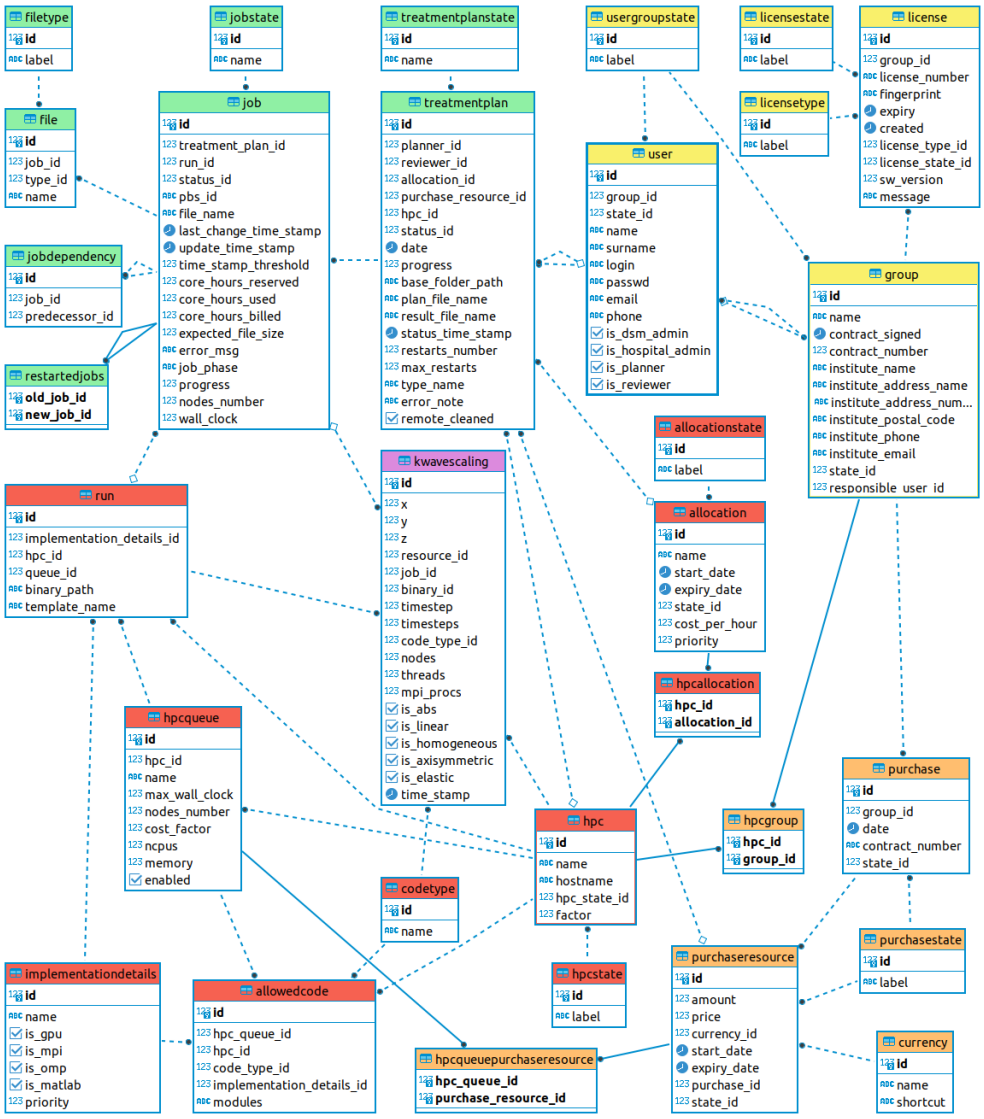


Figure B.1: Simplified Entity Relationship Diagram (ERD) of the k-Dispatch's database.

Explanatory notes

Table B.1: Brief description of database tables excluding tables that implement N-N relationship. Tables that define the states of the particular table are depicted on the right.

Table name	Meaning	Status index
treatmentplan	A workflow.	*treatmentplanstate*
job	A task in the workflow.	*jobstate*
jobdependency	Dependency relationship amongst tasks.	–
restartedjobs	A track of jobs that replaced error one.	–
file	A file associated with the job.	–
filetype	Type of the file, e.g., job script, log.	–
user	A user.	*usergroupstate*
group	A group of users.	*usergroupstate*
license	A license defining the software usage rights.	*licensestate*
licensetype	Type of the license.	-
hpc	A remote computing facility.	*hpcstate*
hpcqueue	A computing queue in the HPC.	–
allocation	An internal resource allocation.	*allocationstate*
codetype	A code type. For each code type, multiple implementations may exist.	–
implementationdetails	A binary (particular code type implementation) definition.	–
run	A submission script template for a particular binary.	–
allowedcode	Allowed code and the binary on the particular HPC and its queue.	–
purchase	A purchase of resource allocation made by a group.	*purchasestate*
purchaseresource	A purchase details on a particular HPC queue.	*purchasestate*
currency	A currency in which the purchase was made.	–
kwavescaling	Performance database for selected k-Wave codes.	-

Appendix C

Estimation of Task Execution Time using Symbolic Regression and Genetic Programming

The content of this chapter was created in collaboration with a master's student at Faculty of Information Technology, Brno University of Technology, Martin Buchta, who helped me with experiments and their evaluation.

The goal of this research is to better exploit the measured performance database and estimate more accurately the task execution time for a given amount of resources, input data, and computational facility. For this purpose, we decided to use symbolic regression and genetic programming.

Symbolic Regression is a type of machine learning technique that searches for a mathematical expression that best fits a given dataset. In our case, we are looking for a formula that describes the parallel scaling for a wide range of input data based on their characteristic features, such as domain size. Construction of this expression is, however, a complex problem requiring an extensive training dataset, a search space technique and a quality measure.

Genetic Programming (GP) [48] is a subfield of evolutionary computing that involves the use of evolutionary algorithms to automatically generate computer programs or models to describe a specific problem. The outputs generated by GP are mathematical expressions traditionally represented as tree structures since they can be easily evaluated in a recursive manner. Non-tree representations, however, have also been successfully used. For example, *Linear Genetic Programming (LGP)* uses a sequence of instructions from imperative programming languages and *Cartesian Genetic Programming (CGP)* uses a graph representation. Different forms of GP and their inner individual representation suit better different problems and applications, e.g., digital circuits, image recognition, data analysis, time-series predictions, etc. The quality measure used as a fitness function is typically based on some error metrics such as the mean square error.

We used two toolboxes, the Python Pandas[1] module and HeuristicLab[2] to create the scaling models. Since HeuristicLab achieved much better results, the most of experimenters were conducted in this toolbox. We created a training set containing 822 performance records of the MPI k-Wave solver. The training set covers 21 different domain sizes and

[1] https://pandas.pydata.org/

[2] https://dev.heuristiclab.com/trac.fcgi/wiki

runs executed over up to 100 nodes utilizing up to 32 computational cores (maximum) and up to 3072 MPI ranks. The test set was created from two distinct domain sizes counting 64 performance records.

As input variables/features considered by the symbolic regression, we selected the following parameters:

- domain size denoted as *ds*,
- maximum domain factor denoted as *mdf*,
- number of compute nodes denoted as *nodes*,
- number of MPI ranks denoted as *ranks*,
- MPI ranks ratio, denoted as *rr*, defined as a ratio of MPI ranks used to the number of cores available in a single node,
- number of used computational cores, denoted as *cores*, per a single node,
- MPI ranks used ratio, denoted as *rur*, saying how well is the computation distributed over available MPI ranks.

Since k-Wave is memory- and communication-bound, it is beneficial to execute fewer ranks per compute node than there are cores. Although some cores will become idle, the memory congestion will decrease. Spreading the MPI ranks over more nodes also brings higher aggregated network throughput.

Following mathematical functions were used (based on the default settings of HeuristicLab): *addition, subtraction, multiplication, division, average, exponential, logarithm, absolute value, constant (a random value), variable (input value multiplied by the constant), power of two, power of three, power of n, square root, cube root, nth root, if-then-else, greater than, less than, and, or, not, and xor.*
The optimization process was executed with the following settings and constraints:

- maximum tree depth: 16,
- maximum number of nodes: 42,
- train samples: 92% (822 out of 886 records),
- elites: 1,
- maximum number of generations: 1,000,000,
- mutation probability: 10%,
- population size: 100,
- fitness function: mean square error.

Using a desktop computer virtualised using Parallels Desktop,[3] a suitable model was found in generation number 845,242. The training process took 24 hours. The model reached the maximum error of 7.3 % which is very promising. The formula that models the strong scaling is depicted in Fig. C.1.

[3] https://www.parallels.com/eu/

Let us note, that the model training can be done offline and does not slow down the task submission. The model inference is almost instant. It is expected that retraining will only appear after substantial changes in the performance database (significantly different input data, new executables with new optimizations/features, computing facility upgrades, etc.).

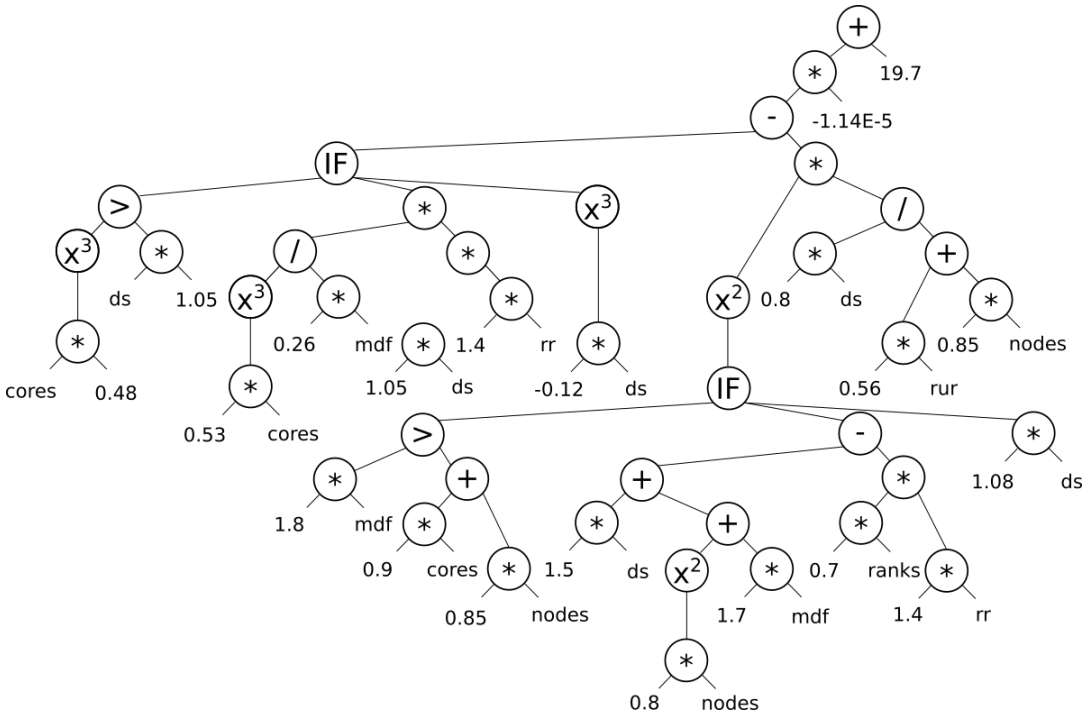


Figure C.1: Mathematical formula calculating the estimated k-Wave execution time based on a given set of features. Nodes contain tradition mathematical operations: *addition (+), subtraction (−), multiplication (∗), division (/), square root (x^2), cube root (x^3); conditions (IF)* and *compares (>)*. Leaves represent input variables and randomly generated numbers.

This experiment serves as a proof of concept, still under active investigation and thus, has not been published yet. Since the results obtained are very promising, in the near future we will (1) move the training to the cluster environment to reduce the training time, (2) add new mathematical operations such as modulo, rounding up and down to extend the default set of operators, (3) create a validation set, (4) extend the test set, and (5) perform other experiments, and discuss results and issues.

Related Papers

Related Paper I

Optimizing Biomedical Ultrasound Workflow Scheduling Using Cluster Simulations

JAROS Marta, KLUSACEK Dalibor and JAROS Jiri

Job Scheduling Strategies for Parallel Processing (JSSPP) 2020
DOI: 10.1007/978-3-030-63171-0_4

Optimizing Biomedical Ultrasound Workflow Scheduling Using Cluster Simulations

Marta Jaros[1( )], Dalibor Klusáček[2], and Jiri Jaros[1]

[1] Faculty of Information Technology, Centre of Excellence IT4Innovations, Brno University of Technology, Brno, Czech Republic
{martajaros,jarosjir}@fit.vutbr.cz
[2] CESNET a.l.e., Brno, Czech Republic
klusacek@cesnet.cz

Abstract. Therapeutic ultrasound plays an increasing role in dealing with oncological diseases, drug delivery and neurostimulation. To maximize the treatment outcome, thorough pre-operative planning using complex numerical models considering patient anatomy is crucial. From the computational point of view, the treatment planning can be seen as the execution of a complex workflow consisting of many different tasks with various computational requirements on a remote cluster or in cloud. Since these resources are precious, workflow scheduling plays an important part in the whole process.

This paper describes an extended version of the k-Dispatch workflow management system that uses historical performance data collected on similar workflows to choose suitable amount of computational resources and estimates execution time and cost of particular tasks. This paper also introduces necessary extensions to the Alea cluster simulator that enable the estimation of the queuing and total execution time of the whole workflow. The conjunction of both systems then allows for fine-grain optimization of the workflow execution parameters with respect to the current cluster utilization. The experimental results show that this approach is able to reduce the computational time by 26%.

Keywords: Scheduling · Workflow · k-Dispatch · Simulation · Alea

1 Introduction

The use of ultrasound as a diagnostic imaging tool is well-known, particularly during pregnancy where ultrasound is used to create pictures of developing babies. In recent years, a growing number of therapeutic applications of ultrasound have also been demonstrated [17]. The goal of therapeutic ultrasound is to modify the function or structure of biological tissue in some way rather than produce an anatomical image. This is possible because the mechanical vibrations caused by ultrasound waves can affect tissue in different ways, for example, by

D. Klusáček et al. (Eds.): JSSPP 2020, LNCS 12326, pp. 68–84, 2020.
https://doi.org/10.1007/978-3-030-63171-0_4

causing the tissue to heat up or by generating internal forces that can agitate the cells or tissue scaffolding. These ultrasound bioeffects offer enormous potential to develop new ways to treat major diseases. In the last few years, clinical trials of different ultrasound therapies have demonstrated the ability of ultrasound to destroy cells through rapid heating for the treatment of cancer and neurological disorders, target the delivery of anticancer drugs, stimulate or modulate the excitability of neurons, and temporarily open the blood-brain barrier to allow drugs to be delivered more effectively [12]. These treatments are all completely noninvasive and have the potential to significantly improve patient outcomes.

The fundamental challenge shared by all applications of therapeutic ultrasound is that the ultrasound energy must be delivered accurately, safely, and noninvasively to the target region within the body identified by the doctor. This is difficult because bones and other tissue interfaces can severely distort the shape of the ultrasound beam. In principle, it is possible to predict and correct for these distortions using models of how ultrasound waves travel through the body. However, the underlying physics is complex and typically must consider nonlinear wave propagation through absorbing media with spatially varying material properties. Simple formulas do not exist for this scenario, so models used for studying therapeutic ultrasound are instead based on the numerical solution of the wave equation (or the corresponding constitutive equations) [19].

The k-Wave toolbox designed for the time-domain simulation of acoustic waves in biomedical materials has become very popular in the international ultrasonic community [18]. Nevertheless, modelling ultrasound treatments using this toolbox requires very complex and intensive computations that generally cannot be satisfied by desktop computers or small servers [6]. It is thus essential to offload the computational work to cloud or HPC clusters. Unfortunately, using these facilities and composing the processing workflow representing the treatment is complex even for experienced developers. Therefore, it is crucial to offer clinical end-users a middleware layer that features a simple interface (e.g., web page, medical GUI, etc.) to upload treatment setups with related data and automate the execution. This middleware layer is implemented by our software package called k-Dispatch [9].

k-Dispatch, however, offers much more than simple job submission with semi-automated execution and monitoring such as HTCondor [8] or Pegasus [3]. k-Dispatch additionally provides a low level automatization by selecting suitable execution parameters specifying the amount of compute resources and estimates required execution time for particular tasks. This is enabled by a fixed set of medically certified binaries serving as building blocks for user's workflows, and collected performance data updated after every successful run. Based on the task input data, k-Dispatch searches the performance database to estimate scaling of particular binaries on the fly, and tune the execution parameters to minimize execution time and/or computational cost. Nevertheless, since the computational resources are shared by multiple users and workflows, the queuing times and user interference may depreciate the execution plan.

Therefore, this paper deals with the extension of the Alea cluster simulator [10] to estimate the workflow makespan, i.e., the overall execution time including the queuing times as well as the computational cost for complex biomedical ultrasound workflows. For every workflow, k-Dispatch prepares a candidate set of execution parameters and passes them to Alea which simulates the workflow execution with respect to the cluster parameters, job scheduling system setup, and background workload.

This paper is organized as follows. In Sect. 2, the considered workflow scheduling problem is discussed thoroughly. Section 3 describes the Alea simulator and its new workflow-related functionality. Next, Sect. 4 demonstrates the newly developed simulation capabilities which are crucial for the k-Dispatch's scheduling module when analyzing the quality of considered workflow execution plan(s). The paper is concluded and the future work is discussed in Sect. 5.

2 Problem Description

The k-Dispatch's mission is to enable fully automated offloading of biomedical ultrasound workflows built on the top of the acoustic k-Wave toolbox to the HPC and cloud environment. These workflows are used for pre-operative treatment planning based on the patient specific images to maximize the treatment outcome. Every treatment plan consists of many tasks carrying out data processing, ultrasound sonications, and thermal and tissue model evaluations. Their orchestration is encoded in the form of a directed acyclic graph (DAG) describing the data dependency and precedence relations [14]. Every task is evaluated by an appropriate piece of software included in the k-Wave toolbox. The most time consuming ultrasound tasks can be executed by a variety of simulation codes optimized for particular hardware platform including shared memory systems, single Nvidia GPU, and distributed memory CPU and GPU clusters. Each binary is suitable for a different simulation size and complexity and has associated a different simulation cost. The shared memory/GPU versions can be used for treatment planning in small volumes such as prostate, while the distributed versions are suitable for large treatments in the brain, liver or kidney.

Working within the medical environment implies all software must undergo a strict regulatory and certification process. It is thus not possible for users to use their own binaries. Instead, only authorized personnel are allowed to deploy the simulation binaries within a strictly controlled environment, e.g., inside Singularity [7] or Docker [13] containers. The clinical users are, of course, allowed to compose different workflows from predefined modules, change the number of sonications, their parameters or upload different patient images.

These restrictions, on the other hand, open great opportunity for automated performance tuning and resource allocation. Since the binaries are fixed, their execution can be monitored, and the performance data collected and analysed for future use. k-Dispatch maintains a complex performance database including information about every successful task containing binary name, cluster name, queue name, amount of resources, simulation medium size and properties, wall

clock time and computational cost. Once a new ultrasound workflow is received, k-Dispatch decodes individual tasks and assigns them suitable binaries, appropriate resources, and estimates the wall clock time. Then, the tasks are handed over to the cluster job scheduler that is responsible for their execution.

The optimizations of execution parameters help minimizing the computational cost and/or the execution time of individual tasks. However, since every workflow contains many tasks and there are usually multiple workflows being simultaneously executed, the isolated optimization of individual tasks may lead to poor cluster utilization or long queuing times. It is necessary to focus on bigger picture and take into account the dependencies between tasks of (multiple) DAGs. However, the optimization complexity can become exponential [15]. Therefore, there is a need for heuristics that include fast cluster simulations to evaluate the overall execution time of all workflows currently in the system. This information provides the feedback to the planning logic to adjust the amount of resources for particular tasks.

2.1 Workflows and Infrastructure

There are many workflow templates supported by k-Dispatch [9]. Figure 1 shows an intracranial neuromodulation workflow used for treatment planning of essential tremor and Parkinson's disease procedures. The purpose of this workflow is to verify the ultrasound hits the desired target but does not rise the tissue temperature above safety levels.

The workflow starts with the aberration correction pre-processor converting the treatment parameters and patient data into input files for the following ultrasound simulations. This task is usually simple and only employs a single compute node for a couple of minutes per sonication. The total execution time thus increases with the number of sonications (N) being executed (see the first line in Table 1). Next, a number of independent aberration correction simulations is executed. For this particular example, an ultrasound transducer with a driving frequency of 550 kHz, and a medium of 25 cm $\times$ 29 cm $\times$ 19 cm is used.

Table 1. Execution time and amount of resources for particular tasks within the neurostimulation workflow measured on the Anselm Supercomputer. The number of sonications (denoted by N) influences the total execution time.

Code type	Number of nodes	Execution time
Aberration correction pre-processor	1	$400 + 250 \cdot N$ [s]
Aberration correction simulation	1–16	$< 34.31, 4.96 >$ [h]
Aberration correction post-processor	1	$115 + 95 \cdot N$ [s]
Forward planning pre-processor	1	$650 + 310 \cdot N$ [s]
Forward planning simulation	1–16	$< 30.90, 4.72 >$ [h]
Forward planning post-processor	1	$105 + 60 \cdot N$ [s]
Thermal simulation	1	$30 + 720 \cdot N$ [s]

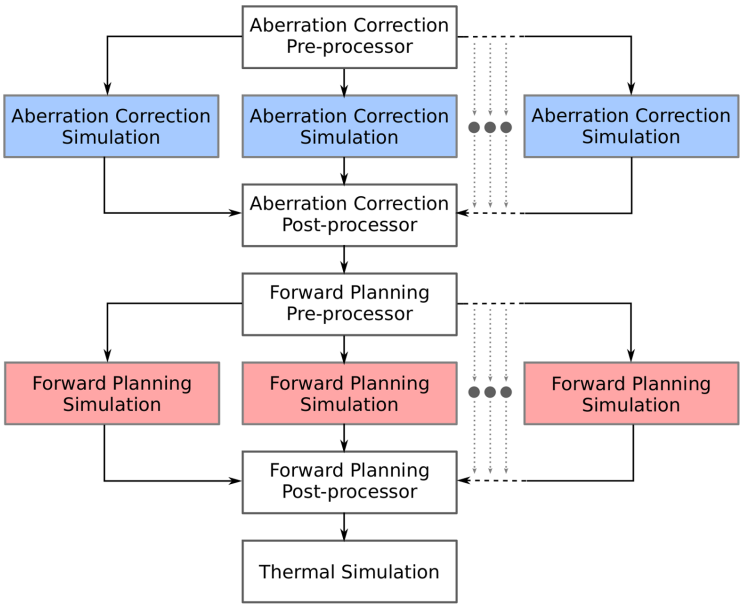


Fig. 1. Typical neurostimulation simulation workflow using the reverse focusing for aberration corrections. After pre-processing, reverse ultrasound propagation simulation from particular targets are executed. After aberration correction, forward ultrasound simulations are executed to calculate energy deposition. Finally, a thermal simulation is executed to estimate overall heat deposition and temperature rise in the tissue.

A simulation of such a size can be executed by the distributed CPU code running on 1 to 16 nodes. The number of sonications is usually between 1 and 32. After all aberration correction simulations have completed, the aberration correction post-processor joins the results from the previous step and derives corrected transducer signals. The forward planning pre-processor consequently generates new ultrasound simulation files. Both these tasks require a single node only. The forward planning simulations use the same code as the aberration correction simulations but with different driving signal. The execution times are therefore very similar. This stage is closed by the forward planning post-processors, which collects the heat deposition from particular sonications. Again, a single node is sufficient for this task. Finally, the thermal simulation is executed to calculate the temperature rise in the brain and evaluate the treatment outcome. This code uses a single simulation node only.

The target infrastructure used for the evaluation of the planning capabilities is based on a 16 node partition of the Anselm supercomputer run by the IT4Innovations National Supercomputing Centre[1]. Each node is equipped with two 8-core Sandy Bridge processors, 64 GB RAM and 40 GB InfiniBand connection. The supercomputer is managed by the PBS Pro scheduler with a backfilling job scheduling.

[1] https://docs.it4i.cz/anselm/compute-nodes/.

2.2 Optimization Criteria

In general, k-Dispatch aims to find the best execution parameters for particular tasks to minimize the *overall execution time*, *computational cost* and *queuing times*. This is achieved by using the database maintaining information about previously completed tasks that allows us to approximate execution time and amount of resources for new workflows, and cluster simulations that evaluate queuing times for given execution parameters.

The optimization criteria can be minimized independently using a multi-objective approaches to create a Pareto front, or aggregated into a single criterion by associated weights. To limit the time complexity of the optimization process, the following aggregated criteria f is used:

$$f = w_t * (t + q) + w_c * c \tag{1}$$

where w_t and w_c are the weights promoting the execution time and computation cost, respectively, t is the wall clock execution time of all tasks, q is the aggregated queuing time, and c is the overall computation cost. Five different combinations of the weights are evaluated in this paper:

- $w_t = 1 \wedge w_c = 0$ minimizing execution time but ignoring cost,
- $w_t = 0 \wedge w_c = 1$ minimizing execution cost but ignoring time
- $w_t = 0.5 \wedge w_c = 0.5$ looking for a trade-off between execution time and cost,
- $w_t = 0.7 \wedge w_c = 0.3$ preferring execution time to cost,
- $w_t = 0.3 \wedge w_c = 0.7$ preferring execution cost to time.

2.3 Execution Parameters Selection

Before the workflow is submitted to the cluster, the execution parameters for particular tasks have to be set. For this purpose, k-Dispatch employs four modules: (1) *Optimizer* that employs a simple hill climbing algorithm traversing the search space of promising execution parameters, (2) *Interpolator* that provides estimations of execution time and cost for given tasks and their execution parameters, (3) *Simulator* that evaluates the queuing times and calculates the overall execution time of the complete workflow, see Eq. (1), and (4) *Collector* that updates the performance database after the workflow execution.

Let us start with Interpolator which is supposed to estimate the execution time and cost for a given task and execution parameters provided by Optimizer. This module searches the performance database for similar tasks. If there is a direct match, i.e., a task of the same type and size has already been executed, the records are filtered by the age and sorted according to the execution parameters used. If there are multiple records for the same execution parameters, the median value is taken. Consequently, a strong scaling plot is constructed, see Fig. 2. From this plot, it is straightforward to estimate the execution time and cost for given execution parameters (number of nodes in this case). If some values are missing, e.g., Optimizer asks about an odd number of compute nodes, the execution time and cost are interpolated. If there is not a direct match, which indicates

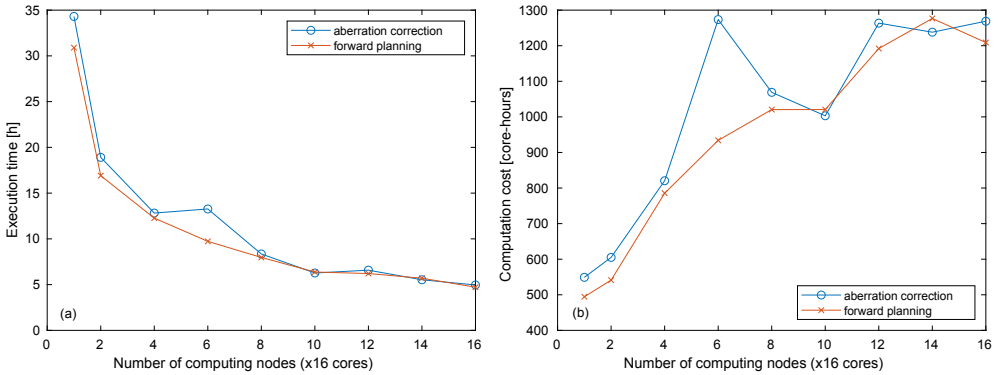


Fig. 2. Strong scaling of the (a) execution time and (b) execution cost for aberration correction and forward planning simulations. The anomalies in the plots are caused by unbalanced work distribution over compute nodes.

such a task has not been executed before, a dual interpolation is performed. Interpolator searches all tasks of similar size, constructs multiple scaling plots, and interpolates between them. If the interpolation fails due to oscillations of the interpolation polynomial or a low number of records found, a default wall clock time with the associated cost are returned. This is, however, a very rare situation, since the more tasks get executed, the more records are in the database, and the more precise interpolations are.

Once the execution parameters have been set for all tasks, the workflow schedule is handed over to Simulator. Although many job schedulers offer some kind of queuing time estimation, the number of such requests is very limited, e.g., one per 5 min. Therefore, the actual state of the cluster is downloaded and fed into the Alea simulator. After the evaluation, the overall execution time (makespan) is calculated as the sum of the estimated execution and queuing times over all tasks. Since the queuing times are not included in the execution cost, the simulator only returns the overall time. Let us note that on a real system, the execution times of particular tasks may slightly vary due to cluster workload (network and I/O congestion, varying temperature and clock frequency between nodes, etc.). These oscillations are, however, neglected since being usually below 5%, and if there is a significant transient performance drop, the k-Dispatch monitoring module detects such an anomaly and terminates affected tasks.

Optimizer tries to select appropriate execution parameters to minimize the aggregated criteria for the whole workflow, see Fig. 3. The parameters of the tasks may be initialized randomly, using the recently best known values, or by individual optimization of each task. In order to search the space, the execution parameters are slightly perturbed in every iteration, the compute time and cost estimated by Interpolator, and the makespan evaluated by Simulator. After a predefined number of iterations, the best workflow parameters are used to submit the workflow to the cluster. In order to broaden the performance database, there is a small probability that Optimizer selects such execution parameters that have

not been tried before. This helps adapt the workflow schedules to changes in the cluster software configuration, hardware upgrades, long-term performance anomalies, etc. After the workflow has been executed by the cluster, the amount of resources used is stored in the performance database along with the actual execution time and cost.

Figure 3 shows two examples of the execution plans designed by k-Dispatch. In the first example, all aberration correction simulations use the same amount of resources, which may yield the best value of the optimization criteria for individual tasks. This may however lead to a suboptimal execution schedule when the cluster size is limited. A better solution may be to use 2 nodes for first 16 tasks and 4 nodes for the last four. Should the number of nodes assigned per task happen not to be a divider of the cluster size, there would be wasted computing slots. The main objective of k-Dispatch is to prevent such inefficiencies.

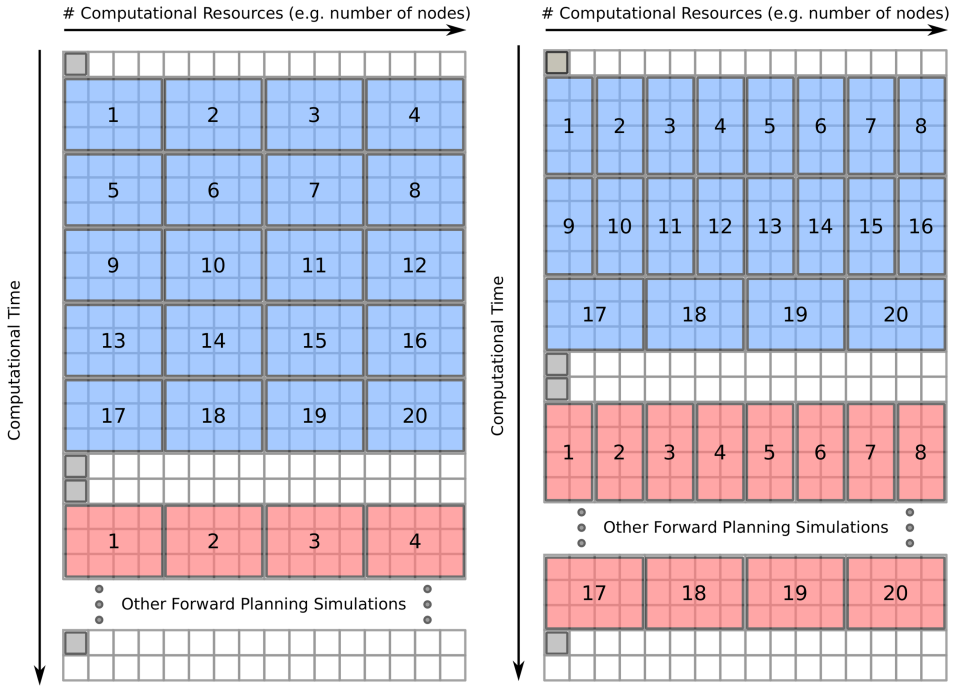


Fig. 3. Example of two different execution plans of the neurostimulation workflow on a 16-node cluster. On the left, every job was optimized independently neglecting the queuing times. On the right, the complete workflow was optimized leading to different resources allocations for particular simulations to minimize the overall execution time.

3 Simulator

As the basis for our workflow scheduling simulator, we have adopted the *Alea* job scheduling simulator [10]. Alea is a platform-independent event-driven discrete time simulator written in Java built on the top of the GridSim simulation toolkit [16]. GridSim provides the basic functionality to model various entities in a simulated computing system, as well as methods to handle the simulation events. The behavior of the simulator is driven by an event-passing protocol. For each simulated event, such as job arrival or completion, one message defining this event is created. It contains the identifier of the message recipient, the type of the event, the time when the event will occur and the message data. Alea extends this basic GridSim's functionality and provides a model allowing for detailed simulation of the whole scheduling process in a typical HPC/HTC system. To do that, Alea either extends existing GridSim classes or it provides new classes on its own, especially the core `Scheduler` class and classes responsible for data parsing and collection/visualization of simulation results.

Figure 4 shows the overall scheme of the Alea simulator, where boxes denote major functional parts and arrows express communication and/or data exchange within the simulator.

3.1 General Description

The main part of the simulator is the centralized job scheduler. The scheduler manages the communication with other parts of the simulator. It maintains important data structures such as job queue(s). Job scheduling decisions are performed by scheduling algorithms that can be easily added using existing simple interfaces. Furthermore, scheduling process can be further influenced by using additional system policies, e.g., the fair-sharing policy which dynamically prioritizes job queue(s). Also, system queues including various limits that further refine how various job classes are handled are supported. Additional parts simulate the actual computing infrastructure, including the emulation of machine

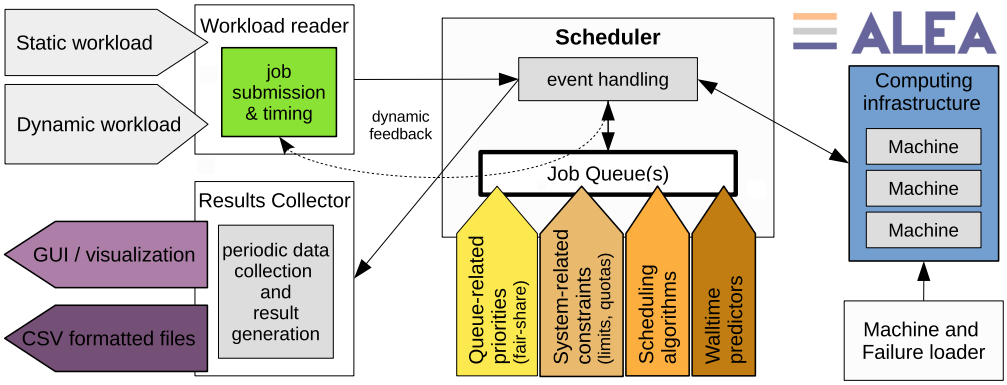


Fig. 4. Main components of the Alea jobs scheduling simulator.

failures/restarts. Workload readers are used to feed the simulator with input data about jobs being executed and the simulator also provides means for visualization and generation of simulation outputs. Alea is freely available at GitHub [1].

The primary benefit of Alea is that it allows for realistic testing of workload execution subject to (different) scheduling policies or setups of computing systems. It models all important features that have significant impact on the performance of the system. These features enable us to mimic real-life systems properly with a reasonably high realism [11].

3.2 Workflow Support

One of the main contributions of this work is the development of workflow (DAG) execution support in Alea. This has been mostly achieved by modifying two components in the simulator: the workload reader and the scheduler. Workload reader has been modified to properly parse new DAG-compatible workload format (see Sect. 3.3). In the scheduler, new logic has been added to properly handle inter job dependencies. The most important modification was the addition of a new *hold queue* for all jobs with unfinished predecessors. Using this queue, these jobs are excluded from the normal scheduling loop until all their dependencies are resolved, i.e., all their predecessors are completed.

The list of all *unfinished predecessors* is kept up-to-date throughout the execution of DAGs. Once a job completes its execution, it is removed from the list of *unfinished predecessors* and the *hold queue* is scanned to check if any job now has all of its precedence constraints satisfied. If so, this job is immediately moved to the normal *scheduling queue* where it waits until it is actually started. Figure 5 demonstrates how the inter-DAG dependencies are handled, using the *hold queue* together with the list of all *unfinished predecessors*.

Otherwise, only minor changes were necessary in Alea, e.g., job definition as well as simulation outputs have been extended to reflect that each job (task) may have predecessors.

3.3 DAG Workload Format

For convenience, we use slightly extended Standard Workload Format (SWF) which is used in the Parallel Workloads Archive [4]. SWF is a simple format where each workload is stored in a single ASCII file [5]. Each job (or task) is represented by a single line in the file. Lines contain a predefined number of fields, which are mostly integers, separated by whitespace. Fields that are irrelevant for a specific log or model appear with a value of -1. To represent DAGs, we have extended the standard 18 entries with two new entries that allow us to distinct which line corresponds to which DAG (`DAG_id`) and which task within a given DAG this job represents (`task_id`). Also, we have modified the existing `Preceding Job Number` such that it can point to more than one job (task). If a given job has more than one predecessor in the DAG, then `&` character is used

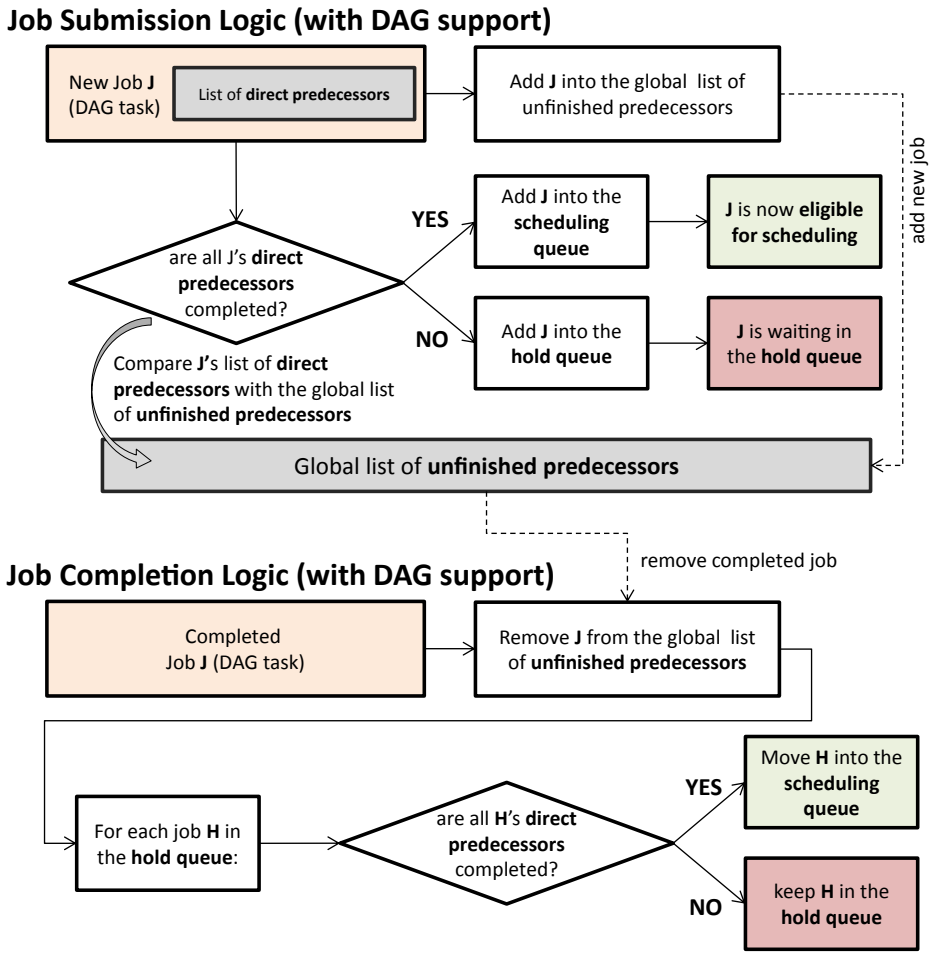


Fig. 5. Added logic handling correct execution order of DAG-like workflows within Alea simulator. Job dependencies are checked during new job arrival (top) and updated once a job completes its execution (bottom). At this point, waiting jobs from the *hold queue* are moved to the *scheduling queue* if their dependencies are satisfied.

to concatenate the list of these predecessor IDs. For example, `1&2&3` means that the given job can only start once jobs 1, 2 and 3 are all completed[2].

4 Alea Simulation Capabilities

Alea job scheduling simulator is well known for its capability to simulate and also optimize various setups of HPC/HTC systems [2,10]. In this section we will demonstrate the novel DAG-oriented simulation capability. We illustrate how the newly extended Alea simulator can be used to evaluate various setups of ultrasound simulations in order to choose the best available setup.

[2] This string corresponds to the list of *direct predecessors* used in Fig. 5.

As discussed in Sect. 2, k-Dispatch keeps its internal performance database to predict rather accurately what the execution time needed to complete such a task will be. The problem is, that task-level optimization does not guarantee that good results will be achieved. Instead, we need to optimize the execution parameters of the whole workflow(s) to achieve good performance. An example of such situation has been shown in Fig. 3. Also, as the available computing infrastructure may change over the time, k-Dispatch must be able to adapt existing scheduling plans once, e.g., the amount of available resources has changed.

In the first example, we use Alea to model and execute (simulate) the problem depicted previously in Fig. 3. In this case, the same workflow uses two different sets of task execution parameters which influence the total execution time. The Gantt chart presented in Fig. 6 shows the execution of all tasks (Y-axis) over the time (X-axis). Clearly, these results correspond to the illustration used in Fig. 3.

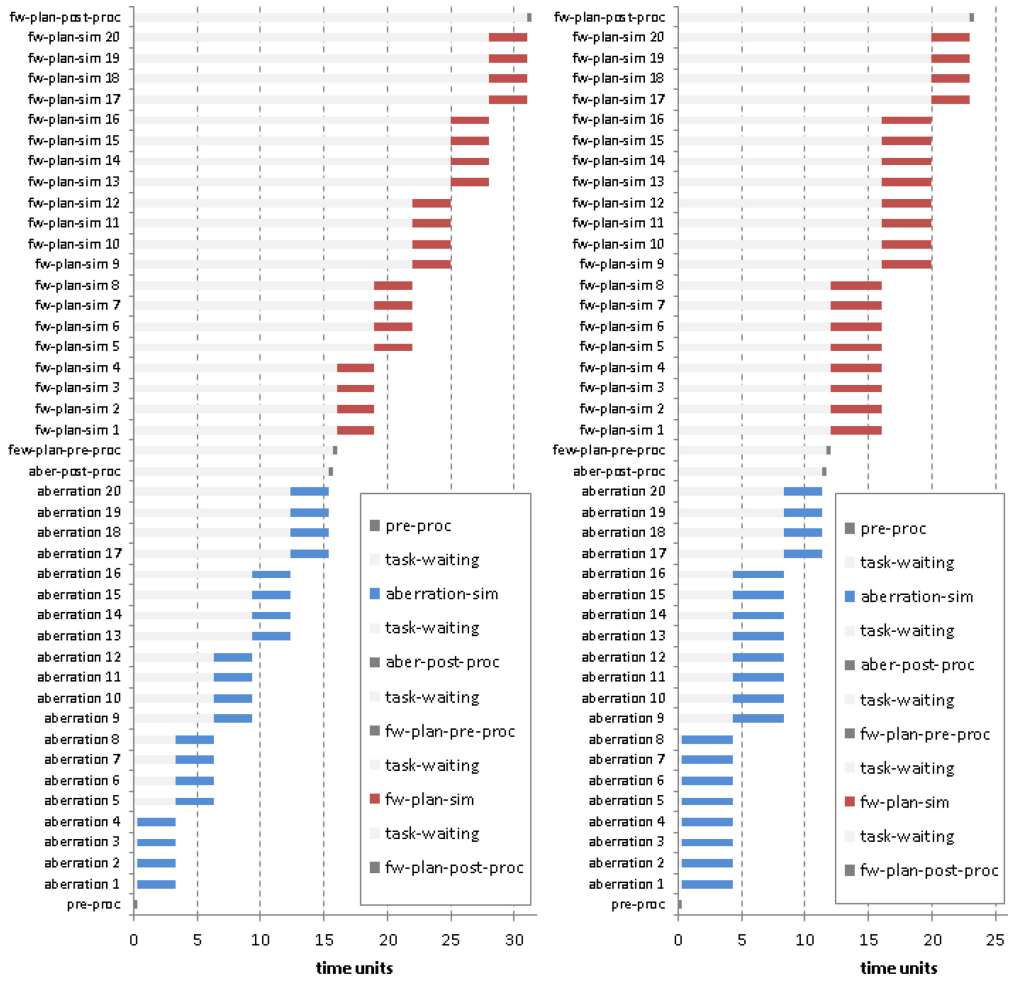


Fig. 6. Alea simulator used to measure the impact of task-level (left) vs. workflow-level (right) optimization on the total workflow execution time (makespan).

We can observe, that task-level execution time optimization (see the Gantt chart on the left in Fig. 6) is suboptimal compared to the workflow-level optimization (right). In this particular case, the second (right) scenario decreased the total execution time from 31 time units to just 23, representing 26% improvement by means of cost/time.

Of course, there are more scenarios that can be modelled and analysed in Alea. For example, we may analyse how several workflows will perform when executed simultaneously. Such an experiment may be very useful when finding the trade-off between total execution time and cost. In other words, we can use such experiment to see how many resources are needed to compute N workflows in a given time T. We illustrate this situation in Fig. 7. Here we show the impact of concurrently executed workflows on the queuing time and the total execution time (makespan). Also, the impact of varying number of available number of CPU cores (i.e., the cost) is shown.

For this demonstration, we use identical workflows, each consisting of 3 tasks that are directly dependent upon each other[3]. We start with a scenario where we execute 3 such workflows together (see the top chart in Fig. 7). As we can see, the system (16 nodes) is capable of executing all 3 workflows concurrently. The situation changes once we add the fourth workflow (see the middle chart in Fig. 7). In this case, the system is not large enough to execute all four tasks no. 2 simultaneously, i.e., the task no. 2 from the fourth workflow (denoted as DAG-4 [2]) has to wait until at least one task no. 2 of the remaining workflows is completed. As a result, the makespan gets higher. As illustrated in the bottom chart in Fig. 7, the makespan gets even worse once we shrink the available resources to a half (8 nodes).

Clearly, the Alea simulator allows us to compare various alternatives and decide which combination of parameters and/or what cost leads to acceptable makespan. Simulations like these can be then used by the k-Dispatch's scheduling module when deciding which parameter settings to choose for the tasks that must be scheduled.

Finally, we would like to briefly mention the simulation overhead of Alea when dealing with DAG-like workflows. Naturally, we need the simulator to be fast when emulating the execution of realistically complex workflows. Therefore, we have performed a set of experiments, where we measured the time needed to perform a simulation. We investigated the influence of both DAG complexity (number of tasks within a workflow) and the number of DAGs being simulated simultaneously[4]. The results are shown in Fig. 8. Simulations use various number of DAGs (up to 64 DAGs) while each such DAG has different number of tasks (2, 4, 8, 16, 32 or 64 tasks per DAG). The figure shows that the simulator is capable to simulate DAG executions in a reasonable amount of time. Even with the most demanding setup (64 DAGs, each having 64 tasks per DAG) the total simulation time is below 2.5 s.

[3] The corresponding DAG looks like this: task 1→ task 2→ task 3.

[4] The experiments were performed on a machine running Windows 10 with Intel Core i7-7500U CPU running at 2.7 Ghz and having 8 GB of RAM.

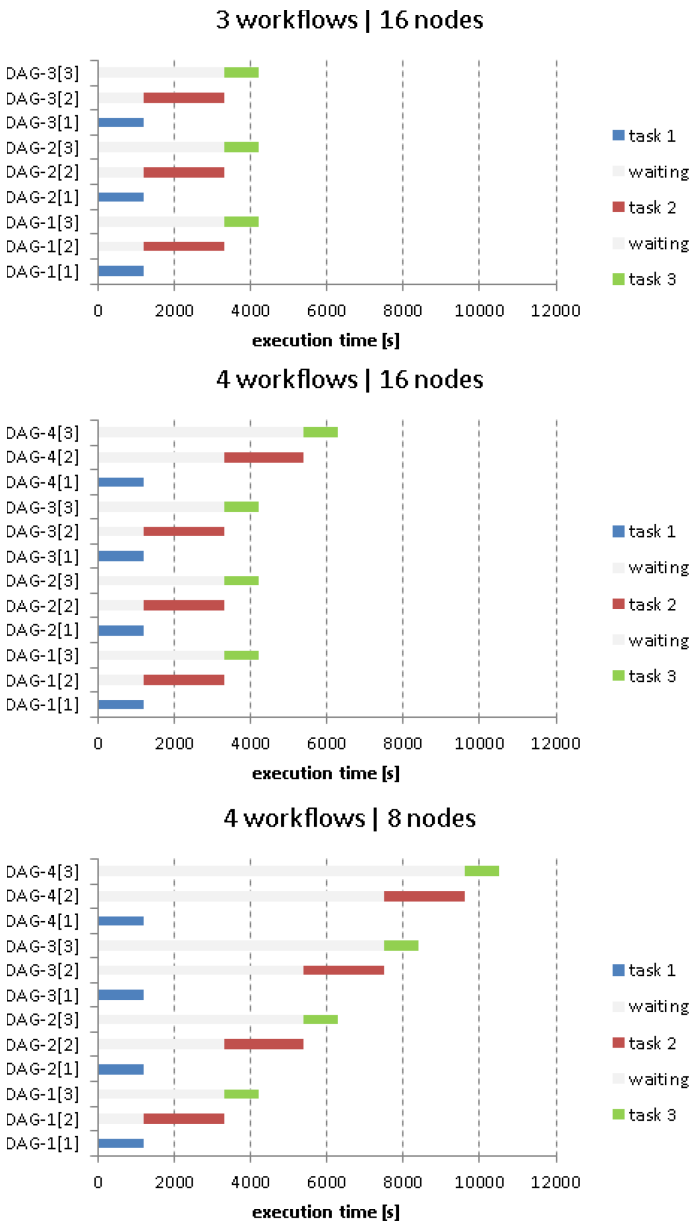


Fig. 7. Makespan and wait time (queuing time) as impacted by the number of concurrently executed workflows and the size of the infrastructure.

This means that Alea is capable of evaluating many different workflow parameter setups within just few seconds. Such a small overhead is clearly no issue for the k-Dispatch workflow management system.

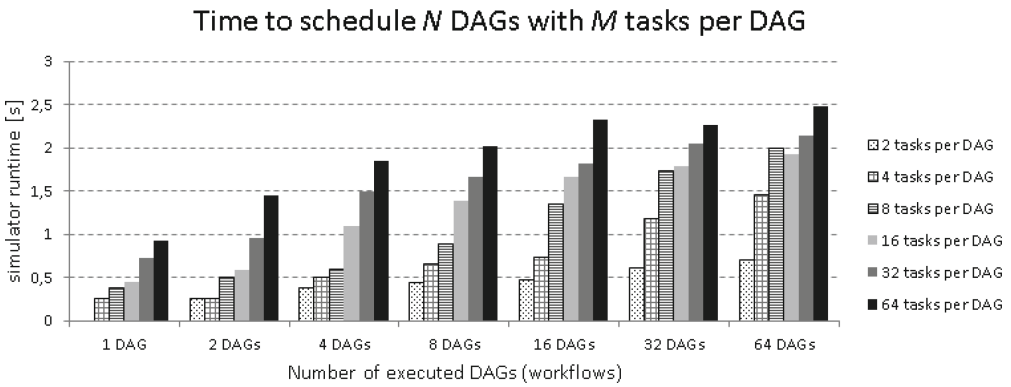


Fig. 8. The time needed to execute one simulation with respect to the number and complexity of simulated workflows (DAGs).

5 Conclusions

In this paper, we have described the scheduling problem related to proper setup of complex biomedical ultrasound workflows. Moreover, we have provided an example of real life-based problem instances (workload describing DAG-like workflows) and developed an extension for the open source job scheduling simulator Alea. Using this extension, basic DAG-like workflows can be simulated and the impact of varying workflow execution parameters (number of tasks and their requirements) can be quickly analysed. Also, thanks to the main focus of the Alea simulator, detailed system-oriented setups and resource policies (e.g., scheduling algorithms, queue setup or fair-share priorities) can be easily emulated, thus providing more realistic outputs and performance predictions.

In the future, we would like to integrate this functionality with the k-Dispatch workflow management system. The newly extended Alea simulator with DAG scheduling support can be freely obtained on GitHub [1]. Also, we invite other researchers to look into the data provided along with this paper that describe real-life based workflows used within the international ultrasonic community. These workloads include the examples used in this paper and will be available at the website of the workshop[5].

This work has a significant impact on the biomedical ultrasound community. Not only the clinicians do not have to bother with selecting suitable computing facilities, deploying simulation codes, moving data forth and back, job submission, execution and monitoring, but their workflows are executed efficiently minimizing the execution time and cost. This all is done without any user intervention, actually, the users do not even know such a process exists.

Acknowledgments. We kindly acknowledge the support provided by the project Reg. No. CZ.02.1.01/0.0/0.0/16_013/0001797 co-funded by the Ministry of Education, Youth and Sports of the Czech Republic. Computational resources were supplied by

[5] http://jsspp.org/workload/.

the project "e-Infrastruktura CZ" (e-INFRA LM2018140) provided within the program Projects of Large Research, Development and Innovations Infrastructures.

This work was also supported by The Ministry of Education, Youth and Sports from the National Programme of Sustainability (NPU II) project IT4Innovations excellence in science - LQ1602 and by the IT4Innovations infrastructure which is supported from the Large Infrastructures for Research, Experimental Development and Innovations project IT4Innovations National Supercomputing Center - LM2015070. This project has received funding from the European Union's Horizon 2020 research and innovation programme H2020 ICT 2016–2017 under grant agreement No 732411 and is an initiative of the Photonics Public Private Partnership.

References

1. Alea job scheduling simulator, May 2020. https://github.com/aleasimulator/alea/tree/FIT
2. Azevedo, F., Klusáček, D., Suter, F.: Improving fairness in a large scale HTC system through workload analysis and simulation. In: Yahyapour, R. (ed.) Euro-Par 2019. LNCS, vol. 11725, pp. 129–141. Springer, Cham (2019). https://doi.org/10.1007/978-3-030-29400-7_10
3. Deelman, E., et al.: Pegasus: a workflow management system for science automation. Future Gener. Comput. Syst. **46**, 17–35 (2014)
4. Feitelson, D.G.: Parallel workloads archive, May 2020. http://www.cs.huji.ac.il/labs/parallel/workload/
5. Feitelson, D.G.: The standard workload format, May 2020. https://www.cse.huji.ac.il/labs/parallel/workload/swf.html
6. Georgiou, P.S., et al.: Beam distortion due to gold fiducial markers during salvage high-intensity focused ultrasound in the prostate. Med. Phys. **44**(2), 679–693 (2017)
7. Godlove, D.: Singularity. In Proceedings of the Practice and Experience. In: Advanced Research Computing on Rise of the Machines (learning), pp. 1–4, New York, NY, USA. ACM, July 2019
8. HTCondor. HTCondor - high throughput computing (2019)
9. Jaros, M., Treeby, B.E., Georgiou, P., Jaros, J.: k-Dispatch: a workflow management system for the automated execution of biomedical ultrasound simulations on remote computing resources. In: Proceedings of the Platform for Advanced Scientific Computing Conference, PASC 2020, New York, NY, USA. Association for Computing Machinery (2020)
10. Klusáček, D., Soysal, M., Suter, F.: Alea – complex job scheduling simulator. In: Wyrzykowski, R., Deelman, E., Dongarra, J., Karczewski, K. (eds.) PPAM 2019. LNCS, vol. 12044, pp. 217–229. Springer, Cham (2020). https://doi.org/10.1007/978-3-030-43222-5_19
11. Klusáček, D., Tóth, Š.: On interactions among scheduling policies: finding efficient queue setup using high-resolution simulations. In: Silva, F., Dutra, I., Santos Costa, V. (eds.) Euro-Par 2014. LNCS, vol. 8632, pp. 138–149. Springer, Cham (2014). https://doi.org/10.1007/978-3-319-09873-9_12
12. Konofagou, E.E.: Trespassing the barrier of the brain with ultrasound. Acoust. Today **13**(4), 21–26 (2017)
13. Merkel, D.: Docker: lightweight Linux containers for consistent development and deployment. Linux J. **2014**(239), 2 (2014)

14. Robert, Y.: Task graph scheduling. In: Padua, D. (ed.) Encyclopedia of Parallel Computing, pp. 2013–2025. Springer, Boston (2011). https://doi.org/10.1007/978-0-387-09766-4_42
15. Sarkar, V.: Partitioning and scheduling parallel programs for multiprocessors. In: Research Monographs in Parallel and Distributed Computing, pp. 1–183. MIT Press, Cambridge (1989)
16. Sulistio, A., Cibej, U., Venugopal, S., Robic, B., Buyya, R.: A toolkit for modelling and simulating data Grids: an extension to GridSim. Concurr. Comput. Pract. Exp. **20**(13), 1591–1609 (2008)
17. Szabo, T.L.: Diagnostic Ultrasound Imaging: Inside Out (2014)
18. Treeby, B.E., Cox, B.T.: k-Wave: MATLAB toolbox for the simulation and reconstruction of photoacoustic wave fields. J. Biomed. Optics **15**(2), 021–314 (2010)
19. Treeby, B.E., Jaros, J., Martin, E., Cox, B.T.: From biology to bytes: predicting the path of ultrasound waves through the human body. Acoust. Today **15**(2), 36–44 (2019)

Related Paper II

k-Dispatch: A Workflow Management System for the Automated Execution of Biomedical Ultrasound Simulations on Remote Computing Resources

JAROS Marta, TREEBY Bradley E., GEORGIOU Panayiotis and JAROS Jiri

Proceedings of the Platform for Advanced Scientific Computing Conference, PASC 2020
DOI: 10.1145/3394277.3401854

k-Dispatch: A Workflow Management System for the Automated Execution of Biomedical Ultrasound Simulations on Remote Computing Resources

Marta Jaros*
Brno University of Technology, Faculty of Information Technology, Centre of Excellence IT4Innovations
Brno, Czech Republic
martajaros@fit.vutbr.cz

Bradley E. Treeby
University College London, Medical Physics and Biomedical Engineering, Biomedical Ultrasound Group
London, United Kingdom
b.treeby@ucl.ac.uk

Panayiotis Georgiou
University College London, Medical Physics and Biomedical Engineering, Biomedical Ultrasound Group
London, United Kingdom
p.s.georgiou@ucl.ac.uk

Jiri Jaros
Brno University of Technology, Faculty of Information Technology, Centre of Excellence IT4Innovations
Brno, Czech Republic
jarosjir@fit.vutbr.cz

ABSTRACT

Therapeutic ultrasound is increasingly being used for applications in oncology, drug delivery, and neurostimulation. In order to adapt the treatment procedures to patient needs, complex physical models have to be evaluated prior to the treatment. These models, however, require intensive computations that can only be satisfied by cloud and HPC facilities. Unfortunately, employing these facilities and executing the required computations is not straightforward even for experienced developers.

k-Dispatch is a novel workflow management system aimed at modelling biomedical ultrasound procedures using the open-source k-Wave acoustic toolbox. It allows ultrasound procedures to be uploaded with a single click and provides a notification when the result is ready for download. Inside k-Dispatch, there is a complex workflow management system which decodes the workflow graph, optimizes the workflow execution parameters, submits jobs to remote computing facilities, monitors their progress, and logs the consumed core hours. In this paper, the architecture and deployment of k-Dispatch are discussed, including the approach used for workflow optimization. A key innovation is the use of previous performance data to automatically select the utilised hardware and execution parameters. A review of related work is also given, including workflow management systems, batch schedulers, and cluster simulators.

CCS CONCEPTS

• **Theory of computation → Scheduling algorithms**.

*corresponding author.

PASC '20, June 29-July 1, 2020, Geneva, Switzerland

ACM ISBN 978-1-4503-7993-9/20/06…$15.00
https://doi.org/10.1145/3394277.3401854

KEYWORDS

Workflow management system, middleware, HPC as a service, biomedical workflows, automation, container, personalised medicine.

ACM Reference Format:
Marta Jaros, Bradley E. Treeby, Panayiotis Georgiou, and Jiri Jaros. 2020. k-Dispatch: A Workflow Management System for the Automated Execution of Biomedical Ultrasound Simulations on Remote Computing Resources. In *Proceedings of the Platform for Advanced Scientific Computing Conference (PASC '20), June 29-July 1, 2020, Geneva, Switzerland*. ACM, New York, NY, USA, 10 pages. https://doi.org/10.1145/3394277.3401854

1 INTRODUCTION

Personalised medicine is an emerging approach to patient care in which an individual's characteristics guide clinical decisions aiming for the right treatment for the right patient at the right time [24]. Personalised medicine is particularly important in oncology, where there is an increased emphasis on prevention and precise surgical strategies. Appropriate selection of treatment for patients, to maximise efficacy and minimise toxicity, has long been a fundamental part of routine clinical practice, but until recently clinicians had had limited tools to determine benefits and potential threats.

The applications of biomedical ultrasound sit at the heart of rapidly emerging cancer diagnosis and treatment procedures. In comparison to conventional cancer diagnosis and treatment modalities, such as biopsy, open surgery, radio- and chemo-therapy, ultrasound is non-invasive, non-ionising, and has fewer complications after treatment. When talking about high-intensity focused ultrasound (HIFU) surgery, over 250,000 patients have been treated throughout the world with great success [17]. The number of patients being screened by ultrasound is countless.

In order to adapt therapeutic ultrasound procedures to the patient needs, complex physical models have to be evaluated prior to the treatment to tailor treatment parameters and estimate the treatment outcome. These models can also be used during the treatment to monitor the procedure progress, and after the treatment to evaluate the treatment outcome and predict further disease development. One physical model widely used in the international community is the open source k-Wave toolbox designed for the time-domain

simulation of acoustic waves propagating in tissues in 1, 2, or 3 dimensions [45]. The toolbox has a wide range of functionality, but at its heart is an advanced numerical model that can account for both linear and nonlinear wave propagation, an arbitrary distribution of heterogeneous material parameters, power law acoustic absorption, and the heating induced in tissue.

Over the last decade, k-Wave has attracted a lot of interest amongst biomedical physicists, ultrasonographers, neurologists, oncologists and other clinicians. Numerous applications of k-Wave have been reported, including in photoacoustic breast screening [27], transcranial brain imaging [30], or small animal imaging [35]. k-Wave has also been used for exciting applications in HIFU, including treatment planning of kidney [1, 43], liver [20] and prostate tumour ablations [44], ultrasound neurosurgery and targeted drug delivery [36], and neurostimulation [8].

All these applications, however, require very complex and intensive computations that generally cannot be performed using desktop computers or small servers. Thus, it is essential to offload the computational work to cloud or HPC clusters. Unfortunately, using these facilities and composing the processing workflow is complex even for experienced developers. Therefore, it is crucial to offer clinical end-users a middleware layer that allows a treatment setup and other data to be uploaded using a simple interface (e.g., web page, medical GUI, etc.) and automate the hard work behind the scenes. k-Dispatch is such a tool.

2 K-DISPATCH MISSION

The mission of k-Dispatch is to make HPC and cloud computational resources accessible as a service to clinical end-users with no prior expertise in computational science. On the other hand, k-Dispatch has to remain flexible enough to cover typical ultrasound simulation workflows, ensure a certain level of fault-tolerance, quality of service, and medical data protection. It must also enable user, system and data management, monitoring and accounting. Generally, there are two kinds of k-Dispatch users: (1) ordinary users who want to have their job computed in the simplest possible way, and (2) administrators who manage the software installation, computational services and accounting.

Since treatment planning applications built on k-Wave are considered software as a medical device, strict quality and risk management policies apply to all software used. The users are thus not allowed to use their own binaries but have to use certified ones installed by authorized personnel. This restriction has a dramatic impact on the k-Dispatch philosophy and makes it different from other workflow management systems, see Sec. 6.

Ordinary users are only allowed to create a medical procedure using predefined templates, e.g., HIFU treatment planning, neurostimulation, etc. The file describing the selected procedure along with other data is consequently submitted to k-Dispatch. The first step for k-Dispatch is to decode the procedure and assemble a computational workflow. Next, the true magic comes. k-Dispatch inspects the list of available HPC resources and finds a suitable one. Then it selects the best binaries for given tasks according to the input data size and available hardware. Since the binaries are a priori known and their performance scaling well described, k-Dispatch can optimize the amount of computational resources assigned to particular tasks to minimize several objectives such as computational time, computational cost or waiting times in processing queues. After submission into the computational queues, k-Dispatch periodically monitors all running jobs and detects performance anomalies such as frozen jobs to recover from typical faults. After the complete workflow has been computed, the results are downloaded from the HPC resource back to k-Dispatch and the user is notified that the result is available for download.

The main benefit of k-Dispatch is that ordinary users are completely hidden from the complexity of the HPC or cloud resources. They do not have to know anything about the cluster submission system, job batch schedulers, queues and their policies. Moreover, they do not have to set the number of compute nodes and cores, choose between CPUs and GPUs, or estimate the computation time. Everything is done automatically.

From the perspective of administrators, k-Dispatch collects performance statistics about the executed workflows and learns their performance scaling, logs the usage for different HPC or cloud resources, and detects offline resources and automatically forwards computations to available ones. On the other hand, the administrators are responsible for user management, introducing new workflow templates, installing new software or computational resources, setting up the policies and user priorities, etc.

k-Dispatch is highly optimized for efficient execution of a relatively small number of different workflow templates. Although modifications to the workflow structure are straightforward, introduction of a new task type and/or binaries requires collection of a relatively large performance dataset necessary for optimization of the execution parameters. Therefore, the workflows are currently hard-coded in simple Python classes. In the future, this part may be extended to support a common syntax such as CWL [3] to allow other experienced users or administrators to deploy their codes using k-Dispatch. There is a possibility the optimization core will be released as a plug-in for existing WMS such as Pegasus. Three typical workflows are described below and in Fig. 1:

- **HIFU treatment planning.** HIFU treatments use multiple sonications to ablate the diseased tissue as a single sonication can only cover a volume about the size of a grain of rice. These sonications are displayed in Fig. 1a as columns. The goal is to precisely set the transducer focal positions and the sonication parameters such as the intensity and sonication duration. For every sonication, an acoustic model is evaluated to calculate the energy deposition using a distributed CPU or GPU implementation, typically spanning across 16-32 computing nodes and running for several hours. If time reversal focusing is used, multiple invocations of the acoustic model may be necessary for each sonication. Next, the thermal model is executed to calculate the temperature rise and thermal dose. This typically requires a single GPU for a few minutes.
- **Neurostimulation.** The example neurostimulation workflow, shown in Fig. 1b, is similar to the HIFU workflow except the sonications use much lower intensities such that the wave propagation is linear. The goal is to stimulate the brain but any thermal or mechanical damage must be prevented. In this workflow, the sonications are independent and the

thermal model is used to calculate safety metrics, rather than dose quantities. The acoustic models are typically complex due to the skull and large simulation domains.

- **Photoacoustic imaging**. The example workflow for photoacoustic image reconstruction consists of an a priori known number of iterations of the forward and adjoint acoustic models that reconstruct the tissue structure based on the ultrasound signals sampled at the detectors placed at the surface of the tissue, e.g., breast. The simulations are usually very large and require at least 8 GPUs or 256 computer cores for a few hours. In between the iterations, a simple gradient descent method is executed.

3 SYSTEM ARCHITECTURE

The overall architecture of k-Dispatch is shown in Fig. 2. k-Dispatch consists of three main modules: Web server, Dispatch database and Dispatch core. The user applications, e.g., a stand-alone medical GUI, web app, or Matlab interface, communicate with the Web server using the secured HTTPS protocol and REST API. The Dispatch database holds all the necessary information about the users, submitted workflows, particular jobs, computational resources, available binaries, etc. The Dispatch core is responsible for planning, executing and monitoring submitted workflows. The communication with HPC and cloud facilities is done via SSH and RSYNC protocols.

The architecture of k-Dispatch is generic and modular to enable easy system extensions by adding new workflows, computational resources, interfaces to different job schedulers, etc. Currently, k-Dispatch supports several predefined workflows hard-coded in the structure of the input file and parsing Python classes. Nevertheless,

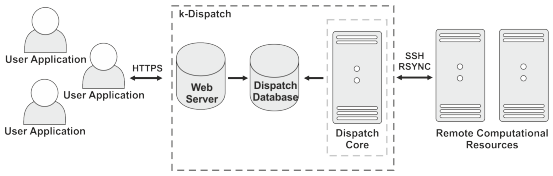


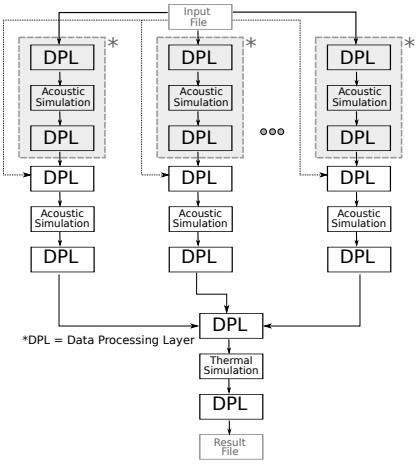
Figure 2: Simplified architecture of k-Dispatch showing three basic modules and their connection to user applications and computational resources.

the file structure is open and the file format is based on the widely adopted HDF5 file format easily readable from Matlab, Octave, Python and other scientific software [15].

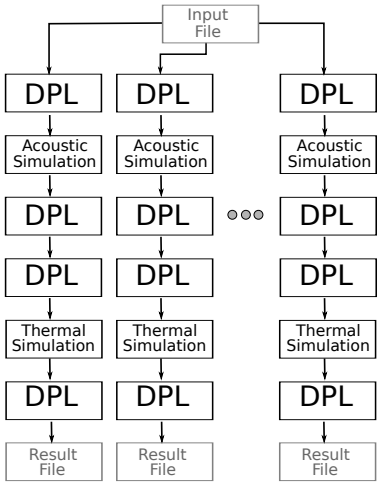
3.1 Web Server and Dispatch Database

The Web server module is based on the Python Flask technology [39] and represents the only entry point to k-Dispatch. The web server communicates with the Dispatch database and with the local storage. The input files with new workflows to compute are stored in the local storage and a new record is made in the database. If the user asks about the status of their workflows, the web server reads appropriate data from the database and reports back to the user. Analogously, when the results are ready for download, the web server sends the result file to the user, updates the database record and clears local storage.

The Dispatch database and the database server is based on the PostgreSQL 10 technology [18]. The database holds all the necessary



(a) An example of HIFU treatment planning workflow.



(b) An example of neurostimulation workflow.

Figure 1: Two generic templates of different ultrasound workflows. The input file holds the patient specific data and simulation parameters, e.g., transducer positions. The star-marked gray blocks may be replicated multiple-times to extend the fundamental workflow structure. DPL blocks denote data processing layers. The procedure results are stored in the result file, usually as an archive with multiple files including essential program logs.

information for planing, executing and monitoring the workflows on remote computational facilities. A simplified entity relationship (ER) diagram of the Dispatch database is shown in Fig 3.

The database tables are divided into four groups. The red group is related to user management. Users form user groups based on their affiliation to companies, hospitals, departments, etc. Users may have different roles and permissions while groups may hold different licenses for k-Dispatch (which includes usage permissions and expiry dates). User groups usually purchase some computational core hours which may be split into several allocations on various computational facilities. The invoices are then stored in the Purchase table.

The green group represents workflows and their execution. When the input data file is parsed, a Workflow record is created together with its Job and Job Dependency records reflecting the workflow execution structure. If a job fails during execution, the job is restarted and a new record in the Restarted Jobs table is created to keep track of faulty jobs and the number of attempts to restart them. All file links used by the job, i.e., submission script, input, output, and log files are stored in the File table.

The blue group reflects supported computer facilities, task types and associated binaries. The information retrieved from these tables is used in the workflow execution optimization.

HPC, HPC Queue and Allocation tables identify the computing facility and the amount of core hours that may be consumed. The Task Type table holds the information about admissible task types, recall the building blocks in Fig. 1, while the Run table specifies available implementations (binaries) for particular task types along with the recipe to generate submission scripts. Let us note that every task type can have several implementations, e.g., a GPU version, a single node version (OpenMP), and a distributed version (MPI). The additional parameters for particular binaries are stored in the Implementation Detail table. Finally, the Allowed Code table specifies where the binary can be executed (which HPC, which queue) and what software modules need to be loaded prior to the execution.

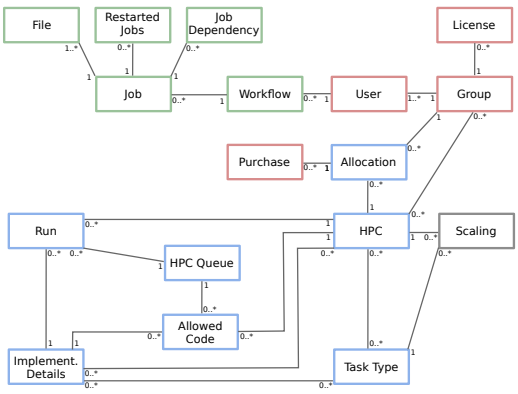


Figure 3: Simplified entity relationship diagram (ERD) for the Dispatch database.

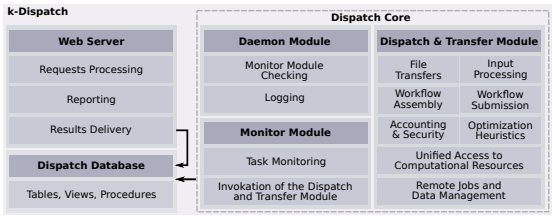


Figure 4: The architecture of k-Dispatch composed of three modules and numerous submodules.

The gray table group comprises information about the performance scaling of particular binaries. The Scaling table collects the performance data about each successfully completed task. This data consists of the binary, HPC and queue identifiers, number of nodes, cores and GPU employed, the size of the simulation domain, basic medium and wave propagation parameters, number of simulation time steps, and execution wallclock time. This data is integral to the workflow execution optimization. In order to provide accounting, the HPC Queue, Allocation and Job tables are used to calculate the amount of consumed core hours and their price, and to update the available group budget.

3.2 Dispatch Core

The Dispatch core is composed of the Daemon module, Monitor module, and the Dispatch and Transfer (D&T) module, each of which is implemented as a Python class. The functionality of these modules is shown in Fig. 4. The Daemon module enables k-Dispatch to be registered as a service in the operating system. The Monitor module periodically scans particular database records and invokes the D&T module to, e.g., plan and submit a new workflow, terminate calculations and delete a workflow, get the status of current jobs, etc. The Monitor module also updates the database records with progress information.

The heart of k-Dispatch lies in the Dispatch and Transfer module. This module unifies the access to different computational resources and their schedulers. This module performs the following operations:

- parses the input file and stores important data for the workflow submission,
- assembles the workflow task graph based on the input data file using predefined Python classes representing a particular workflow structure and its tasks,
- optimizes the workflow execution by finding a suitable allocation on one of the remote computational facilities and assigns appropriate binaries and execution parameters to particular tasks,
- generates HPC-specific job scripts using the Python jinja2 library [37],
- provides data transfers between remote computational facilities and k-Dispatch using the Python fabric library [12],
- (re-)submits, deletes and monitors remote jobs using the fabric library and the batch scheduler commands,

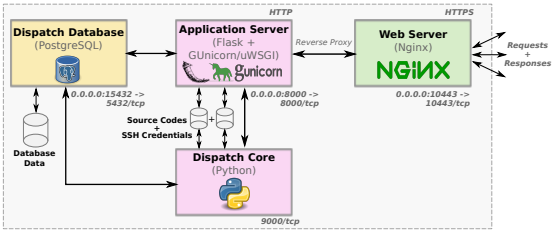


Figure 5: k-Dispatch architecture wrapped into Docker containers and volumes. The text descriptions below individual rectangles show network port mappings used in the current solution.

- detects failures on the remote computational facilities and restarts jobs (restarts only the minimal and necessary amount of dependant jobs, not the whole workflow),
- collects performance scaling data,
- provides accounting by retrieving the amount of consumed core hours directly from the cluster scheduler and multiplying by a price per hour stored in the database, and
- creates and modifies records in the Dispatch database.

3.3 Deployment Using Docker

To simplify the deployment process, maintenance, fault tolerance and data safety of k-Dispatch, a container-based approach using Docker is adopted. Docker [6] is an open-source project based on Linux containers which has undergone significant development and become widespread amongst programmers in recent years. The biggest advantage of this solution is the isolation of k-Dispatch and its dependencies into self-contained units that can run anywhere.

k-Dispatch is split into four Docker containers and three volumes connected via a Docker network deployed by the docker-compose tool, see Fig. 5. These containers individually encapsulate the Dispatch database (depicted in yellow), the Dispatch core and the k-Dispatch's application server Web Server (depicted in pink), and additionally, an Nginx [14] based web server acting as an entry-point for user requests (depicted in green). The Dispatch database stores all persistent data in a dedicated volume. The remaining volumes store k-Dispatch Python source codes and ssh credentials to remote computational facilities, respectively. These volumes are shared between the Dispatch core and the application server to enable easy data updates. The application web server employing the Flask framework cannot be run in the production version without another gateway, e.g., GUnicorn[1] or other uWSGI[2] hosting services, since they only offer HTTP communication. The Nginx container thus adds the required security by providing HTTPS communication.

4 WORKFLOW EXECUTION OPTIMIZATION

This section explains the workflow execution optimization. The goal is to find the best execution parameters for particular tasks to minimize the overall execution time, computational cost and waiting times in the job submission queues. The execution parameters typically only cover the type and the amount of computational resources along with an expected execution time, but can also include the most appropriate queue, desired processor and memory frequencies and other hardware parameters in the future. This information is then written into a submission script and handed over to the HPC or cloud batch scheduler which orchestrates the execution itself. This optimization is only possible thanks to historical performance data collected for the a priori known binaries.

[1]https://gunicorn.org/
[2]https://flask.palletsprojects.com/en/1.0.x/deploying/uwsgi/

4.1 Workflow Definition and Execution Model

The most natural way to define a workflow is to use a Directed Acyclic Graph (DAG), often reffered to as a Task Graph [38], whose nodes are the tasks and the edges are the precedence constraints and data dependencies between tasks. The nodes also encapsulate the task type, input and output files, and the execution parameters.

k-Dispatch allows both task- and data-driven workflows [26] and a static acyclic execution model (see Fig. 6). After the workflow assembly and submission, no conditional behaviors, i.e., dynamic task generation or while loops with an unknown number of iterations, are supported. Since the ultrasound workflows may contain subgraphs that may be either omitted or repeated multiple times, this has to be determined during the planning phase while the final workflow is being assembled.

4.2 Optimization of Execution Parameters

The execution planning process that every HPC job scheduler solves, can be described as a mapping of tasks from the workflow to free time slots and computational resources, see Eq. (1):

$$Q \rightarrow (T' \times R'), T' \subseteq T \wedge R' \subseteq R, \tag{1}$$

where Q is a set of all tasks in the workflow, T and T' are finite sets of all and available time slots, respectively, and R and R' are finite sets of all and idle computation resources at given time slots, respectively. Based on the scheduling policy, each scheduler tries

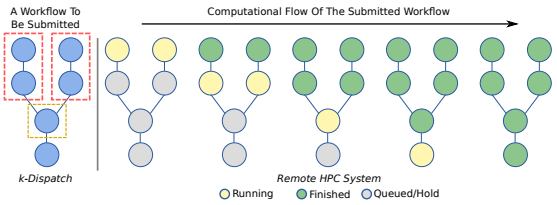


Figure 6: k-Dispatch's execution model. The blue workflow on the left hand side reveals the concurrency and dependencies between tasks. Subgraphs in the red rectangles can be executed concurrently since there are no dependencies between them, however, the tasks inside them have to be executed sequentially. The task in the yellow rectangle has to wait until all red rectangles have finished. The workflows on the right hand side show a possible execution flow on the remote computational machine. The order of the task execution is clearly visible.

to maximize the cluster utilization while guaranteeing quality of service at some level.

HPC and cloud systems often differ in hardware (type of nodes and accelerators, number of cores per node, interconnection, etc.) and software equipment (scheduler and their policy, tools, compilers, etc.). In order to create a favorable execution schedule, the type and amount of resources along with the execution time must be carefully chosen. In many other workflow management systems (WMSs), the end user is responsible for providing this information. This is, however, not viable in our approach and k-Dispatch must automatically find suitable workflow execution parameters. We consider this optimization as the biggest challenge in the development of k-Dispatch.

Since k-Dispatch does not implement its own job scheduler but relies on those used by supported HPC systems, there is a need for cooperation between k-Dispatch and the HPC job scheduler, e.g., PBS Pro or Slurm. The execution parameters are dependent on the current cluster utilization and the list of other queued jobs waiting for execution. Therefore, before the optimization, the current cluster utilization is downloaded along with the actual user priorities, e.g., fairshare priority. This information then guides the optimization process and helps to reduce the queuing times.

There are two approaches to create a heuristic for choosing appropriate execution parameters for particular tasks in the workflow. This heuristic may either be rigid, which always uses predefined default values for the execution parameters of a given simulation code, or adaptive, which takes into account current cluster utilization, code performance scaling and the complexity of the current input data. At the time of writing this paper, only a rigid heuristic has been fully tested. This method always works, however, the throughput and effectiveness of the submission may be limited. The adaptive heuristic, currently under development, can be classified as local or global. The local approach searches for optimal execution parameters of particular tasks independently. While the parameter setting may be suboptimal, the optimization time complexity is linear. On the other hand, the global approach takes into account the dependencies between tasks and can produce better parameters, however, the optimization complexity can become exponential [40].

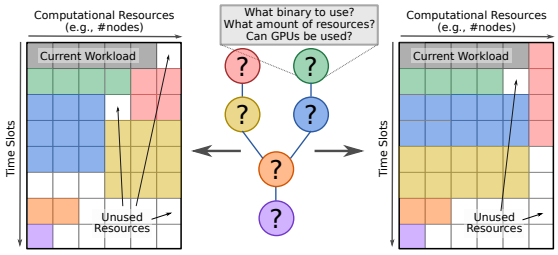


Figure 7: Two examples of the workflow mapping to computational resources and time slots under different execution parameters. Colors show resources occupied by particular tasks. Both mappings take 7 time units to complete. The mapping on the left-hand side is, however, a bit cheaper since it consumes 29 work units while the second one 31. The influence of other jobs coming into the system is not considered.

In both cases, the optimization heuristics assign a specific binary and a set of execution parameters to each task (see Fig. 7) in order to minimize computational time, or cost while not exceeding specified time constraints. The selection of execution parameters is based on the collected performance scaling data. For every task, the size and complexity of the execution can be deduced from the input file. This information can be projected into graphs of strong and weak scaling constructed from the performance data collected for a candidate binary. Finally, the estimated queuing time is taken into account. The workflow with assigned execution parameters is then sent for evaluation either to a simulator or the job scheduler that can provide a more accurate estimation of the launch time. If suitable parameters are found, the workflow is submitted to the job scheduler. The use of adaptive heuristic opens a lot of research questions summarized in Sec. 5. A one-pass local optimization method which uses linear and cubic spline interpolations to find the most suitable amount of computer resources is currently being investigated. Preliminary results show that a cubic spline is a very good model for the strong and weak scaling performance of k-Wave, with errors in the execution time on the order of one percent.

5 CURRENT WORK AND OPEN QUESTIONS

The development of adaptive heuristics for execution parameter optimization opens many new issues. They form the challenges we have been attempting to address and which are described below.

5.1 Data Collection and Processing

The first issue is the collection of performance data. The execution time and cost are defined by strong [2] and weak scaling [22]. However, constructing the scaling for every possible binary, type of resources, and inputs is impossible due to the extreme number of combinations. Therefore, we limited ourselves to only select a small subset of simulation parameters that have the most influence on the computational complexity, e.g., domain size, wave propagation mode, heterogeneity and absorption of the medium. For these parameters we select the most typical values and run benchmark simulations to initially populate the Scaling table. In production, every successfully executed job is used to update this table.

The open question is how to adapt to unseen inputs (e.g., domain sizes), performance fluctuations caused by cluster overloading, or changes in the software and hardware configuration, etc. Both problems can be solved by combinatorial optimization. Having a workflow with a number of tasks (sonications) of the same type and size, the execution parameters can be deliberately perturbed to explore the local neighbourhood of currently optimal parameters. The collected performance data can be also filtered by its age to get the actual state. If a task is encountered that has not been seen before, the optimal parameters can be chosen using interpolation or machine learning.

So far, a set of performance data has been collected for ultrasound simulations and typical domain sizes, various numbers of resources (e.g., number of cores), code implementations (e.g., OpenMP, MPI) and code-specific parameters using the IT4Innovations clusters. Currently, metrics defining the relevance of the records are being

developed considering the age and distance from the investigated simulation size and type.

5.2 Dynamic Cluster Behaviour

As already mentioned, the current HPC utilization may have a strong influence on the optimal values of the workflow execution parameters. Although it does not affect the computational time or cost itself, it may strongly affect the queuing time and lead to exceeding the timespan which users are willing to wait. Usually, jobs asking for small numbers of compute nodes are executed sooner than those asking for a huge portion of the cluster. Of course, this is queue dependent and there may be another queue promoting large jobs.

Another issue is how often to monitor the HPC cluster utilization. The possibility being used now is to take a snapshot before the execution parameter optimization. Nevertheless, what happens if the cluster utilization dramatically changes, e.g., by a burst of high priority jobs from privileged users, or lodging a reservation? The cluster scheduler will recalculate the job priorities and may postpone their execution. If such a situation is detected, the execution parameters of already queued jobs are obsolete and should be altered. The questions under investigation now is how to detect or predict these dramatic changes, how to find some patterns, how to estimate the delay caused, and decide whether it pays off to alter the parameters or not.

5.3 Workflow Parameters Evaluation

Due to many reasons such as the cost of resources, reliability and varying background load, the experimental evaluation of the adaptive heuristics cannot be easily performed on production HPC systems. Moreover, to obtain sensible results, multiple workflows with various execution parameters need to be evaluated under the same and controllable conditions that simulate different real-life scenarios. This is, however, often unachievable.

Therefore, a job scheduling simulator emulating production HPC environments can be used. A short review of the latest simulators is given in Sec. 6. These simulators can provide relatively good estimations of the queuing times. The other alternative is to use dedicated resources (a dedicated queue) and do the experiments there. Nonetheless, this may become quite costly.

6 RELATED WORK

The area of task execution in distributed and heterogeneous systems has been studied for the last decade. There have been many middleware projects developed focusing on running various types of computational workflows on HPC facilities, clouds and grids. We focus mainly on Workflow Management Systems (WMSs) for offloading the task computations to HPC clusters.

The majority of WMSs have been created to address a phenomena called workflow decay. Workflow decay refers to poor reproducibility of scientific workflows which were designed to solve complex scientific problems and accelerate scientific progress. However, scientists often find it difficult to reuse others' workflows. [28]

To characterize WMSs, the following properties may be considered: computational infrastructure (e.g., grids, clouds, HPC clusters), workflow design (e.g., DAG) and means of its composition (e.g., graphical desktop application, web page, command-line tool, programmable interface), types of parallelism, and so on. A novel characterization of WMSs was introduced in [13]. This work uses (1) workflow execution models, (2) heterogeneous computing environments, and (3) data access methods to characterize the workflows. Moreover, the paper classifies 15 state-of-the-art WMSs into an easy-to-use lookup table containing a feature checklist for each WMS.

We especially distinguish WMSs based on the application (type of tasks – long- and short-running) and users' perspective. Many WMSs introduced in this section think about users as scientists or developers. In the daily practice of various user communities, this is simply not the case. WMSs can be deployed in multiple scenarios to serve the needs of various users which places different requirements on the WMS.

Although, not being perfect, WMSs still offer a formal way to define, automate, and repeat multi-step computational procedures. They usually provide services for resource monitoring and management, security and file management, and help scientists to share computing power, databases, tools, etc.

6.1 Workflow Management Systems and Processing Frameworks

Widely used data processing frameworks, especially for big data analytics, include Hadoop [41], a MapReduce-based system for parallel data processing, Apache Spark [47], a system for concurrent processing of heterogeneous data streams, Apache Storm [4], for real-time streaming data processing, and HTCondor [23], for managing compute-intensive jobs. These tools do not allow inter-task dependencies to be specified. Sometimes, such frameworks are implemented within more general WMSs (for example HTCondor/DAGMan [23] and the Pegasus [11] WMS) to schedule and offload the tasks.

When speaking about short-running tasks (one-core, < 1 second), examples of applicable WMSs include Dask [10] and HyperLoom [9]. Since the time needed for resource allocation may create a significant scheduling overhead, these tools usually implement their own scheduling mechanism and heuristics.

Dask handles short running tasks and allows the filesystem usage to be reduced. However, it does not support native pipelining of third-party applications. Dask offers both high-level (e.g., NumPy objects) and low-level programming user interfaces.

HyperLoom is a platform for defining and executing scientific workflows in large-scale high performance computing systems. Its goal is to minimize the overall workflow execution time respecting resource constraints of the tasks and environments. HyperLoom implements an optimized dynamic scheduler that schedules the tasks reactively with a low overhead since the execution time of individual tasks is not known in advance. Moreover, the scheduler respects task dependencies and prioritizes placements that induce the smallest possible inter-node data transfer. Data produced by tasks are kept directly in memory and can be accessed by any other task without additional overhead. HyperLoom allows chaining and execution of third-party applications. HyperLoom enables users to define and execute workflows using its client application. Although HyperLoom was originally designed to be used within HPC

infrastructures, these infrastructures may be unavailable or too expensive especially for small to medium workloads. Therefore, HyperLoom developers started to aim at public cloud providers since performance of their machines is comparable to those in HPC systems. However, the network solutions used in HPC systems offer much higher inter-node throughput. HyperLoom focuses on experienced users as well.

Today's WMSs allow users to compose custom workflows (DAGs) by providing graphical or programmable user interfaces. This determines the potential user of the system. Those WMSs often rely on traditional resource schedulers that are optimized for coarse-grained long-running tasks. The inter-task data transfers are usually performed using a shared distributed filesystem.

FabSim [21] shares functionality with middleware toolkits such as Globus [16] or gLite [19]. However, FabSim is aimed at the experienced computational scientist. The only supported interface is a command line tool which is easy to extend for developers. The key strength of FabSim is its focus on simplifying and accelerating development activities. It simplifies the execution of previously defined workflows as well as the creation of new ones. FabSim does not provide decision-making in terms of planning and monitoring. Its main goal is to simplify researchers' daily tasks.

Taverna [46] is bringing together a range of features to make it easier for users to find, design and execute complex workflows and share them with other people. Therefore, Taverna integrates myExperiment [31] and BioCatalogue [5] and creates an interface to work with these tools. Taverna enables workflows to be run on the user's computer (using Taverna Workbench), on the Taverna server, clouds (for example, on Amazon cloud) and grids, using its own Workflow Management System. Taverna has a huge domain of usage, e.g., in bioinformatics and biology, chemistry, annotation, arts (composing music), astronomy, data mining and analysis, engineering, and so on. Similarly to Taverna, Kepler [26] allows computations over computer clusters and grids. Both provide a graphical environment to help users to perform complex simulation workflows. Kepler is used by projects that operate in bioinformatics, ecological and environmental research, and weather and climate analysis. Both Taverna and Kepler focus on researchers and well-informed users.

Pegasus encompasses a set of technologies that help workflow-based applications execute in a number of different environments including desktops, campus clusters, grids, and clouds. Pegasus bridges the scientific domain and the execution environment by automatically mapping high-level workflow descriptions onto distributed resources. It automatically locates the necessary input data and computational resources necessary for workflow execution. Pegasus enables scientists to construct workflows in abstract terms without worrying about the details of the underlying execution environment. Pegasus has been used in a number of scientific domains including astronomy, bioinformatics, earthquake science, and others. To recover from error, Pegasus provides workflow-level checkpointing.

6.2 Cluster Batch Schedulers

Supercomputing facilities use commercial or open-source job schedulers that contain job scheduling algorithms developed in the past, e.g., backfilling, first come first served (FCFS), etc. For instance, Portable Batch System (PBS) uses the backfilling scheduling algorithm, and considers user and group priorities, and fair-share cluster policies. The IT4Innovations[3] supercomputing center's PBS scheduler gives each job an execution priority first, and then uses this job execution priority to select which job(s) to run. Job execution priority is determined by the queue priority, fairshare priority and eligible time, where the queue priority has the biggest impact. The fair-share priority is calculated on the recent usage of resources per project. Eligible time is the amount of eligible time the job accrued while waiting to run and has the least impact on execution priority. Jobs with higher eligible time gain higher priority. Overall, it is very beneficial to specify the walltime when submitting jobs as this enables better scheduling and better resource usage. Backfilling is an FCFS approach that is improved by increasing the utilization of the system resources and by decreasing the average waiting time in the queue. Backfilling fits smaller jobs in front of higher-priority jobs if it is possible, in such a way that the higher-priority jobs are not delayed. This prevents resources from becoming idle when the top job (job with the highest execution priority) cannot run. [42] A backfilling scheduling algorithm is used by IT4Innovations' clusters.

Another widely employed workload manager is Slurm used by, e.g., Chinese Sunway TaihuLight or Swiss Piz Daint. Slurm performs a best-fit algorithm based on Hilbert curve scheduling or fat tree network topology in order to optimize the locality of task assignments on parallel computers [34]. However, as mentioned before, developers of WMSs sometimes implement their own schedulers, operating above those used in supercomputing centers. Another example is the NCSA (National Center for Supercomputing Applications at the University of Illinois) scheduler tool [32] designed for Blue Waters and other HPC systems. Many HPC facilities limit the number of jobs per user to prevent queues from becoming cumbersome. The NCSA scheduler allows users to aggregate single-core jobs as a single batch and jobs share the node between applications using a simple configuration file. The scheduler allows queuing jobs and manages efficiently independent single-core jobs, can bundle OpenMP (Open Multi-Processing) single-node jobs but cannot bundle MPI (Message Passing Interface) jobs.

6.3 Cluster Simulators

The following text gives a short review of job scheduling simulators. Due to many reasons such as the cost of resources, the reliability, the varying background load or the dynamic cluster behaviour, experimental evaluation generally cannot be performed on real systems. Moreover, to obtain reliable results, multiple workflows with various run configurations need to be performed using the same and controllable conditions that simulate different real-life scenarios which is, however, often not possible.

Simple job scheduler simulators often provide a detailed model of the queuing behaviour as the jobs arrive at the system upon submission, wait for available resources, start their execution, and eventually leave the system upon their completion.

For example, PySS [29] is a trace-driven scheduler simulator. It implements a number of scheduling algorithms, including several backfilling ones. The problem with simple simulators is that they

[3] Czech national supercomputing center, https://www.it4i.cz/

do not really model the target HPC system or the runtime behavior of the applications. PySS takes the job runtime directly from the job trace, although in reality a job's runtime is affected by the specific resources allocated to the job and by the application's runtime behavior, which can be affected by other jobs running simultaneously [33]. Thus, more sophisticated simulators need to be used instead.

Alea 4 [25] is an event-based grid and cluster scheduling simulator that uses the GridSim toolkit [7]. The simulator is able to deal with common problems related to job scheduling like the heterogeneity of jobs, resources, and dynamic runtime changes such as the arrival of new jobs or resource failures and restarts. The main part of the simulator is a complex scheduler which incorporates several common scheduling algorithms working either on the queue or the schedule (plan) based principle. The latest version of Alea uses a dynamic workload adjustment technique enabling user-to-system interactions to be modeled properly. The input is still a static workload (historical workload traces extracted from the HPC system itself, or from a public workload trace repository) but transformed into a dynamic one afterwards.

Performance Prediction Toolkit (PPT) [33] is a full-scale HPC simulator. It can use synthetic workload models or adopt job traces from existing HPC workload archives. The simulator implements several commonly used scheduling algorithms, however, it does not include backfilling algorithms.

Other complex frameworks for studying grids, clouds, HPC or peer-to-peer systems have been developed. However, the majority of these projects seem to be inactive or abandoned. [25]

6.4 Summary

After a detailed review of current WMSs, k-Dispatch seems to be unique in several aspects. Unlike many other low level WMSs, it does not require the end users to have personal access to remote computational facilities (user accounts). k-Dispatch uses its own credentials to access several computing facilities and provides accounting for the end users.

k-Dispatch is also oriented towards workflows composed of large long running jobs. For these jobs, the optimization of execution parameters is crucial to reduce the computation cost, minimize queuing times and offer some estimation of the delivery time. Since the set of possible binaries is limited, statistically relevant performance data can be captured and consequently used for estimations. As far as we have seen, there are no similar tools available, and instead users are generally responsible for providing appropriate parameters themselves.

For the job submission and execution, the PBS and Slurm interfaces provide enough functionality. However, for the evaluation of hundreds of jobs per second, they may be too slow. As a suitable tool for quick execution parameter evaluation, the ALEA simulator appears to be a good candidate and has already been under evaluation.

7 CONCLUSIONS AND FUTURE WORK

Over the last few decades, numerous middleware projects have been developed focusing on running user-defined workflows on various computational platforms including local desktop computers, middle-sized servers up to huge and heterogeneous supercomputing facilities and clouds. These tools are developed as stand-alone desktop applications, web applications or importable libraries which determines the level of interactivity with end users.

Unfortunately, all these tools focus primarily on experienced users from various scientific domains. Despite the orchestration, monitoring and data management being provided by the tools, the users have to compose their own workflows, specify the execution parameters and provide their own binaries manually. This is, however, unfeasible in medical applications where the level of HPC experience is much lower and the computation software has to undergo a strict certification process. Since our search for a suitable tool was not successful, we decided to develop a brand new workflow management system called k-Dispatch.

k-Dispatch is a Python middleware layer bridging clinical end-users with large computing facilities such as clouds or HPC clusters. k-Dispatch offers a list of generic biomedical ultrasound workflows that execute optimized binaries. The users are able to upload treatment parameters and patient specific data using a simple interface and do not have to consider the execution planning, submission, and monitoring of simulations. Furthermore, k-Dispatch provides data management, accounting, reporting, fault tolerance, and most importantly, optimizes the execution parameters and the amount of computational resources to reduce computational time or cost.

We have successfully deployed k-Dispatch using Docker containers. Currently, the predefined workflows may be submitted using a user-friendly web interface. The workflows are executed by the HPC clusters at the IT4Innovations supercomputing center.

7.1 Future Work

The development of k-Dispatch has reached a point where the system is up and running, however, there are several issues to be solved. First, we would like to implement advanced techniques for performance data mining. Next, we would like to adapt HPC cluster simulators to provide us with reliable evaluation of the workflow execution parameters. We would like to investigate adaptive optimization heuristics for execution parameters and also consider dynamically changing HPC utilization. Finally, we would like to develop a graphical user interface for k-Dispatch administrators and advanced clinical users.

8 ACKNOWLEDGEMENT

This project has received funding from the European Union's Horizon 2020 research and innovation programme H2020 ICT 2016-2017 under grant agreement No 732411 and is an initiative of the Photonics Public Private Partnership. This work was supported by The Ministry of Education, Youth and Sports from the National Programme of Sustainability (NPU II) project IT4Innovations excellence in science - LQ1602" and by the IT4Innovations infrastructure which is supported from the Large Infrastructures for Research, Experimental Development and Innovations project IT4Innovations National Supercomputing Center - LM2015070". This work was supported by the Engineering and Physical Sciences Research Council, United Kingdom, grant numbers EP/L020262/1, EP/M011119/1, EP/P008860/1, and EP/S026371/1.

REFERENCES

[1] Magda A. Abbas, Constatin C. Coussios, and Robin O. Cleveland. 2018. Patient specific simulation of HIFU kidney tumour ablation. *Conference proceedings: IEEE Engineering in Medicine and Biology Society.* 2018, 5709–5712. https://doi.org/10.1109/EMBC.2018.8513647

[2] Gene M. Amdahl. 1967. Validity of the single processor approach to achieving large scale computing capabilities. *Proceedings of the April 1820 1967 spring joint computer conference* 23, 4 (1967), 483–485. https://doi.org/10.1145/1465482.1465560

[3] Peter Amstutz, Michael R. Crusoe, Nebojsa Tijanic, Brad Chapman, John Chilton, Michael Heuer, Andrey Kartashov, Dan Leehr, Herve Menager, Maya Nedeljkovich, and et al. 2016. Common workflow language, v1.0. https://doi.org/10.6084/m9.figshare.3115156.v2

[4] Apache. 2019. Apache Storm. http://storm.apache.org/

[5] Jiten Bhagat, Franck Tanoh, Eric Nzuobontane, Thomas Laurent, Jerzy Orlowski, Marco Roos, Katy Wolstencroft, Sergejs Aleksejevs, Robert Stevens, Steve Pettifer, Rodrigo Lopez, and Carole A. Goble. 2010. BioCatalogue: a universal catalogue of web services for the life sciences. *Nucleic Acids Research* 38, suppl_2 (05 2010), W689-W694. https://doi.org/10.1093/nar/gkq394

[6] Carl Boettiger. 2015. An introduction to Docker for reproducible research. *ACM SIGOPS Operating Systems Review* 49, 1 (jan 2015), 71–79. https://doi.org/10.1145/2723872.2723882

[7] Rajkumar Buyya and Manzur Murshed. 2002. GridSim: a toolkit for the modeling and simulation of distributed resource management and scheduling for Grid computing. *Concurrency Computat.: Pract. Exper.* 14 (2002), 1175–1220. https://doi.org/10.1002/cpe.710

[8] Vandiver Chaplin, Marshal Phipps, and Charles Caskey. 2017. A random phased-array for MR-guided transcranial ultrasound neuromodulation in non-human primates. *Physics in Medicine and Biology* 10 (12 2017), 105016. https://doi.org/10.1088/1361-6560/aabeff

[9] Vojtěch Cima, Stanislav Böhm, Jan Martinovič, Jiří Dvorský, Kateřina Janurová, Tom V. Aa, Thomas J. Ashby, and Vladimir Chupakhin. 2018. HyperLoom: A platform for defining and executing scientific pipelines in distributed environments. In *Proceedings of the 9th Workshop and 7th Workshop on Parallel Programming and RunTime Management Techniques for Manycore Architectures and Design Tools and Architectures for Multicore Embedded Computing Platforms.* ACM, 1–6.

[10] Dask. 2019. Dask natively scales Python. https://dask.org/

[11] Ewa Deelman, Karan Vahi, Gideon Juve, Mats Rynge, Scott Callaghan, Philip J. Maechling, Rajiv Mayani, Weiwei Chen, Rafael Ferreira da Silva, Miron Livny, and Kent Wenger. 2014. Pegasus: a Workflow Management System for Science Automation. *Future Generation Computer Systems* (2014).

[12] Fabric. 2020. Fabric – Pythonic Remote Execution. https://www.fabfile.org/

[13] Rafael Ferreira da Silva, Rosa Filgueira, Ilia Pietri, Ming Jiang, Rizos Sakellariou, and Ewa Deelman. 2017. A characterization of workflow management systems for extreme-scale applications. *Future Generation Computer Systems* 75 (oct 2017), 228–238.

[14] Martin Fjordvald and Clement Nedelcu. 2018. *Nginx HTTP Server - Fourth Edition: Harness the Power of Nginx to Make the Most of Your Infrastructure and Serve Pages Faster Than Ever Before* (4th ed.). Packt Publishing.

[15] Mike Folk, Gerd Heber, Quincey Koziol, Elena Pourmal, and Dana Robinson. 2011. An overview of the HDF5 technology suite and its applications. In *Proceedings of the EDBT/ICDT 2011 Workshop on Array Databases - AD '11.* https://doi.org/10.1145/1966895.1966900

[16] Ian Foster. 2006. Globus toolkit version 4: Software for service-oriented systems. *Journal of Computer Science and Technology* 21, 4 (jul 2006), 513–520. https://doi.org/10.1007/s11390-006-0513-y

[17] Focused Ultrasound Foundation. 2019. *2019 state of the field report.* Technical Report. 1230 Cedars Court, Suite 206.

[18] Lutz Fröhlich. 2018. PostgreSQL 10. In *PostgreSQL 10.* Carl Hanser Verlag GmbH & Co. KG, München, I–X. https://doi.org/10.3139/9783446456419.fm

[19] gLite. 2013. gLite introduction. http://grid-deployment.web.cern.ch/grid-deployment/glite-web/introduction

[20] Anthony Grisey, Sylvain Yon, Véronique Letort, and Pauline Lafitte. 2016. Simulation of high-intensity focused ultrasound lesions in presence of boiling. *Journal of Therapeutic Ultrasound* (2016). https://doi.org/10.1186/S40349-016-0056-9

[21] Derek Groen, Agastya P. Bhati, James Suter, James Hetherington, Stefan J. Zasada, and Peter V. Coveney. 2016. FabSim: Facilitating computational research through automation on large-scale and distributed e-infrastructures. *Computer Physics Communications* 207 (2016), 375–385. https://doi.org/10.1016/j.cpc.2016.05.020 arXiv:1512.02194

[22] John L. Gustafson. 1988. Reevaluating Amdahl's law. *Commun. ACM* 31, 5 (may 1988), 532–533. https://doi.org/10.1145/42411.42415

[23] HTCondor. 2019. HTCondor – High Throughput Computing. https://research.cs.wisc.edu/htcondor/

[24] Sarah E. Jackson and John D. Chester. 2015. Personalised cancer medicine. https://doi.org/10.1002/ijc.28940

[25] Dalibor Klusáček, Simon Toth, and Gabriela Podolnikova. 2017. Complex Job Scheduling Simulations with Alea 4. *CEUR Workshop Proceedings* 1828 (2017), 53–59. https://doi.org/10.1145/1235 arXiv:arXiv:1603.07016v1

[26] Bertram Ludäscher, Ilkay Altintas, Chad Berkley, Dan Higgins, Efrat Jaeger, Matthew Jones, Edward A. Lee, Jing Tao, and Yang Zhao. 2006. Scientific workflow management and the Kepler system. *Concurrency and Computation: Practice and Experience* 18, 10 (aug 2006), 1039–1065. https://doi.org/10.1002/cpe.994

[27] Srirang Manohar and Maura Dantuma. 2019. Current and future trends in photoacoustic breast imaging. *Photoacoustics* 16 (1 12 2019). https://doi.org/10.1016/j.pacs.2019.04.004

[28] Haiyan Meng and Douglas Thain. 2017. Facilitating the reproducibility of scientific workflows with execution environment specifications. *Procedia Computer Science* 108 (2017), 705–714. https://doi.org/10.1016/j.procs.2017.05.116 International Conference on Computational Science, ICCS 2017, 12-14 June 2017, Zurich, Switzerland.

[29] Tom Mens, Alexandre Decan, and Nikolaos Spanoudakis. 2018. A method for testing and validating executable statechart models. *Software and Systems Modeling* (2018). https://doi.org/10.1007/s10270-018-0676-3

[30] Leila Mohammadi, Hamid Behnam, Jahan Tavakkoli, and Mohammad R.N. Avanaki. 2019. Skull's photoacoustic attenuation and dispersion modeling with deterministic ray-tracing: Towards real-time aberration correction. *Sensors (Switzerland)* (2019). https://doi.org/10.3390/s19020345

[31] myExperiment. 2018. myExperiment Home. https://www.myexperiment.org/home

[32] NCSA. 2019. GitHub - ncsa/Scheduler: The aggregate job launcher of single-core or single-node applications on HPC sites. https://github.com/ncsa/Scheduler

[33] Mohammad A. Obaida and Jason Liu. 2017. Simulation of HPC job scheduling and large-scale parallel workloads. In *2017 Winter Simulation Conference (WSC).* IEEE, 920–931. https://doi.org/10.1109/WSC.2017.8247843

[34] Jose A. Pascual, Javier Navaridas, and Jose Miguel-Alonso. 2009. Job Scheduling Strategies for Parallel Processing. In *JSSPP 2009. Lecture Notes in Computer Science, vol. 5798,* Uwe Frachtenberg, Eitan Schwiegelshohn (Ed.). Springer, Berlin, Heidelberg. https://doi.org/10.1007/978-3-642-04633-9_8

[35] Joemini Poudel, Yang Lou, and Mark A. Anastasio. 2019. A survey of computational frameworks for solving the acoustic inverse problem in three-dimensional photoacoustic computed tomography. *Physics in Medicine and Biology* (may 2019). https://doi.org/10.1088/1361-6560/ab2017 arXiv:1905.03881

[36] Antonios Pouliopoulos, Shih-Ying Wu, Mark Burgess, Maria Karakatsani, Hermes Kamimura, and Elisa Konofagou. 2019. A Clinical System for Non-invasive Blood-Brain Barrier Opening Using a Neuronavigation-Guided Single-Element Focused Ultrasound Transducer. *Ultrasound in Medicine and Biology* 46 (10 2019), 73–89. https://doi.org/10.1016/j.ultrasmedbio.2019.09.010

[37] The Pallets Projects. 2020. Jinja. https://palletsprojects.com/p/jinja/

[38] Yves Robert. 2011. *Task graph scheduling.* Springer US, Boston, MA, 2013–2025. https://doi.org/10.1007/978-0-387-09766-4_42

[39] Armin Ronacher. 2013. Flask (A Python Microframework). http://flask.pocoo.org/

[40] Vivek Sarkar. 1989. *Partitioning and Scheduling Parallel Programs for Multiprocessors.* MIT Press, Cambridge. 101–154 pages.

[41] Konstantin Shvachko, Hairong Kuang, Sanjay Radia, and Robert Chansler. 2010. The Hadoop distributed file system. In *Proceedings of the 2010 IEEE 26th Symposium on Mass Storage Systems and Technologies (MSST) (MSST '10).* IEEE Computer Society, Washington, DC, USA, 1–10. https://doi.org/10.1109/MSST.2010.5496972

[42] Priya Singh, Zafruddin Quadri, and Anuj Kumar. 2016. Comparative Study of Parallel Scheduling Algorithm for Parallel Job. *International Journal of Computer Applications* 134, 10 (2016), 10–14.

[43] Visa Suomi, Jiri Jaros, Bradley Treeby, and Robin Cleveland. 2016. Nonlinear 3-D simulation of high-intensity focused ultrasound therapy in the Kidney. IEEE, 5648–5651. https://doi.org/10.1109/EMBC.2016.7592008

[44] Visa Suomi, Bradley Treeby, Jiri Jaros, Pietari Makela, Mikael Anttinen, Jani Saunavaara, Teija Sainio, Aida Kiviniemi, and Roberto Blanco. 2018. Transurethral ultrasound therapy of the prostate in the presence of calcifications: A simulation study. *Medical physics* 45 (9 2018), 4793–4805. https://doi.org/10.1002/mp.13183

[45] Bradley E. Treeby and Ben T. Cox. 2010. k-Wave: MATLAB toolbox for the simulation and reconstruction of photoacoustic wave-fields. *Journal of Biomedical Optics* 15, 2 (Mar-Apr 2010), 021314. https://doi.org/10.1117/1.3360308

[46] Katherine Wolstencroft, Robert Haines, Donal Fellows, Alan Williams, David Withers, Stuart Owen, Stian Soiland-Reyes, Ian Dunlop, Aleksandra Nenadic, Paul Fisher, Jiten Bhagat, Khalid Belhajjame, Finn Bacall, Alex Hardisty, Abraham Nieva de la Hidalga, Maria P. Balcazar Vargas, Shoaib Sufi, and Carole Goble. 2013. The Taverna workflow suite: designing and executing workflows of Web Services on the desktop, web or in the cloud. *Nucleic Acids Research* 41, W1 (5 2013), W557–W561. https://doi.org/10.1093/nar/gkt328

[47] Matei Zaharia, Mosharaf Chowdhury, Michael J. Franklin, Scott Shenker, and Ion Stoica. 2010. *Spark: Cluster computing with working sets.* Technical Report.

Related Paper III

Performance-Cost Optimization of Moldable Scientific Workflows

JAROS Marta and JAROS Jiri

Job Scheduling Strategies for Parallel Processing (JSSPP) 2021
DOI: 10.1007/978-3-030-88224-2_8

Performance-Cost Optimization of Moldable Scientific Workflows

Marta Jaros(✉) and Jiri Jaros

Faculty of Information Technology, Centre of Excellence IT4Innovations,
Brno University of Technology, Brno, Czech Republic
{martajaros,jarosjir}@fit.vutbr.cz

Abstract. Moldable scientific workflows represent a special class of scientific workflows where the tasks are written as distributed programs being able to exploit various amounts of computer resources. However, current cluster job schedulers require the user to specify the amount of resources per task manually. This often leads to suboptimal execution time and related cost of the whole workflow execution since many users have only limited experience and knowledge of the parallel efficiency and scaling. This paper proposes several mechanisms to automatically optimize the execution parameters of moldable workflows using genetic algorithms. The paper introduces a local optimization of workflow tasks, a global optimization of the workflow on systems with on-demand resource allocation, and a global optimization for systems with static resource allocation. Several objectives including the workflow makespan, computational cost and the percentage of idling nodes are investigated together with a trade-off parameter putting stress on one objective or another. The paper also discusses the structure and quality of several evolved workflow schedules and the possible reduction in makespan or cost. Finally, the computational requirements of evolutionary process together with the recommended genetic algorithm settings are investigated. The most complex workflows may be evolved in less than two minutes using the global optimization while in only 14 s using the local optimization.

Keywords: Task graph scheduling · Workflow · Genetic algorithm · Moldable tasks · Makespan estimation

1 Introduction

All fields of science and engineering use computers to reach new findings, while the most compute power demanding problems require High Performance Computing (HPC) or Cloud systems to give answers to their questions. The problems being solved nowadays are often very complex and comprise of a lot of various tasks describing different aspects of the investigated problem and their mutual dependencies. These tasks compose a scientific processing workflow [3]. There are immense of such scientific workflows in various fields [22], yet they have one

D. Klusáček et al. (Eds.): JSSPP 2021, LNCS 12985, pp. 149–167, 2021.
https://doi.org/10.1007/978-3-030-88224-2_8

thing in common. They all demand to be computed in the minimum possible time, and more often, for the lowest possible cost.

The execution of a scientific workflow on an HPC system is performed via communication with the HPC front-end, also referred to as job scheduler [13]. After the workflow data has been uploaded to the cluster, the workflow tasks are submitted to the computational queues to wait until the system has enough free resources, and all task dependencies have been resolved (predecessor tasks have been finished).

Modern HPC schedulers control multiple processing queues and implement various techniques for efficient task allocation and resource management [15]. However, the workflow queuing time, computation time and related cost are strongly dependent on the execution parameters of particular tasks provided by the user during submission. These parameters usually include temporal parameters such as requested allocation length, as well as spatial parameters including the number and type of compute nodes, the number of processes and threads, the amount of memory and storage space, and more frequently, the frequency and power cup of various hardware components. These parameters, unfortunately, have to be specified by the end users based on their previous experience with the task implementation and knowledge on the input data nature.

In everyday practice, the estimations of task allocation lengths are quite inaccurate, which disturbs the scheduling process. Most users deliberately overestimate the computational time in order to provide some reserve to mitigate performance fluctuation and prevent premature termination of the task execution [23]. Moreover, many complex tasks are written as moldable distributed parallel programs being able to exploit various amounts and types of computing resources. Nonetheless, it is again the user responsibility to choose appropriate values of these parameters according to the input data.

The task moldability is often limited by many factors, the most important of which being the domain decomposition [6], parallel efficiency [2], and scalability [14]. While the domain decomposition may limit the numbers of processing units (nodes, processes, threads) to rather a sparse list of acceptable values, the parallel efficiency determines the execution time and cost for a given task and a chosen amount of resources. Naturally, the lower the parallel efficiency, the lower the speed-up, and consequently, the longer the computation time and the higher the computational cost. Finally, the scalability upper-bounds the amount of exploitable resources by the overall available memory.

While the field of rigid workflow optimization, where the amount of resources per task cannot be tuned, has been thoroughly studied and is part of common job schedulers such as PBSPro [13] or Slurm [30], the automatic optimization and scheduling of moldable workflows has still been an outstanding problem, although firstly solved two decades ago in [10].

For the last decade, many papers have focused on the estimation of rigid workflow execution time in HPC systems and enhancing the resource management. For example, Chirkin et al. [7] introduces a makespan estimation algorithm that may be integrated into schedulers. Robert et al. [25] gives an overview of

task graph scheduling algorithms. The usage of genetic algorithms addressing the task scheduling problems has also been introduced, e.g., a task graph scheduling on homogeneous processors using genetic algorithm and local search strategies [17] and a performance improvement of the used genetic algorithm [24]. However, handful works have taken into the consideration the moldability and scaling behaviour of particular tasks, their dependencies and the current cluster utilization [4,9,29].

This paper focuses on the automation optimization of the moldable scientific workflow execution using genetic algorithms [28]. The optimization of execution parameters is based on collected historical performance data (i.e., strong scaling) for supported tasks in the workflow. The paper presents several objective functions and trade-off coefficients that allow to customize the pressure either on the overall execution time, or the computational cost, or both.

The rest of the paper is structured as follows. Section 2 describes the optimization algorithm, the solution encoding specifying the amount of resources per task, the objective and fitness functions evaluating the quality of the candidate workflow schedule and the details of the applications use cases. Section 3 elaborates on the quality of the genetic algorithm and its best set-up, presents the time complexity of the search process and compares several workflow execution schedules by the optic of particular objective functions. The last section concludes the paper and draws potential future improvements of this technique.

2 Proposed Algorithm

The assignment of optimal amount of compute resources to particular tasks along with the scheduling of the workflow as a whole is known to be an NP-hard problem [9]. There have been several attempts to use heuristics to solve this problem [4,16,18,26], however, they are either tightly connected to an existing HPC cluster and its scheduler, use idealized models of strong scaling and parallel efficiency, or optimize only one criterion such as makespan, cluster throughput, or execution cost. The user tunability of these approaches are thus limited.

Therefore, we decided to use genetic algorithms, which are highly flexible in combinatorial optimization and scheduling [8]. From the vast number of existing implementations, PyGAD [11], an open-source Python library for building the genetic algorithm, was chosen. PyGAD supports various types of genetic operators and selection strategies, and offers a simple interface for objective function definition.

The overall concept of the moldable workflow scheduling optimization using PyGAD is shown in Fig. 1. The structure of the task graph is converted into a 1D array where each element corresponds to a single task and holds its execution parameters. The genetic algorithm traverses the search space and seeks for good solutions by applying genetic manipulations and selection strategies on the population of candidate solutions. The quality of these candidate solutions is evaluated by the fitness function. Although the paper presents three different methods to evaluate the schedule quality, the concept is similar in all cases. First,

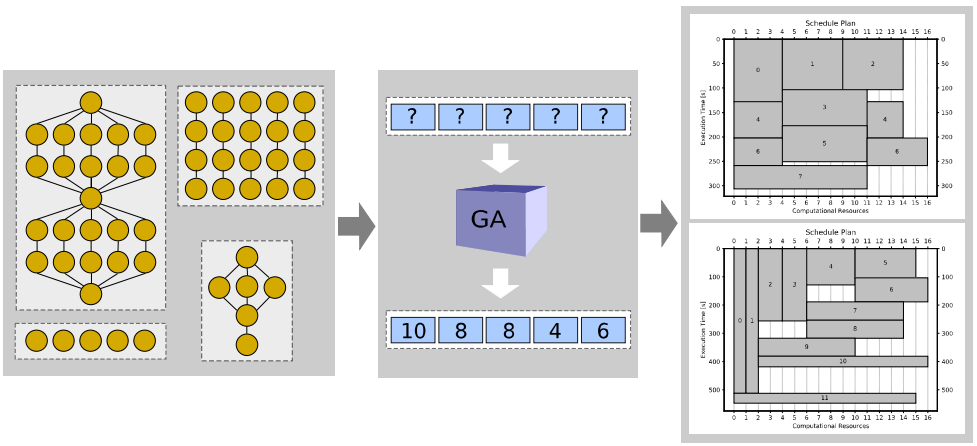


Fig. 1. A workflow is transformed to a vector of integer elements specifying assigned amount of resources to particular tasks. This vector represents a candidate solution of the GA search space. The final output of the optimization can be visualized as a workflow schedule.

the execution time for every task is calculated based on the task type, execution parameters set by the GA, input data size, and known parallel efficiency/strong scaling behavior. Next, the tasks are submitted to the cluster simulator that draws up an execution schedule and calculates the makespan (the critical path through the workflow including queuing times) and execution cost. The output of the optimization is a set of best execution parameters for individual tasks minimizing given criteria implemented by the fitness function.

2.1 Solution Encoding

In order to optimize workflow execution schedules using GA, it is necessary to transform the workflow into a template for candidate solutions (chromosomes) I. The workflow's DAG is traversed in a breath-first manner producing a vector of N tasks (genes). Every gene i corresponds to a single task and holds the execution parameters (resources) R_i assigned to the task i, see Eq. (1).

$$I = (R_1, R_2, \ldots, R_N) \tag{1}$$

The execution parameters being investigated in this study only consider the number of computing nodes assigned to a given task. This set can be simply extended in the future to support, e.g., node cpu frequency and power cup or the number of processes/threads per node.

The number of nodes assignable to a given task is naturally constrained by 1 from the bottom, and by the size of the computing system from the top. Moreover, it is also limited by the type of the task, its scalability, and the size of input data. The strong scaling, parallel efficiency and scalability were measured for each task type and input data size in advance using short benchmark runs

and stored in the performance database. These constraints are imposed at the beginning of the fitness function evaluation.

2.2 Fitness Function

This paper considers three different types of the fitness function looking at the optimization problem from different angles: (1) Independent local optimization of the execution time for particular tasks useful when running small tasks on large HPC systems, (2) Global optimization of the whole workflow minimizing the execution time and related computational cost under on-demand resource allocations, (3) Global optimization of the whole workflow time and cost on statically allocated cluster parts, i.e., the idling nodes also contribute to the computational cost.

Local Optimization of Workflow Tasks. This fitness functions optimizes each task independently considering only the execution time while neglecting the computational cost, see Eq. (2). This fitness function does not use the cluster simulator but only sums the execution time of all tasks. Let us note that the highest possible number of computing nodes may not lead to the fastest execution time due to unbalanced local decomposition, high overhead of parallel computation, etc.

This fitness function relies on the cluster scheduler to assemble a good execution schedule of the whole workflow when provided optimal setup for particular tasks. This statement is likely to be valid for large HPC clusters with hundreds of nodes and tasks employing low tens of nodes. From the scheduling point of view, this fitness function is the fastest one.

$$fitness = t = \sum_{i=1}^{N} t_i(R_i) \tag{2}$$

where t is the aggregated net execution time of N tasks in the workflow, each of which running on R_i nodes for time t_i.

Global Optimization with On-Demand Allocation. This fitness function minimizes the overall execution time t of the workflow given by the sum of the execution time of the tasks along the critical path in the workflow graph (makespan [12]), together with the computational cost c given by a sum of computational cost of all tasks in the workflow, see Eq. (5).

As we know from the problem definition, those two requirements usually go against each other. Therefore, an α parameter to prioritize either makespan or cost is introduced. In order to balance between proportionally very different criteria, a kind of normalization is introduced. The makespan is normalized by the maximum total execution time of the workflow t_{max}, which is considered to be the sum of the execution times of all N tasks executed by only a single computation node in a sequential manner. The cost is normalized by the minimum

execution cost which is the cost of the workflow computed by a single node in a sequential manner, see Eq. (3). This presumption is valid for typical parallel algorithms with sub-linear scaling, i.e., parallel efficiency as a function of the number of nodes is always smaller than 1, $E(P) < 1$.

$$c_{min} = t_{max} = \sum_{i=1}^{N} t_i(1) \tag{3}$$

$$c_i = t_i(R_i) \cdot R_i \tag{4}$$

$$fitness = \alpha \cdot \sum_{j \in M} \left(\frac{t_j(R_j)}{t_{max}}\right) + (1 - \alpha) \cdot \sum_{i=1}^{N} \left(\frac{c_i(R_i)}{c_{min}}\right), \quad \text{where } M = \{i | i \in \text{CriticalPath}\} \tag{5}$$

This fitness function suits best the workflow being executed in environments with shared resources where only truly consumed resources are paid for, e.g., shared HPC systems.

Global Optimization with Static Allocation. The last fitness function described by Eq. (8) also minimizes the workflow makespan, but the computational cost now takes into the consideration also idling nodes. Let us imagine we have a dedicated portion of the cluster consisting of 64 nodes statically allocated before the workflow has started. The computational cost, the user will be accounted for, equals to the size of the allocation multiplied by the makespan, no matter some nodes are not being used for the whole duration of the workflow execution. The fitness function thus attempts to shake down the tasks to minimize the amount of idling resources while still minimizing the makespan. The execution cost is then normalized by the highest possible cost in the dedicated system where only one node works.

$$c_{max} = t_{max} \cdot P \tag{6}$$

$$c = \sum_{i=1}^{N} t_i(R_i) \cdot R_i \tag{7}$$

$$fitness = \alpha \sum_{j \in M} \left(\frac{t_j(R_j)}{t_{max}}\right) + (1 - \alpha) \frac{c_{max} - c}{c_{max}}, \quad \text{where } M = \{i | i \in \text{CriticalPath}\} \tag{8}$$

Similarly to the previous case, t is the overall execution time of the workflow, and t_{max} is the maximum overall execution time obtained for a serial scheduling of sequential tasks. The number of nodes statically allocated to the workflow is denoted by P. The number of nodes assigned per tasks i is R_i. The maximum possible cost is represented by c_{max} while the actual cost based on the current execution parameters and the workflow structure is denoted by c.

2.3 Cluster Simulator

In order to create a workflow execution schedule and calculate the makespan, we developed a simple cluster simulator called *Tetrisator*. The name of this component is inspired by the Tetris game [5] since there is a strong analogy in arranging the blocks of different sizes and shapes with the optimization of the execution schedule to minimize execution time and cost. The blocks can be seen as tasks and their sizes are given by required amount of resources and corresponding execution time. The blocks a.k.a tasks may be molded to be "wider" or "longer" by changing the number of resources, however, their surface does not have to stay constant due to varying parallel efficiency.

Tetrisator simulates the operation of an artificial HPC system with a predefined number of computing nodes P. The tasks are submitted to the simulator in the same order as defined in the chromosome (a breadth-first top down traversal). During the submission, the numbers of nodes assigned to particular tasks are taken from the chromosome and the corresponding execution times are located in the performance database. The breadth-first traversal also allows a simple definition of task dependencies the simulator has to obey. If there are multiple tasks being ready to be executed, the submission order is followed. This is inspired the default behaviour of the PBS job scheduler with the backfilling policy switched off [27].

3 Experimental Results

The experiments presented in this paper have the following goals: (1) confirm the hypothesis that it is possible to find suitable schedules for given workflows using genetic algorithms, (2) investigate the suitability of the α parameter to prefer of one optimization criterion over the other one (overall execution time vs. computational cost), and (3) evaluate the computational requirements of the optimization process on various workflow sizes.

3.1 Investigated Moldable Workflows

The performance and search capabilities of the proposed optimization algorithm were investigated on three scientific workflows inspired by real-world applications of the acoustic toolbox k-Wave [19] for validation of neurostimulation procedures, see Fig. 2. The workflows are composed of two different kinds of tasks, simulation tasks (ST) and data processing tasks (PT). The first workflow shows a barrier behaviour where all simulation tasks at the first level have to finish before the data is processed by a single data processing task. Only after that, the second level of STs can continue. The second workflow uses a reduction tree where the data processing is parallelized in order to reduce the execution time of PT tasks. The last workflow, not shown in the figure, is composed of the set of independent STs executed in embarrassingly parallel manner.

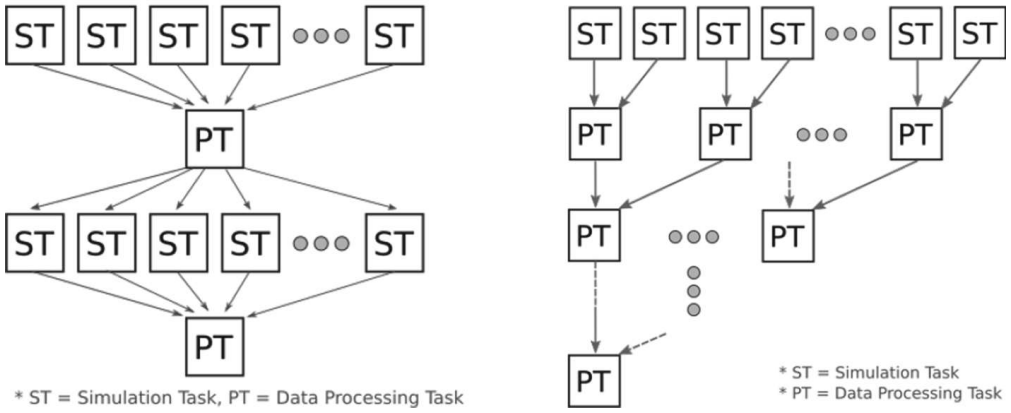


Fig. 2. The structure of two investigated workflows. The simulation tasks are interleaved with data processing tasks implying barriers between stages (left), the data produced by the simulation tasks are merged via a reduction tree (right).

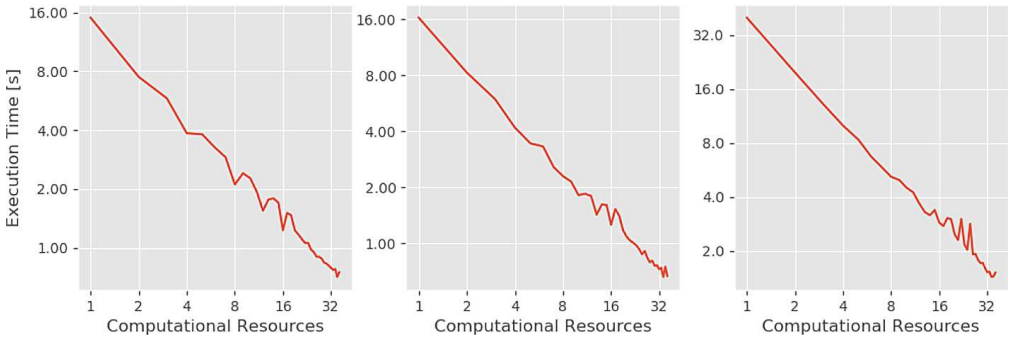


Fig. 3. Strong scaling of the k-Wave toolbox measured for $<1, 36>$ nodes on domain sizes composed of 500^3, 512^3 and 544^3 grid points. k-Wave simulations are the main part of simulations tasks in the examined workflows.

The simulation tasks are heavy computing programs scalable from 1 to 36 nodes, see Fig. 3. Their scaling was measured using the C++/MPI implementation of the k-Wave toolbox on the Barbora supercomputer at IT4Innovations[1]. The scaling behaviour depends on the input data size and shows several local optima for the number of nodes being powers of two. The shortest execution time was seen for 35 nodes. From these three examples, the first one was chosen in our experiments. The data processing tasks are lightweight tasks executable on one or two nodes. Their time complexity grows linearly with the number of input files they have to process.

In real k-Wave applications, the size of the domain for all STs is the same, however, the amount of time steps may vary by up to 25%. This is given by the mutual position of the transducer and the patient's head, which influences

[1] https://docs.it4i.cz/barbora/introduction/.

the distance the ultrasound wave has to travel. Moreover, the performance of all processors in the cluster is not equal. According to [1], the fluctuations may cause up to 5% deviations in the execution time. Both factors are considered by adding random perturbations to the task execution time during the workflow generation.

3.2 Local Task Optimization of the Execution Time

First, we investigated the local optimization of the proposed workflows, which is the simplest kind of optimization. This optimization only considers the net execution time of all tasks neglecting the queueing times and simulation cost. Thus no α parameter is used. This technique shows very good capabilities in optimizing particular tasks. From 20 independent runs of the GA, more than 90% of trials always found the best possible solution, the fitness of which can be analytically derived.

Table 1 shows suitable parameters for the genetic algorithm along with the number of generations necessary to find the optimal schedule, the execution time in seconds and the success rate. Since the variability of the results across different workflows was negligible, we collapsed all results into a single table.

The table reveals that the necessary population size linearly grows with the size of the workflow from 25 up to 150 individuals, but still stays quite small. This is natural behaviour since bigger workflows require longer chromosomes which in turn requires larger populations to keep promising building blocks of the solution. The best selection strategy driving the GA through the search spaces appears to be Steady state selection, although the difference to the Rank and Tournament selections was marginal. The number of generations to be evaluated before the GA finds the optimal schedule stays relatively constant close to 200. On the other hand, the execution time appears to grow quadratically. This growth can be attributed to a product of increasing population size which rises the number of fitness function evaluations, and the linearly growing time complexity of the fitness function evaluation. Nevertheless, an execution time of 14s with 95% of success rate for the biggest workflow is an excellent result.

3.3 Global Workflow Optimization of Execution Time and Cost

The global optimization of the workflow considers both criteria and balances between them using the α parameter. In this section we investigate two fitness functions oriented on on-demand and statically allocated resources.

The Influence of the α Parameter. Let us first investigate the influence of the α parameter on both global fitness functions. In practise, the α parameter can be seen as a user-friendly control slider promoting either the execution time or cost. The following values of α were tested: 0.95 and 0.8 prioritizing the minimal makespan, 0.5 balancing the makespan and the cost/usage of resources), and 0.2 and 0.05 prioritizing the minimal execution cost and unused resources, respectively.

Table 1. Computation requirements of the local optimization method together with recommended genetic algorithm (GA) settings that lead to optimal schedules obtained in the shortest time. Other GA settings common for all experiments is uniform crossover of 0.7 probability, random mutation of 0.01 probability, and 5% elitism.

Workflow size	Population size	Selection method	Median number of generations	Average runtime	Success rate
7	25	Steady State, Rank	180	0.27 s	100%
8	25	Steady State, Rank	220–250	0.37–0.42 s	100%
15	50	Steady State, Rank, Tournament, Roulette Wheel	120–200	0.52-0.87s	100%
16	50	Steady State, Rank, Tournament	180–200	0.98–1.08 s	100%
31	100	Steady State, Rank	100	1.40 s	100%
32	100	Steady State, Rank	190	3.41 s	90–95%
63	100	Rank, Steady State	215	5.61 s	100%
64	150	Steady State, Rank	260	13.29 s	90–95%

For each value of α and suitable GA settings, 20 independent runs were carried out. For the sake of brevity, only a few examples of selected workflows with the best GA settings will be shown. For each example, the results from all runs were collected, sorted, and 5% of the best solutions visualized in the form of a Pareto frontier. The color of the data points and lines representing the frontiers correspond to the α parameter used.

Let us start with the quality of solutions produced by the on-demand allocation fitness, see Fig. 4 and 5. Although evolved solutions for various α parameter may slightly overlap, Fig. 4 shows that we managed to drive the genetic algorithm to find desired solutions (forming clusters) that meet given optimization constraints. By adjusting the α parameter, we move along an imaginary curve composed by the combination of all Pareto frontiers. Thus when the importance is attached to the simulation cost, it is possible to get a schedule that reduces the cost by 10%, however, runs for 12% longer time, and vice versa. It can also be seen that each value of α works well only in a relatively short interval (the

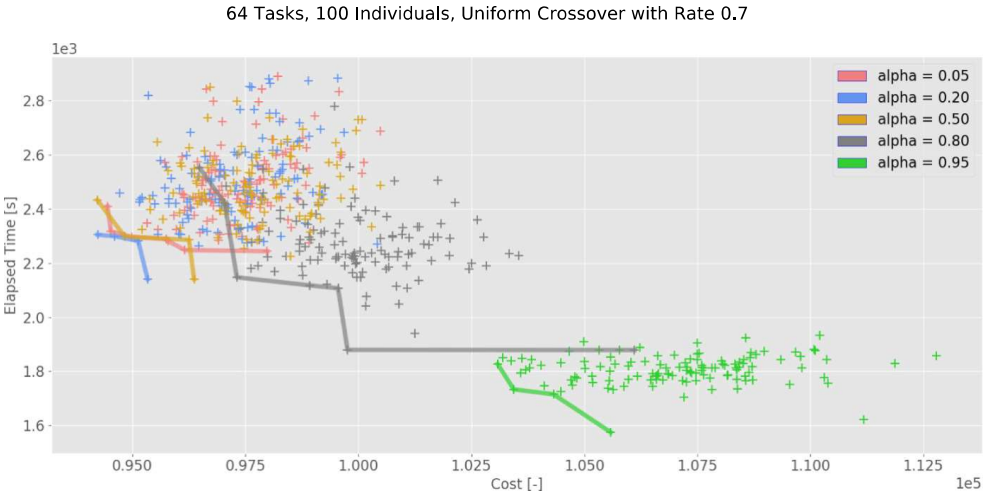


Fig. 4. Pareto frontier and dominated solutions calculated using the fitness function for on-demand allocations for the workflow with two levels of simulation tasks and various values of the α parameter.

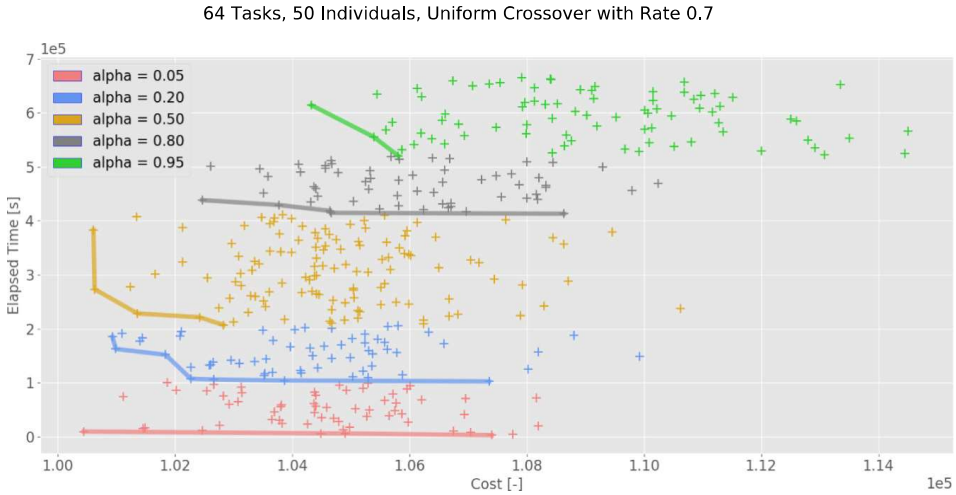


Fig. 5. Pareto frontier and dominated solutions calculated using the fitness function for on-demand allocations for the workflows without dependencies and various values of the α parameter.

middle of the frontier). At the edges it is usually outperformed by a different values of α.

The workflows without dependencies, however, show much worse parametrization, see Fig. 5. The only sensible value of α seems to be 0.05. Other values produce much worse compromises between time and cost. The only exception is the value 0.95 which can offer 15%–20% cost-effective schedules, but many times slower.

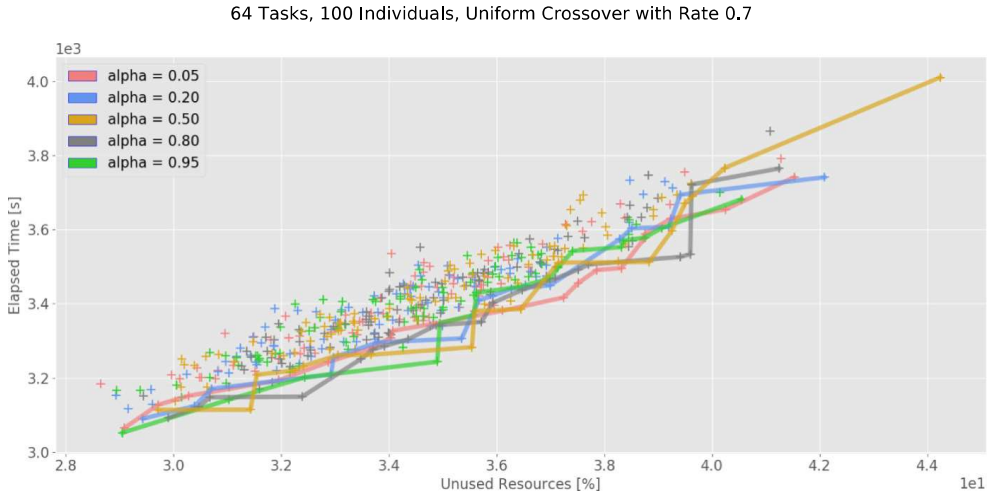


Fig. 6. Pareto frontier and dominated solutions calculated using the fitness function for static allocations for the workflow with two levels of simulation tasks and various values of α balancing between makespan and percentage of unused resources.

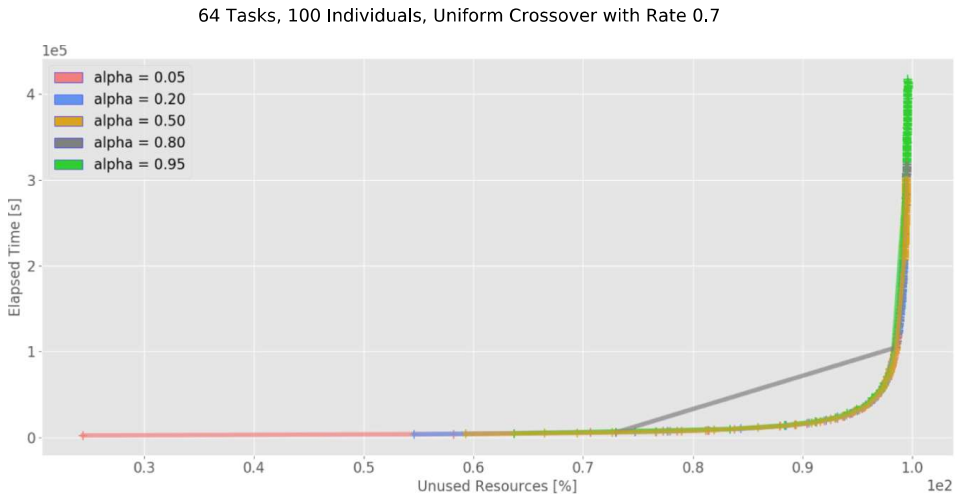


Fig. 7. Pareto frontier and dominated solutions calculated using the fitness function for static allocations for the workflow without dependencies and various values of α balancing between makespan and percentage of unused resources.

For experiments using the fitness function for static cluster allocations, we only show three different α parameters because the solutions highly overlap, see Figs. 6 and 7. The solutions found for workflows containing task dependencies seem to be saturated by the same minimal execution time. Smaller α parameter pushes the genetic algorithm to find solutions with smaller amount of unused resources (up to 44%) but a range of found solutions is quite high. From this point of view, 0.05 for α gives the most reasonable solutions. This is even more

Table 2. Recommended settings for the GA and the on-demand allocation fitness function. Other settings common for all experiments are uniform crossover of 0.7 probability, random mutation and 5% elitism.

Alpha	Workflow size	Population size	Selection method	Mutation probability	Median number of evaluations
0.95	7–16	25	Rank	0.001	2000–19750
	31–64	50	Steady State	0.01	5000–10000
0.80	7–16	25	Steady State	0.01	1250–2500
	31–64	50, 100	Rank	0.001	22000 (50)–45000 (50)
0.50	7–64	50, 100	Rank	0.001	2500 (50)–65000 (100)
0.20	7–64	25, 50	Rank	0.001	8750 (25)–42500 (50)
0.05	7–64	50, 100	Steady State	0.01	5000 (50)–30000 (100)

visible for workflows without task dependencies where 0.05 for the α parameter optimizes both makespan and the amount of unused resources.

The experiments showed that both criteria, the makespan and percentage of idle resources, are highly correlated. Thus, the lower percentage of idle resources the faster execution time. Although this may sound natural, the anomalies in the scaling behaviour of particular tasks has the potential to break this presumption. This experiment, however, shows that the scaling plots in Fig. 3 are very close to the perfect scaling.

Suitable Parameters of the Genetic Algorithm. Table 2 presents recommended settings for the genetic algorithm which produced best results along with the computational requirements expressed as the number of fitness function evaluations (i.e., a product of the number of generations and the population size). When two population sizes are given, the smaller one is used for the smaller workflows, and vice versa. The median number of evaluation is calculated for the actual population size shown in bracket in the last column. The range is bounded by two values, the one for the smallest workflow in the range and the one for the biggest workflow.

Table 3 presents an average execution time for a single generation. In connection with Table 2, the absolute wall clock time of the evolution process can be calculated. As an example, a schedule for a workflow with 64 dependant tasks can be evolved in 2 min and 20 s. We found out that schedules for tasks without dependencies may be evolved in 2 to 3 times shorter time.

The recommended settings of the genetic algorithm covers 0.7 probability of uniform crossover, steady state selection, 1% random mutation and 5% elitism. Workflow with less than 31 tasks could be evolved with 25 individuals in the population whereas bigger workflows (up to 64 tasks) with 50 individuals. It took approximately from 2500 (100 generations for 25 individuals) to 25000 (500 generations for 50 individuals) evaluations to evolve schedules for workflows of 7 to 64 tasks. So, a schedule for the workflow of 64 tasks with dependencies is evolved in a minute.

Table 3. The execution time of the evolution process for various workflows with dependencies and population sizes measured using global fitness functions on the Salomon cluster at IT4Innovations. The evolution runtimes for workflow without dependencies are approximately three times smaller.

Population size	Runtime per a single generation in seconds							
Workflow size	7	8	15	16	31	32	63	64
25	0.004	0.005	0.010	0.010				
50	0.007	0.009	0.019	0.021	0.040	0.043	0.112	0.120
100	0.013	0.018	0.037	0.043	0.077	0.088	0.227	0.220
150					0.110	0.132	0.382	0.335

Investigation of the Workflow Schedules. Here, we show and compare several evolved schedules using different fitness functions. For better visibility, only schedules for workflows of 15 and 16 tasks are shown. Figure 8 shows two execution plans for 15 and 16 tasks, respectively, locally optimized by Eq. (2). Regardless of the number of tasks in the workflow, the genetic algorithm always picks 35 nodes for simulation tasks and 2 nodes for data processing tasks because this selection assures their minimal execution time.

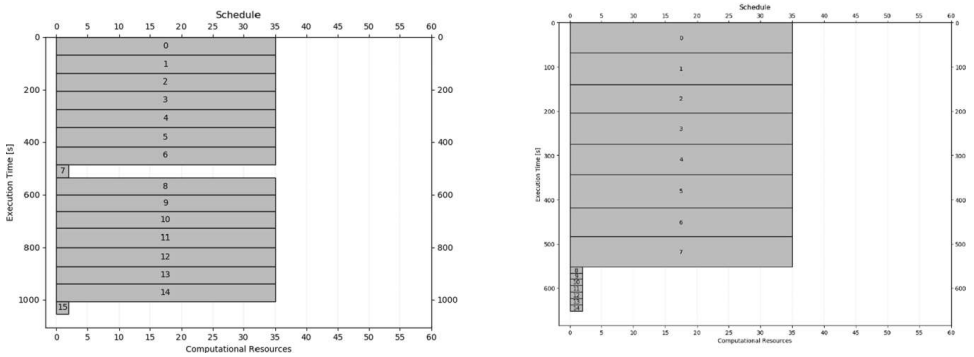


Fig. 8. Evolved schedules for investigated workflows with 16 and 15 dependant tasks using a local optimization.

Figure 9 shows evolved schedules using a global optimization for on-demand allocations balancing the makespan and computational cost with $\alpha = 0.5$. When compared with schedules depicted in Fig. 8, it is evident that the GA preferred much smaller amounts of nodes for simulation tasks which resulted in a cost reduction by 40% and makespan increase by 37% for the workflow of 16 tasks. In the case of the workflow with 15 tasks, we may however observe that the makespan is even better when the global optimization is used. This is given by the way the local optimization works, i.e., the concurrency is not expected. Here,

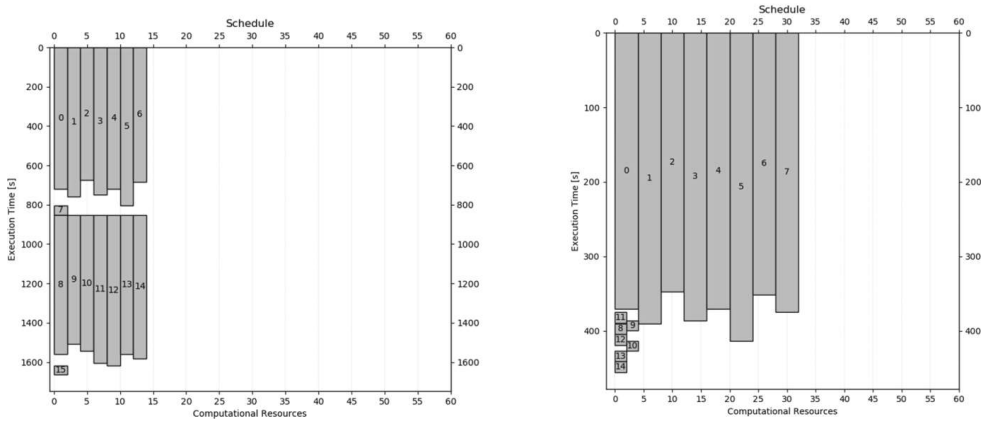


Fig. 9. Evolved schedules for investigated workflows with 16 and 15 dependant tasks using a global optimization for on-demand allocations balancing the makespan and computational cost.

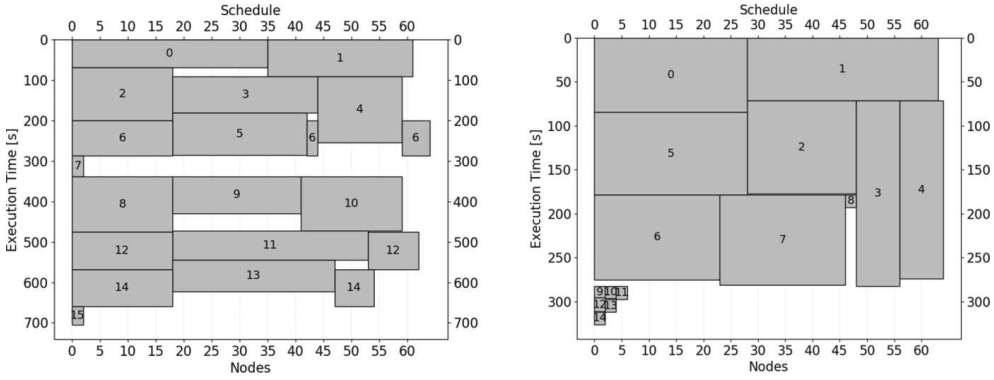


Fig. 10. Evolved schedules for workflows with 16 and 15 dependant tasks using a global optimization balancing the makespan and the amount of unused resources (24.49% for the left workflow, 15.21% for the right workflow).

the global optimization gives better results in both aspects, i.e., the makespan was reduced by 30% and the computational cost by 37%.

Figure 10 shows evolved schedules using a global optimization balancing the makespan and the amount of unused resources with $\alpha = 0.5$. If we compare them with schedules in Fig. 8, we can see that in both cases the makespan is reduced by 51% in case of 15 tasks, and by 35% in case of 16 tasks while the amount of unused resources was reduced from 53.0% and 50.0% to 15.21% and 24.49%, respectively.

Since the global optimization approaches are not comparable, we just emphasize the differences between obtained solutions in Fig. 9 and Fig. 10. It can be seen that the solutions evolved using the global optimization focusing on the amount of unused resources have shorter makespans because there is an effort

to use, i.e., pay for, the resources for the shortest time. If we evaluate these solutions using the fitness function for on-demand allocations, it is obvious that we get more expensive solutions than those which were originally evolved using this fitness function. Since we cannot compare the solutions in absolute numbers, we can come out of the premise that computational cost equals to actually used resources. In other words, the solution for 15 tasks found by fitness function for static allocations used 85% of available resources but the solution found by fitness for on-demand allocations used only 42%. The same states for 16 tasks where the solution found by the fitness function for static allocations used 75% of available resources but the solution found by the fitness function for on-demand allocations used only 19%.

The makespans of 15 tasks schedules differ by 29% while there is a 31% difference in obtained computational cost and the amount of unused resources. The schedules of 16 tasks differ by 59% in their makespans and by 14% in the computational cost.

4 Conclusions

This paper investigates the execution optimization of moldable scientific workflows. It uses genetic algorithm to evolve schedules for workflows comprising of two kinds of tasks with and without mutual dependencies. The presented objective functions use collected historical performance data for supported workflow's tasks. Those objective functions implement a trade-off coefficients that allow the schedule customization to either minimize one objective to another or to balance them. The paper introduces three objective functions that provide the (1) local optimization of workflow tasks minimizing their execution times, (2) global optimization with on-demand resource allocation balancing the workflow makespan and its computational cost, and (3) global optimization with static resource allocation balancing the workflow makespan and the cluster's idling nodes.

After performing the experiments, we confirmed our hypothesis that (1) we are able to generate good schedules for various workflows as well as meet different optimization criteria. For local optimization, we got very good results where more than 90% of performed trials found the optimal solution. (2) When performing a multi objective optimization, we introduced an α trade-off parameter and confirmed we can prioritize one objective to another. Here, we got the best results for the global optimization with on-demand resources and workflows with task dependencies where solutions found for the different parameter α form clusters. Let us note, that the trade-off parameter allows to customize the solution parameters only in a limited scale of makespan and cost, e.g., 10%. Much worse parametrization could be seen for workflows without task dependencies for both global optimization methods where the only value 0.05 of α produced sensible solutions. (3) We measured and summarized computational demands of each presented objective function and workflows of different sizes. Using the local optimization, the workflow of 64 tasks could be evolved in 14 s.

Global optimization is more computationally demanding but we managed to get the schedule for the most complex workflow with task dependencies in roughly 2 min. Finally, the paper provides the genetic algorithm settings to reproduce the presented results.

4.1 Future Work

Here we summarize several ideas to be addressed soon. First, we would like to better tune the presented trade-off coefficient α and better define ourselves among other already existing optimization heuristics. Next, we would like to validate our approach against standard task graphs[2] of different sizes.

Furthermore, a couple of the algorithm improvements is to be addressed. Currently used cluster simulator traverses tasks within a workflow in a breadth-first top down order and follows the task submission order when multiple tasks are ready to be executed. A mechanism such as backfilling commonly presented in the PBS job scheduler is not implemented. In practise, we use mainly PBS-based clusters for our workflows, thus, we would like to integrate this functionality to the presented Tetrisator. We will also consider a possibility to integrate an already existing cluster simulator, e.g., ALEA [21].

Next, more real world tasks together with their measured performance data would be incorporated. In reality, we usually cannot measure and hold performance data for all input data sizes and input parameters options. Therefore, we need to implement interpolation based heuristics [20].

Acknowledgments. This project has received funding from the European Union's Horizon 2020 research and innovation programme H2020 ICT 2016–2017 under grant agreement No 732411 and is an initiative of the Photonics Public Private Partnership. This work was supported by the Ministry of Education, Youth and Sports of the Czech Republic through the e-INFRA CZ (ID: 90140). This work was supported by Brno University of Technology under project number FIT/FSI-J-21-7435.

References

1. The shift from processor power consumption to performance variations: fundamental implications at scale. Comput. Sci. - Res. Dev. **31**(4), 197–205 (2016)
2. Amdahl, G.M.: Validity of the single processor approach to achieving large scale computing capabilities. In: Proceedings of the April 1820 1967 Spring Joint Computer Conference, vol. 23, no. 4, pp. 483–485 (1967)
3. Bharathi, S., Chervenak, A., Deelman, E., Mehta, G., Su, M.-H., Vahi, K.: Characterization of scientific workflows. In: 2008 Third Workshop on Workflows in Support of Large-Scale Science, pp. 1–10. IEEE, November 2008
4. Bleuse, R., Hunold, S., Kedad-Sidhoum, S., Monna, F., Mounie, G., Trystram, D.: Scheduling independent moldable tasks on multi-cores with GPUs. IEEE Trans. Parallel Distrib. Syst. **28**(9), 2689–2702 (2017)

[2] http://www.kasahara.cs.waseda.ac.jp/schedule/.

5. Breukelaar, R., Demaine, E.D., Hohenberger, S., Hoogeboom, H.J., Kosters, W.A., Liben-Nowell, D.: Tetris is hard, even to approximate. Int. J. Comput. Geomet. Appl. **14**, 41–68 (2004)
6. Chan, T.F., Mathew, T.P.: Domain decomposition algorithms. Acta Numer. **3**, 61–143 (1994)
7. Chirkin, A.M., et al.: Execution time estimation for workflow scheduling. Future Gener. Comput. Syst. **75**, 376–387 (2017)
8. Cotta, C., Fernández, A.J.: Evolutionary Scheduling. Studies in Computational Intelligence, vol. 49. Springer, Heidelberg (2007). https://doi.org/10.1007/978-3-540-48584-1
9. Dutot, P.-F., Netto, M.A.S., Goldman, A., Kon, F.: Scheduling moldable BSP tasks. In: Feitelson, D., Frachtenberg, E., Rudolph, L., Schwiegelshohn, U. (eds.) JSSPP 2005. LNCS (LNAI and LNB), vol. 3834, pp. 157–172. Springer, Heidelberg (2005). https://doi.org/10.1007/11605300_8
10. Feitelson, D.G., Rudolph, L.: Toward convergence in job schedulers for parallel supercomputers. In: Feitelson, D.G., Rudolph, L. (eds.) JSSPP 1996. LNCS (LNAI and LNB), vol. 1162, pp. 1–26. Springer, Heidelberg (1996). https://doi.org/10.1007/BFb0022284
11. Gad, A.F.: Geneticalgorithmpython: Building genetic algorithm in python (2021). https://github.com/ahmedfgad/GeneticAlgorithmPython/tree/05a069abf43146e7f8eb37f37c539523bf62ac9a
12. Hejazi, S.R., Saghafian, S.: Flowshop-scheduling problems with makespan criterion: a review. Int. J. Prod. Res. **43**(14), 2895–2929 (2005)
13. Henderson, R.L.: Job scheduling under the portable batch system. In: Feitelson, D.G., Rudolph, L. (eds.) JSSPP 1995. LNCS (LNAI and LNB), vol. 949, pp. 279–294. Springer, Heidelberg (1995). https://doi.org/10.1007/3-540-60153-8_34
14. Hill, M.D.: What is scalability? ACM SIGARCH Comput. Archit. News **18**(4), 18–21 (1990)
15. Hovestadt, M., Kao, O., Keller, A., Streit, A.: Scheduling in HPC resource management systems: queuing vs. planning. In: Feitelson, D., Rudolph, L., Schwiegelshohn, U. (eds.) JSSPP 2003. LNCS (LNAI and LNB), vol. 2862, pp. 1–20. Springer, Heidelberg (2003). https://doi.org/10.1007/10968987_1
16. Huang, K.-C., Huang, T.-C., Tsai, M.-J., Chang, H.-Y.: Moldable job scheduling for HPC as a service. In: Park, J., Stojmenovic, I., Choi, M., Xhafa, F. (eds.) Future Information Technology. Lecture Notes in Electrical Engineering, vol. 276, pp. 43–48. Springer, Heidelberg (2014). https://doi.org/10.1007/978-3-642-40861-8_7
17. Izadkhah, H.: Learning based genetic algorithm for task graph scheduling. Appl. Comput. Intell. Soft Comput. **2019**, 15 (2019). Article ID 6543957. https://doi.org/10.1155/2019/6543957
18. Jansen, K., Land, F.: Scheduling monotone moldable jobs in linear time. In: 2018 IEEE International Parallel and Distributed Processing Symposium (IPDPS), pp. 172–181. IEEE, May 2018
19. Jaros, J., Rendell, A.P., Treeby, B.E.: Full-wave nonlinear ultrasound simulation on distributed clusters with applications in high-intensity focused ultrasound. Int. J. High Perform. Comput. Appl. **30**(2), 1094342015581024- (2015)
20. Jaros, M., Sasak, T., Treeby, B.E., Jaros, J.: Estimation of execution parameters for k-wave simulations. In: Kozubek, T., Arbenz, P., Jaroš, J., Říha, L., Šístek, J., Tichý, P. (eds.) HPCSE 2019. LNCS, vol. 12456, pp. 116–134. Springer, Cham (2021). https://doi.org/10.1007/978-3-030-67077-1_7

21. Klusacek, D., Podolnikova, G., Toth, S.: Complex job scheduling simulations with Alea 4. In: Tan, G. (ed.) Proceedings of the 9th EAI International Conference on Simulation Tools and Techniques, Belgium, pp. 124–129. ICST (2016)
22. Lamprecht, A.-L., Turner, K.J.: Scientific workflows. Int. J. Softw. Tools Technol. Transf. **18**(6), 575–580 (2016)
23. Mu'alem, A.W., Feitelson, D.G.: Utilization, predictability, workloads, and user runtime estimates in scheduling the IBM SP2 with backfilling. IEEE Trans. Parallel Distrib. Syst. **12**(6), 529–543 (2001)
24. Omara, F.A., Arafa, M.M.: Genetic algorithms for task scheduling problem. J. Parallel Distrib. Comput. **70**(1), 13–22 (2010)
25. Robert, Y., et al.: Task Graph Scheduling. Encyclopedia of Parallel Computing, pp. 2013–2025 (2011)
26. Srinivasan, S., Krishnamoorthy, S., Sadayappan, P.: A robust scheduling technology for moldable scheduling of parallel jobs. In: Proceedings IEEE International Conference on Cluster Computing CLUSTR-03, pp. 92–99. IEEE (2003)
27. Srinivasan, S., Kettimuthu, R., Subramani, V., Sadayappan, P.: Selective reservation strategies for backfill job scheduling. In: Feitelson, D.G., Rudolph, L., Schwiegelshohn, U. (eds.) JSSPP 2002. LNCS (LNAI and LNB), vol. 2537, pp. 55–71. Springer, Heidelberg (2002). https://doi.org/10.1007/3-540-36180-4_4
28. Sudholt, D.: Parallel evolutionary algorithms. In: Kacprzyk, J., Pedrycz, W. (eds.) Springer Handbook of Computational Intelligence, pp. 929–959. Springer, Heidelberg (2015). https://doi.org/10.1007/978-3-662-43505-2_46
29. Ye, D., Chen, D.Z., Zhang, G.: Online scheduling of moldable parallel tasks. J. Sched. **21**(6), 647–654 (2018)
30. Yoo, A.B., Jette, M.A., Grondona, M.: SLURM: simple linux utility for resource management. In: Feitelson, D., Rudolph, L., Schwiegelshohn, U. (eds.) JSSPP 2003. LNCS (LNAI and LNB), vol. 2862, pp. 44–60. Springer, Heidelberg (2003). https://doi.org/10.1007/10968987_3

Related Paper IV

Estimation of Execution Parameters for k-Wave Simulations

JAROS Marta, SASAK Tomas, TREEBY Bradley E. and JAROS Jiri

High Performance Computing in Science and Engineering (HPCSE) 2019
DOI: 10.1007/978-3-030-67077-1_7

Estimation of Execution Parameters for k-Wave Simulations

Marta Jaros[1(✉)], Tomas Sasak[1], Bradley E. Treeby[2], and Jiri Jaros[1]

[1] Faculty of Information Technology, Centre of Excellence IT4Innovations, Brno University of Technology, Bozetechova 2, 612 66 Brno, Czech Republic
{martajaros,jarosjir}@fit.vutbr.cz, xsasak01@stud.fit.vutbr.cz
[2] Medical Physics and Biomedical Engineering, Biomedical Ultrasound Group, University College London, Malet Place Engineering Building, London WC1E 6BT, UK
b.treeby@ucl.ac.uk

Abstract. Estimation of execution parameters takes centre stage in automatic offloading of complex biomedical workflows to cloud and high performance facilities. Since ordinary users have no or very limited knowledge of the performance characteristics of particular tasks in the workflow, the scheduling system has to have the capabilities to select appropriate amount of compute resources, e.g., compute nodes, GPUs, or processor cores and estimate the execution time and cost.

The presented approach considers a fixed set of executables that can be used to create custom workflows, and collects performance data of successfully computed tasks. Since the workflows may differ in the structure and size of the input data, the execution parameters can only be obtained by searching the performance database and interpolating between similar tasks. This paper shows it is possible to predict the execution time and cost with a high confidence. If the task parameters are found in the performance database, the mean interpolation error stays below 2.29%. If only similar tasks are found, the mean interpolation error may grow up to 15%. Nevertheless, this is still an acceptable error since the cluster performance may vary on order of percent as well.

Keywords: Workflow management system · Performance data collection · Interpolation · Job scheduling · HPC as a service

1 Introduction

Computation of complex scientific applications may no longer be satisfied by personal computers and small servers manually operated by highly experienced users. First, the extent of data being processed and the computational requirements highly exceed the capacity of such machines. Increasing number of applications is thus moving to the cluster or cloud environments. Second, scientific applications often feature a very complex processing workflows consisting of many particular tasks employing different computer codes, and complex data

T. Kozubek et al. (Eds.): HPCSE 2019, LNCS 12456, pp. 116–134, 2021.
https://doi.org/10.1007/978-3-030-67077-1_7

dependencies. Third, scheduling, execution and monitoring of such workflows require automated tools to remove the burden from the experienced users, enable ordinary users to routinely execute their applications, and increase the throughput of the computing facilities.

To face these challenges, the scientific and software development communities have adopted the workflow paradigm to describe the processing flow. The most common formalism used is the weighted directed acyclic graph (DAG) defining computational tasks by the nodes, and the dependencies and data movements by the edges. The weights in the nodes describe the computational requirements while the weights on the edges denote the amount of data being transferred between tasks [15].

In order to automate workflow execution, several workflow management systems (WMSs) have been developed and used within the scientific community. The most popular tools such as Pegasus [2,3], Globus [4] or Kepler [12] now offer automated execution of scientific workflows on remote computational resources in a more or less general way. However, these tools focus on expert users who know the behaviour of the computational codes used within the workflow, and are able to estimate the amount of computational resources needed by each task. The scheduling and mapping of the workflow on the computational resources are usually left to the cluster batch processing systems such as PBS[1] or Slurm[2].

These task schedulers provide their best effort to execute the tasks in the earliest possible time depending on the cluster workload and user/task priorities. However, what they cannot deal with is the execution parameters settings. If the user overestimates the amount of the computing resources, the tasks may be waiting in the queue for much longer time while making only little benefit from increased amount of resources, e.g., processor cores. On the other hand, underestimating these requirements may lead to the premature task termination due to exhausting the execution time.

This paper focuses on the heuristic-based selection of the execution parameters for a list of predefined computing codes used in the biomedical workflows supported by the k-Wave toolbox [18]. Since all binaries are fixed and known in advance, their performance characteristics such as strong and weak scaling can be automatically collected and used for prediction. Limiting the users in uploading their binaries also enables fine-grain performance tuning of the underlying codes for target machines and simplifies the workflows composition by the use of high-level processing blocks.

The next section describes the k-Plan system supporting the design of ultrasound workflows via a graphical user interface, and workflow offloading, scheduling, execution and monitoring using the k-Dispatch module. Section 3 describes a single pass optimization of the workflow execution parameters and related interpolation heuristics. Section 4 investigates the quality of interpolation for known and unknown tasks, and Sect. 5 concludes the paper.

[1] https://www.altair.com/pbs-works/.
[2] https://slurm.schedmd.com/.

2 Automatic Offloading of k-Wave Workflows

The k-Wave toolbox [18] is an open source Matlab toolbox designed for the time-domain simulation of acoustic waves propagating in tissues. The toolbox has a wide range of functionality, but at its heart is an advanced numerical model that can account for both linear and nonlinear wave propagation, an arbitrary distribution of heterogeneous material parameters, power law acoustic absorption and its thermal effects on the tissue. During recent years, k-Wave has attracted a lot of attention amongst biomedical physicists, ultrasonographers, neurologists and oncologists. Many k-Wave-based applications have been reported in photoacoustic breast screening [13], transcranial brain imaging [14], and high intensity focused ultrasound treatment planning for kidney [1,16], liver [7] or prostate tumour ablations [17].

However, all these applications require very intensive computations. During the last decade, the simulation core has been rewritten in C++ and parallelized by various technologies, such as OpenMP for shared memory systems [19], CUDA for GPU accelerated systems [10], and MPI for large distributed clusters [8]. These implementations now cover a wide range of ultrasound simulations in domains of various sizes reaching the limits of the top supercomputers.

To support clinicians in executing ultrasound workflows, a complex system called k-Plan [9], consisting of tree modules, is being developed, see Fig. 1:

1. TPM - Treatment Planning Module implements user front-end with the graphical user interface to compose the processing workflow. Advanced users may also use a Matlab interface or third-party applications.
2. DSM (k-Dispatch) - Dispatch Server Module is responsible for the workflow offloading to remote computing facilities. It also schedules particular tasks, estimates computing requirements, and monitors the workflow progress.
3. SEM - Simulation Execution Module covers the deployed binaries necessary to run particular tasks. Due to strict medical restrictions, all binaries have to be certified, thoroughly tested and properly deployed.

Although designed for the k-Wave toolbox, k-Dispatch remains as general as possible to support other applications and workflow types. User applications such as TPM communicate with k-Dispatch through the Web server, see Fig. 2. The Dispatch database maintains users and groups, their resource allocations, history of calculated and submitted workflows, available computing facilities, executable binaries with their performance characteristics, etc. Besides decoding the workflows, data transfers, monitoring and communication with remote computing facilities, the k-Dispatch core performs the optimization of the workflow execution parameters.

Users can create new ultrasound procedures by altering predefined workflow templates and packing them with the patient's data. Once delivered to k-Dispatch, the execution workflow is constructed from the provided input file. Next, the list of available computing resources is scanned to find a suitable one, e.g., the one with the lowest actual workload. Consequently, appropriate binaries for particular tasks are filled in to the workflow template according to the

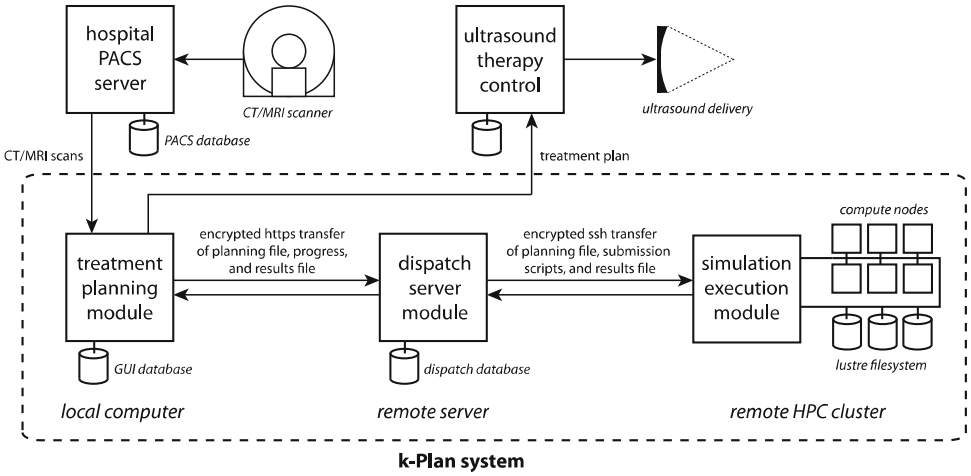


Fig. 1. Architecture of the k-Plan system. The dispatch server module (k-Dispatch) arranges for the workload scheduling, execution, monitoring and data transfers between client applications and computing facilities.

tasks input data size and available hardware. Since k-Dispatch knows the performance scaling of the given binaries, it can optimize the amount of computational resources (i.e., number of nodes) assigned to particular tasks and minimize several objectives such as cost, execution time and queuing time, see Algorithm 1.

After the tasks have been submitted to the computational queues, k-Dispatch keeps monitoring them, detects anomalies such as frozen/crashed jobs, and restarts them if necessary. After the workflow computation has been completed, the results are downloaded from the remote computing facility back to the k-Dispatch and the user is notified that the results are available for download.

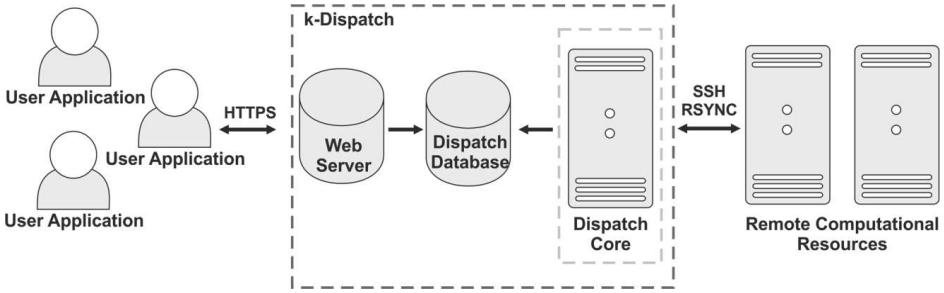


Fig. 2. k-Dispatch stands between user applications and remote computational resources. The communication with user applications is based on standard web services while the SSH protocol is used to communicate with remote computational resources. The dispatch core is responsible for the workflow submission, monitoring and other service mechanisms.

Algorithm 1: Adaptive execution planning algorithm

Presumptions :

1 Let $G = (V, E)$ be a workflow where V is a set of tasks and $E \subseteq V \times V$ is a set of task dependencies.
2 Let C be a set of active resource allocations with enough resources to satisfy the workflow G. It holds $C \subseteq A$, where A is a set of all allocations the user has got access to.
3 All executable binaries for supported task types available in a given allocation $a \in A$ are defined as $D \in (B_1, B_2, \ldots, B_N)$, where N is the number of task types within the workflow G, and $B_i = \{b_1, b_2, \ldots, b_M\}$ is the set of available binaries for a given task type. B_i may be an empty set.
4 Let $p : G \times C \times D \rightarrow \mathbb{R}^+$ be a price function returning the aggregated computational cost of the workflow G.
5 Let $t : G \times C \times D \rightarrow \mathbb{R}^+$ be a function returning the aggregated execution time of the workflow G. This value is calculated as a critical path through the workflow considering both the net execution time e and the queuing time q.
6 Let workflow evaluation f serving as quality metric be defined as $f = \alpha \cdot p + (1 - \alpha) \cdot t$, where α is a selectable ratio prioritizing the minimal computational cost or the execution time.

Algorithm :

1 Create a workflow $G = (V, E)$ from the workflow template and input data.
2 Select a set of candidate allocations $C = \{c \in A^+ \mid c.status == active \wedge c.hours\_left > 0.0\}$.
3 Set appropriate execution parameters for all tasks and evaluate the workflow G for all combinations of candidate allocations C and binary executables D.
4 Return the best parameters for a given workflow G as $\text{argmin}_{(c \in C, d \in D)} f(G)$.

3 Optimization of Workflow Execution Parameters

A typical course an ordinary user takes when executing a complex workflow is to use default execution parameters for each task, often consisting of one computational node and 24 h of wall time. If a task fails due to insufficient memory or time, another node or more time is allocated and the workflow restarted. Nevertheless, experienced users usually run a few benchmarks with various input sizes and number of nodes to create a strong scaling plot and predict the extent of computational resources for each task, which is the idea k-Dispatch has adopted.

In [9], three levels of workflow optimization were introduced. The naive one using the default execution parameters was implemented to compare k-Dispatch with other WMSs which use firmly set values directly provided by the users. This paper deals with a single-pass, task level optimization, processing each task independently. As we will show later, this is a viable solution with a linear time complexity providing sufficient results when execution cost and time is only considered. However, optimizing also for the queuing time requires a multi-pass, global optimization which may lead to an exponential time complexity, and needs a cluster simulator loaded with actual snapshots of cluster workload.

3.1 Single-Pass Optimization

The goal of the single-pass optimization is to independently find such execution parameters for each task i that minimize the workflow evaluation given by

$$f = \sum_{i=1}^{N} (\alpha * p_i + (1 - \alpha) * e_i) \tag{1}$$

where α is a weight preferring either execution cost or time, p is the execution cost and e is the net execution time. The queuing time is omitted here. Currently, the execution parameters to be optimized only cover the number of allocated nodes/cores and the execution time. Nevertheless, it is straightforward to extend the optimization to select the most suitable code, computational queue, node type (accelerated/fat/slim), etc.

Figure 3 illustrates the optimization of the task execution parameters as a black box with a task type and task input file provided by the workflow as the inputs. The task input file is parsed to extract information necessary to estimate the computational requirements. This information typically includes the size of the simulation domain, the simulation timespan, type of the medium, transducer definition, etc. Next, the collected performance data is searched to find similar records. Having a filtered out performance dataset, the plot of strong scaling can be constructed and several interpolation techniques can be used to estimate the task duration and cost for suitable amounts of resources. Once the best execution parameters are selected, the machine specific job scripts are generated and submitted to the computing queue. After the task has been properly finished, the performance data is used to update the performance database.

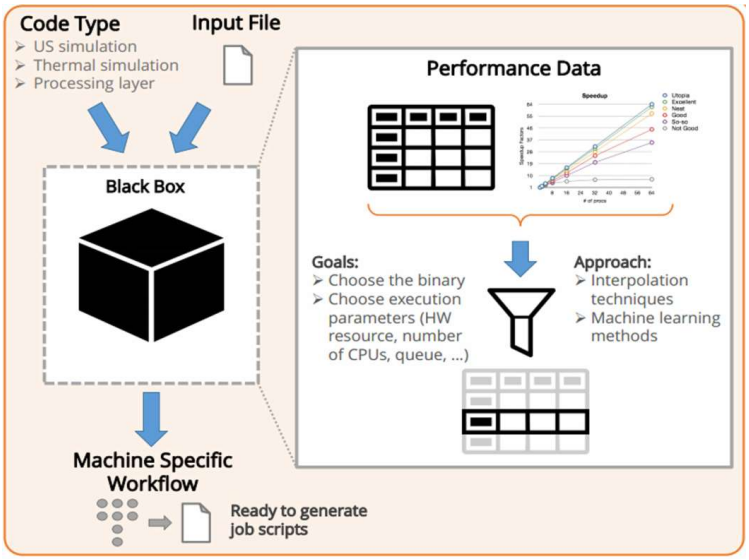


Fig. 3. Optimization of the execution parameter for a given task using a couple of heuristics and historically collected performance data for known code types.

3.2 Interpolation Heuristics

The goal of interpolation heuristics is to estimate the execution time and cost using the measured performance data from previous runs. Since the users are not limited in the size of the simulation domain and many other simulation parameters influencing the execution time, the performance data will never be complete.

There are three basic situations which may happen during the execution time and cost estimation:

1. The same simulation has been seen before. In such a case, the execution time and cost can be taken as a median value over multiple records stored in the database. If the values for particular amount of resources are unknown, an interpolation is used. Figure 4a shows this situation for four different domain sizes where the performance data are only known for 1, 2, 4 and 8 threads. The values for other numbers of threads have to be interpolated, see the question marks.
2. The simulation has not been seen before. In such a case, similar simulations are sought for in the database. First, the total number of grid points is calculated as a product of the dimension sizes. This may, however, unfavourably impact the estimation, since the actual shape does have an impact on the execution time, see Sect. 3.3. Next, all simulations with the number of grid points close to the one being estimated are selected. Finally, the execution time and cost are interpolated from the selected data. Figure 4b shows a situation where the performance data was only measured for 4 different domain sizes. The others have to be interpolated, see the yellow area.
3. The interpolation fails and it is necessary to use queue default wall time and amount of compute resources. This may happen if the simulation is too far from the known ones, or the interpolation method begins to oscillate and produces, e.g., negative values. Fortunately, this is a transient situation because as soon as the task is executed at least once, the measured values can be used next time.

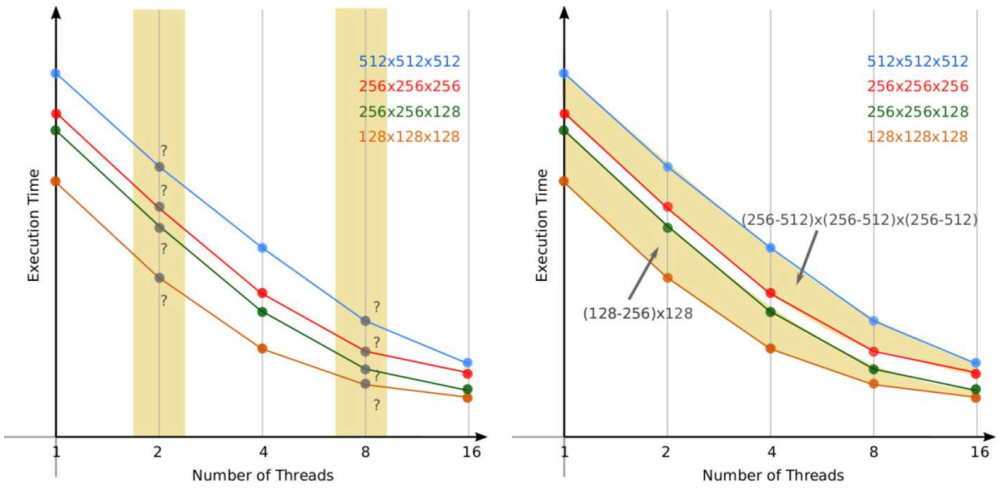
Four interpolation methods offered by the SciPy [20] Python package were investigated in this paper:

- linear interpolation (LI),
- cubic spline interpolation (CS),
- nearest neighbour interpolation (NN),
- radial basis function interpolation (RBF).

As the quality measure for the interpolation methods, *L1-*, *L2-* and *L-Infinity* norms were used [6]. Additionally, the mean percentage error of the obtained data series with respect to the measured values was calculated using Eq. (2).

$$meanPercentError = mean(\frac{|\mathbf{a} - \mathbf{b}|}{|\mathbf{a}|}) \times 100 \tag{2}$$

where $\mathbf{a}$ is a vector of reference data series and $\mathbf{b}$ is a vector of interpolated data series.



(a) Simulation size already seen. (b) Simulation size not seen before.

Fig. 4. (a) The performance database misses data (highlighted in yellow) for some numbers of threads. The interpolation works with the corresponding strong scaling curves. (b) The performance database misses data for a range of domain sizes (highlighted by yellow areas). The interpolations works with several strong scaling curves from the close proximity. (Color figure online)

3.3 k-Wave Workflow Properties

A typical biomedical ultrasound workflow consists of several data processing and numerical simulation tasks. Together, they form a workflow with approximately 100 tasks. Figure 5 shows an example of the neurostimulation workflow. While the pre- and post-processor tasks require only a single computing node, the aberration correction, forward planning, and thermal simulations may employ various executables to run on a single node, a single GPU, or multiple nodes.

The simulation domain size and timespan is given by the subject anatomy, transducer position, and the ultrasound frequency. Considering small animal neurostimulations, the domain sizes can be as small as $162 \times 192 \times 128$ grid points with 3,000 simulation time steps. The move towards human patients may expand the simulation domain size up to $768 \times 900 \times 600$ grid points with 16,800 simulation time steps.

Figure 6 shows the performance behaviour of the distributed MPI version of the k-Wave toolbox for the largest practical domain normalised to a single simulation time step. The execution times were measured on the Anselm supercomputer using 1 to 16 compute nodes, each of which with 16 cores and 64 GB of memory. It can be seen that the performance scaling is not perfect with the maximum speed-up of 6.5 yielding the parallel efficiency of 40%. The yellow, green and orange dots mark the ideal amount of computational resources for three different values of the α parameters. If the execution time is preferred, the highest possible number of nodes is selected. On the other hand, if the execution

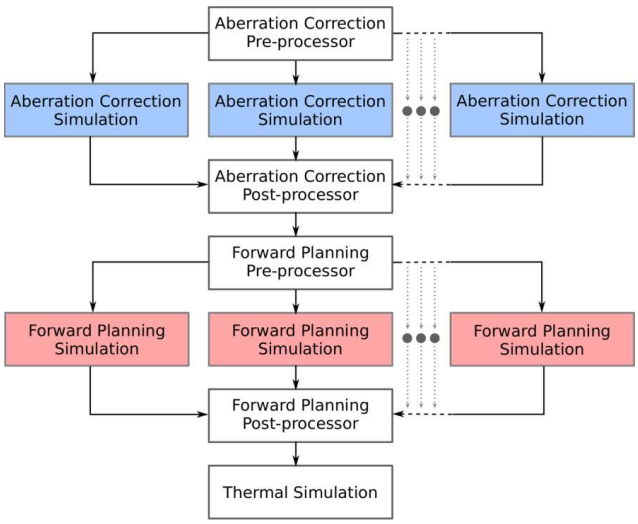


Fig. 5. A neurostimulation workflow consisting of several data processing and simulation tasks. The task dependencies are shown by the arrows, meaning the simulations depicted in red or blue may be executed concurrently. (Color figure online)

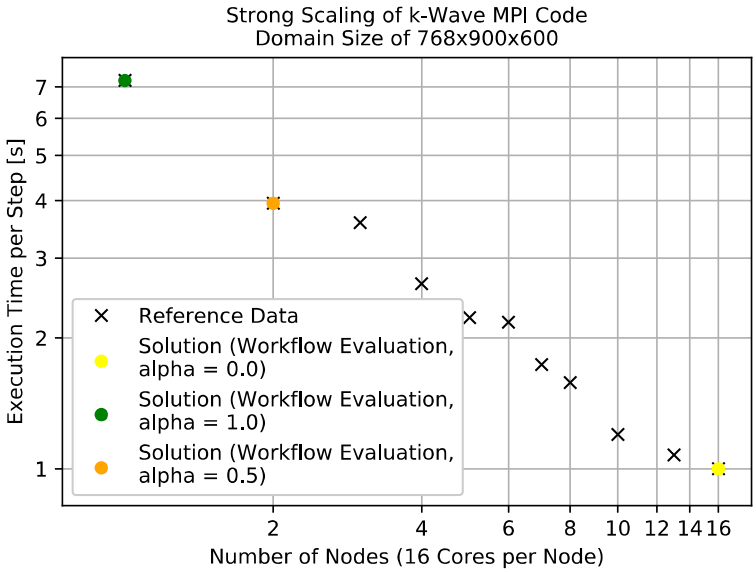


Fig. 6. Strong scaling of the MPI version of the k-Wave simulation in a domain consisting of $768 \times 900 \times 600$ grid points. The yellow, green and orange dots show the best number of nodes when minimizing the computational time, computational cost, or composite workflow evaluation, respectively. (Color figure online)

cost is preferred, a single node is selected. Finally, if both the time and cost have the same weight, two computing nodes looks as a good compromise.

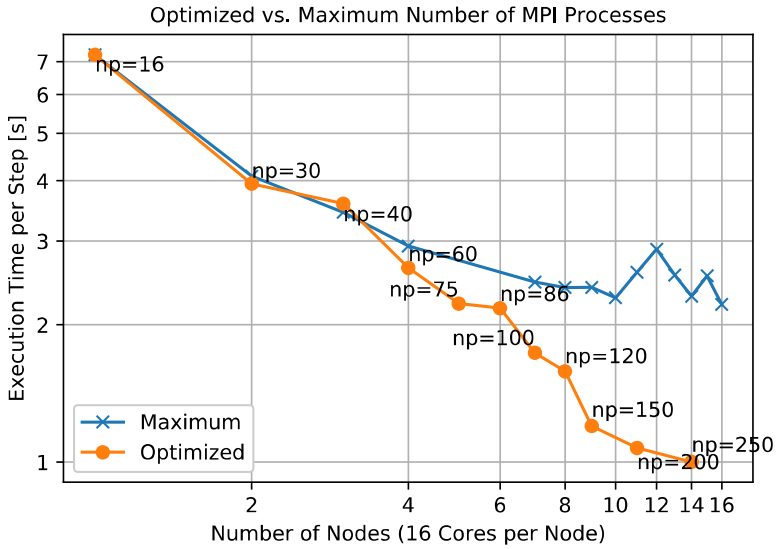


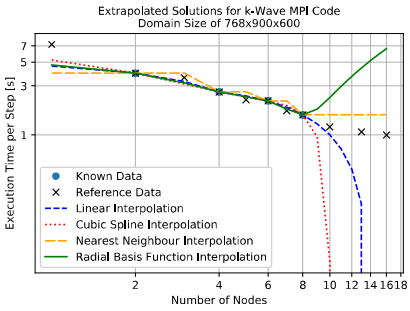
Fig. 7. Strong scaling of the MPI version of the k-Wave simulation on a domain of 768 × 900 × 600 domain size executed with the maximum (blue line) and optimal (orange line) numbers of MPI processes (np). (Color figure online)

When working with the MPI version of the k-Wave toolbox, balanced work distribution must be paid attention to. Since the code uses a one-dimensional grid decomposition over the z dimension, and the grid is z-y transposed several times every time step, the z and y dimensions must be divisible by the number of MPI processes. Otherwise, the work is not balanced evenly and the code does not scale well. Figure 7 shows the scaling of the code executed with the maximum numbers of MPI processes for given number of nodes, and with reduced numbers of processes ensuring commensurability. It is obvious, the optimized numbers of processes yield higher performance.

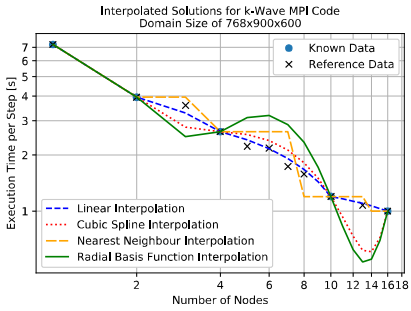
3.4 Typical Problems of Performance Data Interpolations

The interpolation and extrapolation methods have several drawbacks that will be discussed in this section. We used measured performance data from Fig. 6 and tried to manually fit interpolation curves through the measured data.

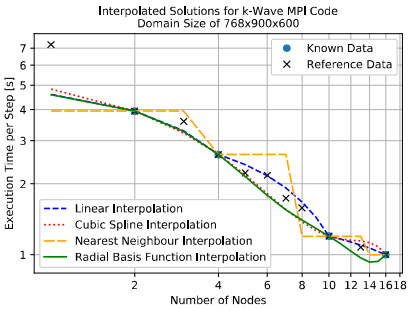
Generally, the k-Wave codes have a linearithmic computation complexity $O(n \cdot log \cdot n)$ due to extensive use of 3D Fourier transform. However, the significant amount of communication stemming from the distributed FFT may lead to quadratic communication complexity. Moreover, the proper workload balancing as well as other restrictions imposed on the domain size make the scaling even more difficult to predict [5,8]. Therefore, there are significant differences in the course of the scaling curves at low and high numbers of threads/nodes.



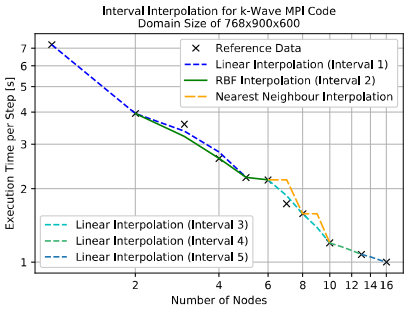
(a) Extrapolation based on the knowledge of scaling on 2, 4, 6 and 8 nodes.



(b) Interpolation based on the knowledge of scaling on 1, 2, 4, 10 and 16 nodes.



(c) Interpolation based on the knowledge of scaling on 2, 4, 10 and 16 nodes.



(d) Interval based interpolation, each interpolation uses 2 or 3 closest values.

Fig. 8. (a) Unsuccessful extrapolation trained on a small number of nodes. (b) and (c) Oscillation caused by distant known values. (d) Interval interpolation not suffering from the oscillations.

Figure 8a shows a poor attempt to extrapolation where the performance data is only known for 2, 4, 6 and 8 nodes. The estimation of the execution time for number of nodes above 8 is not acceptable. The linear extrapolation as well as cubic spline extrapolation predict much shorter execution time. Nevertheless, the code scales much worse for higher number of nodes because the communication component starts to dominate. The nearest neighbour extrapolation could be used as the worst case, however, Fig. 6 suggests that the performance can even deteriorate with higher number of compute nodes. Finally, the radial basis interpolation does not produce meaningful predictions.

Figure 8b and c point out the need to abide appropriate interval between known values to eliminate oscillations. Figure 8b uses an additional value for one node compared to Fig. 8c. This value is usually an outlier causing unintended oscillations since having no communication. To reduce them, several interpolations may be performed on smaller intervals. The impact of this technique is shown in Fig. 8d, where the scaling data is divided into 5 intervals of 2 to 3 values. However, it is not clear how to determine the interval size automatically.

4 Experimental Results

This section describes performed experiments and the results. The experiments show the application of the selected interpolation methods in order to autonomously find the suitable execution parameters.

Due to the necessity of collecting an extensive performance dataset, we limited ourselves to only consider the OpenMP k-Wave implementation running on a single node, however, with various numbers of threads. The execution cost was then calculated as a product of the execution time and the number of processor cores used. In principle, similar results are expected to be obtained for the CUDA implementation of k-Wave. On the other hand, the MPI version poses more restrictions and may feature different results, see Sect. 3.4.

The performance data collected for the OpenMP code was obtained on Anselm with 16 cores per node, and Salomon with 24 cores per node. The performance data was divided into the training and testing datasets both of which containing over 6,500 records of the aberration correction k-Wave simulation running over 24 different domain sizes (32^3 to 512^3 grid points) and with various number of threads.

4.1 Comparison of Interpolation Techniques for Known Simulation

We first investigated the behaviour of all four interpolation techniques on the known domain size of 512^3 grid points. The first experiment used 6 known execution times from the Anselm cluster measured for 1, 2, 4, 8, 12 and 16 threads. Table 1 and Fig. 9 show the course of the interpolation functions. It can be seen that the linear and cubic spline interpolation methods reached less than 3% mean error. The linear interpolation can be thought of as a pessimistic one since overestimating the execution times. Although this may lead to a bit longer queuing times, it is safer than underestimation produced by the cubic spline interpolation, which may lead to premature termination of the simulation. The nearest neighbour interpolation shows significantly worse accuracy as well as the radial basis function interpolation deeply oscillating, especially for high numbers of threads.

The second experiment extended the number of measured values and also included the Salomon cluster. For Anselm, the performance data was extracted from the database for 1, 2, 4, 5, 8, 10, 13, and 15 threads, while for Salomon

Table 1. Comparison of selected interpolation methods for domain size of 512^3 grid points domain size and 6 known values measured on Anselm.

Interpolation method	L1-Norm	L2-Norm	L2-Infinity Norm	Mean error [%]
Linear	1.27	0.59	0.46	2.89
Cubic Spline	0.93	0.42	0.35	2.29
Nearest Neighbour	4.60	2.40	2.06	9.85
Radial Basis Function	3.41	1.37	0.79	8.77

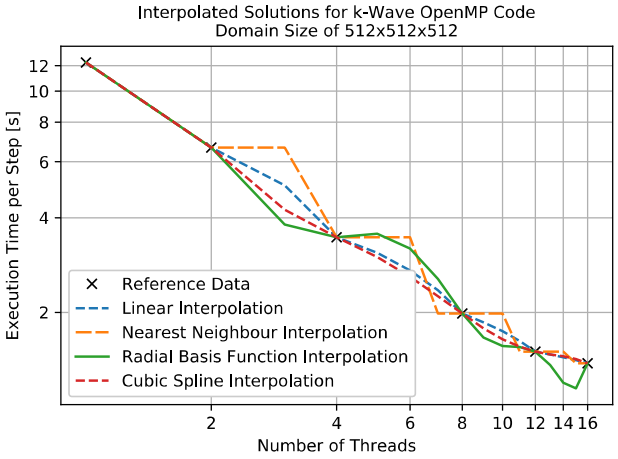


Fig. 9. Comparison of various interpolation techniques for the OpenMP implementation of k-Wave running on Anselm with a domain size of 512^3 grid points.

the set was further extended by performance data for 17, 20, 22, and 24 threads. This covers 50% of all possible thread numbers usable on both clusters. The domain size remained the same (512^3 grid points).

Tables 2 and 3 show significant improvement in the prediction accuracy. The mean error produced by the linear interpolation was reduced from 2.89% to 1.81%, and 1.27% on Anselm and Salomon, respectively. Even better results were achieved for the cubic spline interpolation which produced estimation with only 1.23% and 1.12% error. Even the other interpolation methods improved

Table 2. Comparison of selected interpolation methods for domain size of 512^3 grid points domain size and 8 known values measured on Anselm.

Interpolation method	L1-Norm	L2-Norm	L2-Infinity Norm	Mean error [%]
Linear	0.80	0.45	0.41	1.81
Cubic Spline	0.56	0.38	0.37	1.23
Nearest Neighbour	2.99	2.03	1.95	5.70
Radial Basis Function	1.61	0.90	0.67	4.67

Table 3. Comparison of selected interpolation methods for domain size of 512^3 grid points domain size and 12 known values measured on Salomon.

Interpolation method	L1-Norm	L2-Norm	L2-Infinity Norm	Mean error [%]
Linear	0.62	0.33	0.29	1.27
Cubic Spline	0.60	0.40	0.39	1.12
Nearest Neighbour	2.73	1.71	1.63	4.68
Radial Basis Function	1.08	0.69	0.66	2.01

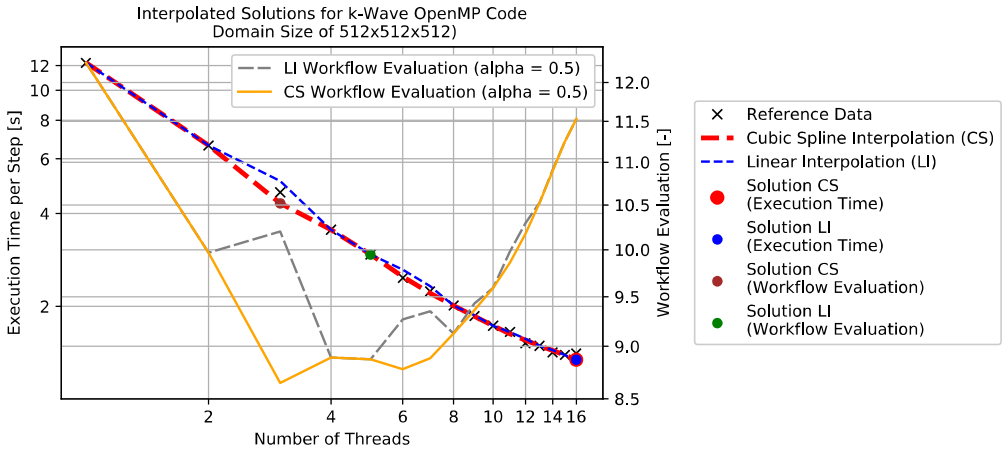


Fig. 10. Estimation of the best execution configuration according to the workflow evaluation function for a domain size of 512^3 grid points on the Anselm cluster produced by linear and cubic spline interpolation.

the error close to or below 5%. This can be considered as a very good result since there is always a slight variation in execution times between different runs caused by the underlying cluster workload (mainly network and I/O parts), and variations in clock frequency amongst different cluster nodes.

Figure 10 illustrates the result of the interpolation for linear and cubic spline interpolation for the extended training set, and the domain size of 512^3 grid points. The curves show a very good agreement without any significant oscillations. The orange and grey curves are the visualizations of the workflow evaluation functions with $\alpha = 0.5$. If looking for the fastest solution, both the linear and cubic spline interpolations predict 16 threads to be the best solution. In the case the combined workflow evaluation metric is minimized, 3 and 5 threads are predicted as best compromises by the cubic spline and linear interpolations, respectively.

4.2 Comparison of Interpolation Techniques for Unknown Simulations

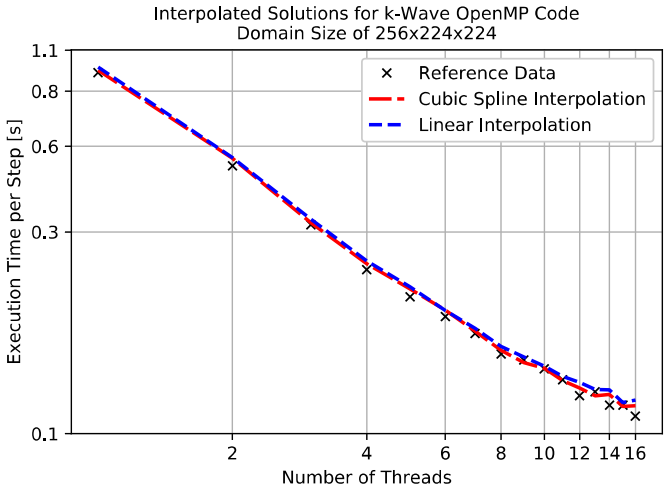
This set of experiments evaluates the capabilities of the proposed interpolation methods to estimate the execution time for simulations that have not been seen before. In this case, the closest simulations in terms of the total number of grid points are used to fit the interpolation curves. Since the results were similar for both clusters, we only present measurements on Anselm.

Three different unknown domain sizes were tested:

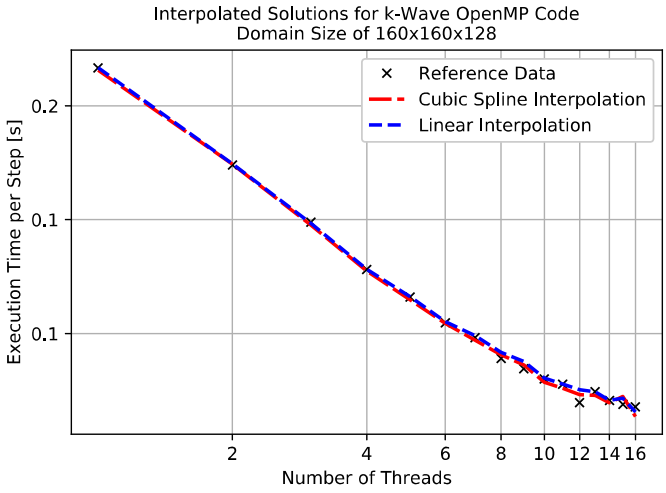
1. Tested simulation size 256×224^2, training set containing simulations of 224^3, $256^2 \times 224$, and $224^2 \times 192$ grid points.
2. Tested simulation size $160^2 \times 128$, training set containing simulations of 144^3, 160^3, and 132×128^2 grid points.

3. Tested simulation size 144^3, training set containing simulations of 160^3, 160×128^2, and 132×128^2 grid points.

Figure 11 shows the results of selected interpolations on the first two simulation domains. Both linear and cubic spline interpolations show a very close agreement with the reference data stored in the testing set. As Tables 4 and 5 quantify, the mean error for the biggest domain reaches 4.7% and 3.1% for linear and cubic spline interpolations, respectively. For the smaller domain, the error decreases to 1.75% and 2.25%. Interestingly, the cubic spline produces slightly



(a) Domain size of 256×224^2 grid points



(b) Domain size of $160^2 \times 128$ grid points

Fig. 11. Comparisons of linear and cubic spline interpolation methods for unknown domain sizes. The reference data points are used for the error evaluation.

Table 4. Comparisons of selected interpolation methods for an unknown domain sizes of 256×224^2 grid points.

Interpolation method	L1-Norm	L2-Norm	L2-Infinity Norm	Mean error [%]
Linear	0.17	0.056	0.034	4.724
Cubic Spline	0.11	0.037	0.025	3.073
Nearest Neighbour	0.84	0.271	0.191	22.35
Radial Basis Function	352	99.26	47.99	11492

Table 5. Comparisons of selected interpolation methods for an unknown domain sizes of $160^2 \times 128$ grid points.

Interpolation method	L1-Norm	L2-Norm	L2-Infinity Norm	Mean error [%]
Linear	0.015	0.005	0.003	1.75
Cubic Spline	0.023	0.007	0.005	2.25
Nearest Neighbour	0.252	0.089	0.068	17.7
Radial Basis Function	0.371	0.121	0.089	29.2

worse estimations here. The nearest neighbour interpolation gives much worse estimation with a mean error of 22% and 18% for those two cases. Finally, the radial basis interpolation appears to be unusable for the largest domain. The extreme error is caused by high oscillations. In case of the medium-sized domain, the error decreases to 29%. Unfortunately, this still exceeds acceptable values.

The smallest domain size of interest suffers from very poor results which are summarized in Table 6 and Fig. 12. The only usable estimations are provided by the linear interpolation, however, with a mean error of 16%. The cubic spline completely fails in this case while the best estimation is surprising provided by the nearest neighbour interpolation. The radial basis interpolation also fails on this domain size. The overestimation is very likely caused by a small domain size when a single grid can fit into L3 cache memory leading to much faster execution of the Fourier transforms and overall algorithm speed-up. On the other hand, even overestimation by 200% may be thought of as acceptable considering such a simulation is executed within 2 min using 16 threads.

Table 6. Comparisons of linear and cubic spline interpolation methods for an unknown domain sizes of 144^3 grid points.

Interpolation method	L1-Norm	L2-Norm	L2-Infinity Norm	Mean difference [%]
Linear	0.196	0.061	0.041	15.99
Cubic Spline	2.080	0.527	0.185	212.6
Nearest Neighbour	0.177	0.064	0.050	13.40
Radial Basis Function	4.050	1.024	0.356	416.8

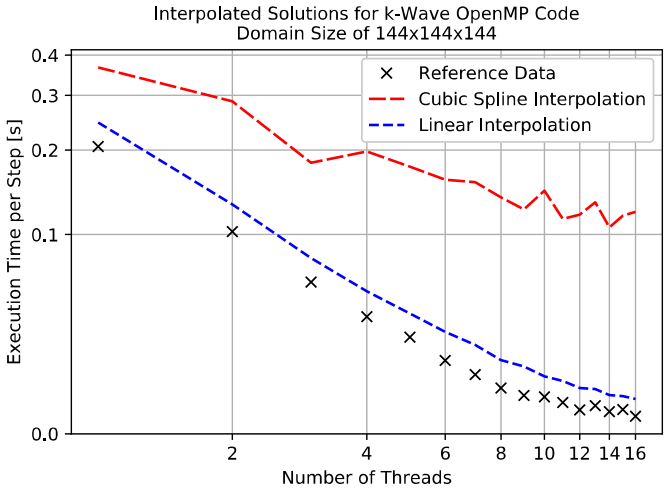


Fig. 12. Comparisons of linear and cubic spline interpolation methods for unknown domain size of 144^3. The reference data points are used for the error evaluation.

5 Conclusions

The need for offloading complex scientific workflows to cluster and cloud environment is ubiquitous. k-Dispatch is a workflow management system providing automated execution, planning and monitoring of biomedical workflows composed of k-Wave ultrasound and thermal simulations. Its interface enables connection of various user applications and unifies the access to different computational resources.

One of the key challenges in automated execution of complex workflows is the proper setting of execution parameters for particular tasks. Since the end users have no or very limited knowledge about the amount of computational resources to be allocated for each task, it is necessary to provide as good estimation as possible based on the performance characteristics of particular codes and actual input data. Unsuitable values may lead to long queueing times or early tasks termination due to exhausted time allocation.

This paper has presented a single pass algorithm traversing the workflow and optimizing the execution parameters for every task independently. For every task, the input file is inspected, the task parameters retrieved, and the performance database searched for similar ones. If there is a direct match, the execution time and cost are loaded for known execution parameters, i.e., number of compute nodes, GPUs, processor cores, etc. Missing values may be filled in using interpolation techniques. However, if the task parameters have not been seen before, the interpolation is used to estimate the execution time and cost using a training set composed of tasks with similar parameters.

Four different interpolation techniques have been investigated. When the task parameters have been seen before, the cubic spline interpolation showed the best results with mean error between 1.12% and 2.29%. In the case the

task parameters have not been seen before, the linear interpolation showed the best results. Depending on the similarity of the records found in the performance database, the mean error varies between 1.17% and 15%. It should be noted that the highest error showed up only for very small tasks where the overestimation of execution time or cost do not play a significant role.

5.1 Future Work

Future work will be focused on multi-pass optimization of workflow execution parameters. The goal is to minimize not only the execution time and cost but also the queuing times. This however requires the knowledge of the actual cluster workload and queues occupancy as well as a cluster simulator to quickly estimate the queuing times for the whole workflow under different execution parameters. We are considering the adaptation of the ALEA simulator [11] to match the scheduling algorithms and hardware configurations of IT4Innovations clusters, and the characteristics of the k-Wave workflows.

We would also like to implement more sophisticated heuristics to select an appropriate number of compute nodes as well as optimal number of MPI processes for large simulations to avoid performance penalizations. Consequently, we would like to study machine learning methods since we expect to have collected large performance dataset, and perform experiments on both, artificial and real-world workflows.

Acknowledgement. This work was supported by the FIT-S-17-3994 Advanced parallel and embedded computer systems project. This work was supported by The Ministry of Education, Youth and Sports from the National Programme of Sustainability (NPU II) project IT4Innovations excellence in science - LQ1602 and by the IT4Innovations infrastructure which is supported from the Large Infrastructures for Research, Experimental Development and Innovations project IT4Innovations National Supercomputing Center - LM2015070. This work was supported by the Engineering and Physical Sciences Research Council, United Kingdom, grant numbers EP/L020262/1, EP/M011119/1, EP/P008860/1, and EP/S026371/1.

References

1. Abbas, A., Coussios, C., Cleveland, R.: Patient specific simulation of HIFU kidney tumour ablation, vol. 2018, pp. 5709–5712 (July 2018). https://doi.org/10.1109/EMBC.2018.8513647
2. Su, M.H.: Pegasus: a framework for mapping complex scientific workflows onto distributed systems. Sci. Program. **13**, 219–237 (2005)
3. Deelman, E., et al.: Pegasus: a workflow management system for science automation. Fut. Gener. Comput. Syst. **46**, 17–35 (2014)
4. Foster, I.: Globus toolkit version 4: software for service-oriented systems. J. Comput. Sci. Technol **21**(4), 513–520 (2006). https://doi.org/10.1007/s11390-006-0513-y
5. Frigo, M., Johnson, S.: The design and implementation of FFTW3. Proc. IEEE **93**(2), 216–231 (2005). https://doi.org/10.1109/JPROC.2004.840301

6. Gradshtein, I.S.: Table of Integrals, Series, and Products. Academic Press, San Diego (2000)
7. Grisey, A., Yon, S., Letort, V., Lafitte, P.: Simulation of high-intensity focused ultrasound lesions in presence of boiling. J. Ther. Ultrasound (2016). https://doi.org/10.1186/S40349-016-0056-9
8. Jaros, J., Rendell, A.P., Treeby, B.E.: Full-wave nonlinear ultrasound simulation on distributed clusters with applications in high-intensity focused ultrasound. Int. J. High Perform. Comput. Appl. **30**(2), 137–155 (2016). https://doi.org/10.1177/1094342015581024
9. Jaros, M., Treeby, B.E., Georgiou, P., Jaros, J.: k-Dispatch: a workflow management system for the automated execution of biomedical ultrasound simulations on remote computing resources. In: Proceedings of the Platform for Advanced Scientific Computing Conference, PASC 2020. Association for Computing Machinery, New York (2020). https://doi.org/10.1145/3394277.3401854
10. Kadlubiak, K., Jaros, J., Treeby, B.E.: GPU-accelerated simulation of elastic wave propagation. In: 2018 International Conference on High Performance Computing & Simulation (HPCS), pp. 188–195. IEEE (July 2018). https://doi.org/10.1109/HPCS.2018.00044
11. Klusacek, D., Toth, S., Podolnikova, G.: Complex job scheduling simulations with Alea 4. CEUR Workshop Proc. **1828**, 53–59 (2017). https://doi.org/10.1145/1235
12. Ludäscher, B., et al.: Scientific workflow management and the Kepler system. Concurrency Comput. Pract. Exp. **18**(10), 1039–1065 (2006). https://doi.org/10.1002/cpe.994
13. Manohar, S., Dantuma, M.: Current and future trends in photoacoustic breast imaging. Photoacoustics **16**, 100134 (2019). https://doi.org/10.1016/j.pacs.2019.04.004
14. Mohammadi, L., Behnam, H., Tavakkoli, J., Avanaki, M.R.: Skull's photoacoustic attenuation and dispersion modeling with deterministic ray-tracing: towards real-time aberration correction. Sensors (Switzerland) (2019). https://doi.org/10.3390/s19020345
15. Robert, Y.: Task graph scheduling. In: Padua, D. (ed.) Encyclopedia of Parallel Computing. Springer, Boston (2011). https://doi.org/10.1007/978-0-387-09766-4_42
16. Suomi, V., Jaros, J., Treeby, B., Cleveland, R.: Nonlinear 3-D simulation of high-intensity focused ultrasound therapy in the Kidney. Conf. Proc. IEEE Eng. Med. Biol. Soc., 5648–5651 (2016). IEEE. https://doi.org/10.1109/EMBC.2016.7592008
17. Suomi, V., et al.: Transurethral ultrasound therapy of the prostate in the presence of calcifications: a simulation study. Med. Phys. **45**, 4793–4805 (2018). https://doi.org/10.1002/mp.13183
18. Treeby, B.E., Cox, B.T.: k-Wave: MATLAB toolbox for the simulation and reconstruction of photoacoustic wave-fields. J. Biomed. Opt. **15**(2), 021314 (2010)
19. Treeby, B.E., Jaros, J., Rendell, A.P., Cox, B.T.: Modeling nonlinear ultrasound propagation in heterogeneous media with power law absorption using a k-space pseudospectral method. J. Acoust. Soc. Am. **131**(6), 4324–4336 (2012). https://doi.org/10.1121/1.4712021
20. Virtanen, P., Gommers, R., Oliphant, T.E., et al.: SciPy 1.0: fundamental algorithms for scientific computing in Python. Nat. Meth. **17**, 261–272 (2020)

Related Paper V

Optimization of Execution Parameters of Moldable Workflows under Incomplete Performance Data

JAROS Marta and JAROS Jiri

Job Scheduling Strategies for Parallel Processing (JSSPP) 2022
DOI: 10.1007/978-3-031-22698-4_8

Optimization of Execution Parameters of Moldable Ultrasound Workflows Under Incomplete Performance Data

Marta Jaros(✉) and Jiri Jaros

Faculty of Information Technology, Centre of Excellence IT4Innovations, Brno University of Technology, Brno, Czech Republic
{martajaros,jarosjir}@fit.vutbr.cz

Abstract. Complex ultrasound workflows calculating the outcome of ultrasound procedures such as neurostimulation, tumour ablation or photoacoustic imaging are composed of many computational tasks requiring high performance computing or cloud facilities to be computed in a sensible time. Most of these tasks are written as moldable parallel programs being able to run across various numbers of compute nodes. The number of compute nodes assigned to particular tasks strongly affects the overall execution and queuing times of the whole workflow (makespan) as well as the total computational cost.

This paper employs a genetic algorithm searching for a good resource distribution over the particular tasks, and a cluster simulator evaluating the makespan and cost of the candidate execution schedules. Since the exact execution time cannot be measured for every possible combination of the task, input data size, and assigned resources, several interpolation techniques are used to predict the task duration for a given amount of compute resources. The best execution schedules are eventually submitted to a real cluster with a PBS scheduler to validate the whole technique.

The experimental results confirm the proposed cluster simulator corresponds to a real PBS job scheduler with a sufficient fidelity. The investigation of the interpolation techniques showed that incomplete performance data can successfully be completed by linear and quadratic interpolations keeping the maximum mean error below 10%. Finally, the paper introduces a user defined parameter instructing the genetic algorithm to prefer either the makespan or cost, or find a suitable trade-off.

Keywords: Task graph scheduling · Workflow · Genetic algorithm · Moldable tasks · Makespan estimation · Performance scaling interpolation

1 Introduction

All fields of science and engineering use computers to reach new findings, while the most compute power demanding problems require High Performance Computing (HPC) or Cloud systems to give answers to their questions. The problems

D. Klusáček et al. (Eds.): JSSPP 2022, LNCS 13592, pp. 152–171, 2023.
https://doi.org/10.1007/978-3-031-22698-4_8

being solved nowadays are often very complex and comprise a lot of various tasks with mutual dependencies describing different aspects of the investigated problem. Their computation can be formally described using scientific workflows [2], also referred to as task graphs [22].

Ultrasound computing workflows aim at various applications of the ultrasound such as neurostimation, tumour ablation, targeted drug delivery, or photoacoustic imaging [23]. The goal of ultrasound treatment workflows is to asses the outcome the treatment and adjust the parameters of the ultrasound traducers to deliver the acoustic energy into desired area while preventing any damage to heathy tissue. The goal of photoacoustic imaging is to reconstruct the tissue structure by running an iterative inverse ultrasound models on signals recorded at the body surface [20]. Since the wavelength of the ultrasound signals are very small compared to the investigated area, e.g. human head or chest, and there are tight deadlines by when the simulation outcome has to be delivered, it is necessary to optimize the workflow execution to reduce both the execution time as well as the cost.

The execution of scientific workflows on HPC systems is performed via communication with the HPC front-end, also referred to as the job scheduler [11]. After the workflow data has been uploaded to the cluster, the workflow tasks are submitted to the computational queues where waiting until the system has enough free resources, and all task dependencies have been resolved (predecessor tasks have been finished).

Modern HPC schedulers implement advanced techniques for efficient task and resource management [12]. However, the queuing time, computation time and related cost depend on the task execution parameters provided at submission. These parameters include the required execution time accompanied by the number of compute nodes, cores and accelerators, the amount of main memory and storage space, and more and more frequently, the frequency and power cap of various hardware components. In most cases, only experienced users are endowed by sufficient knowledge to estimate these parameters appropriately knowing the size of the input data for particular tasks. In other cases, default parameters may be chosen leading to inefficient workflow processing.

Complex compute tasks are usually written as moldable distributed programs being able to exploit various amounts and types of computing resources, i.e., they can run on different numbers of compute nodes. However, the moldability is often limited by many factors, the most important of which being the domain decomposition [4] and parallel efficiency (strong scaling) [1]. The goal of the workflow execution optimization is posed as the assignment of suitable amount of compute resources to individual tasks in order to minimize the overall computation time and cost.

While the field of rigid workflow optimization, where the amount of resources per task is fixed or specified by the user in advance, has been thoroughly studied, and is part of common job schedulers such as PBSPro [11] or Slurm [28], the autonomous optimization and scheduling of moldable workflows has still been an outstanding problem, although firstly opened two decades ago in [8].

During the last decade, many papers have focused on the prediction of rigid workflow execution time and enhancing the HPC resource management. For example, Chirkin et al. [5] introduces a makespan estimation algorithm that may be integrated into job schedulers. Robert et al. [22] gives an overview of task graph scheduling algorithms. The usage of genetic algorithms addressing the task scheduling problems has also been introduced, e.g., a task graph scheduling on homogeneous processors using genetic algorithm and local search strategies [13], and performance improvement of the used genetic algorithm [19]. However, a handful works have taken into the consideration the moldability and strong scaling behavior of particular tasks, their dependencies and the current cluster utilization [3,7,27].

In all cases, the estimation of the makespan and optimization of the tasks execution parameters rely on the performance database storing strong and weak scaling. However, it is often not possible to benchmark the execution time for all possible combinations of the task type, task inputs and execution parameters. If a task has already been executed with given inputs and execution parameters, the execution time can be retrieved from the performance database. However, for unseen combinations, some kind of interpolation or machine learning techniques have to be used.

In our previous work [16], Genetic Algorithms (GA) [10] and a simple cluster simulator were used to find optimal execution parameters for various workflows on systems with on-demand and static allocations. This paper follows up with our previous work and its main goals are to (1) prove that GA is able to find execution plans for different workflows when using incomplete performance datasets, (2) prove a trade-off parameter to find different solutions meeting contradictory optimization criteria can be introduced, and finally (3) extend the cluster simulator by adding support for backfilling and considering the initial cluster workload. The resilience of the optimization techniques will be investigated on several scenarios and validated against the real workflow makespan measured on the Barbora supercomputer[1].

2 Automatic Optimization of Workflow Execution Parameters

Selection of suitable execution parameters for workflow tasks plays a crucial role in scheduling process and the maskespan/cost optimization. A naive selection of the execution parameters often leads to various unpleasant situations such as unnecessarily long waiting times and idling nodes if high amounts of compute resources were chosen, or on the other hand, premature task termination and crashes if the amount of compute resources was not sufficient.

Even having enough experience with applications used within the workflow, setting the execution parameters properly to get good performance is a difficult and tedious task. The key to get short makespan is to look at the workflow as a

[1] IT4Innovations, Czech republic, https://docs.it4i.cz/barbora/introduction/.

whole. There are many dependencies among tasks and selection of best execution parameters for each task independently may lead to a suboptimal solution since there is a limited total amount of resources offered by the HPC facilities.

Although batch schedulers implement several optimization methods and heuristics to maintain high cluster utilization and low queueing times, bad execution parameters spoil their submission schedules, e.g., when tens of tasks enter the queue asking for 24 h allocations but actually finishing after an hour.

2.1 k-Dispatch Workflow Management System

Molding scientific workflows during the scheduling process goes beyond the capabilities of common batch job schedulers which schedules tasks independently only paying attention to their dependencies and requires the resource requirements to be specified in advance. For the modlable workflow scheduling, a workflow management system sitting in between the end user and the batch job scheduler is required [18,24]. k-Dispatch [18] is a Workflow Management System (WMS) [6,26] allowing the end users to submit complex workflows with associated data via a simple web interface and have them automatically executed on remote HPC facilities. Although oriented on the ultrasound community and the popular k-Wave acoustic toolbox [24], its general design allows simple adaptation to other workflows and toolboxes by integrating new task graphs, registering new binaries and adding performance tables.

k-Dispatch consists of three main modules depicted in Fig. 1: Web server, Dispatch database and Dispatch core. The user applications, e.g., a standalone medical GUI, Web application, or Matlab interface, communicate with the Web server using the secured HTTPS protocol and REST API. The Dispatch database holds all necessary information about the users, submitted workflows, jobs, computational resources, available binaries and the performance data collected over all executed tasks suitable for the execution time estimation. The Dispatch core is responsible for planning, executing and monitoring submitted workflows. The communication with HPC and cloud facilities is done via SSH and RSYNC protocols. For more information, please refer to [18].

2.2 Workflow Optimization Within k-Dispatch

The optimization algorithm providing suitable parameters for particular tasks of the workflow is integrated inside the Dispatch core. It is composed of four modules: Optimizer, Estimator, Evaluator and Collector [16].

The Optimizer is based on a Genetic Algorithm implemented in the PyGAD library [9] and its parameter settings have been thoroughly investigated in [16]. The goal of the Optimizer is to generate high quality candidate solutions, each of which holding a list of execution parameters for all tasks in the workflow. In the simplest case, a candidate solution is a vector where the position of the task is given by a breath first traversal through the workflow task graph and the value determines the number of compute nodes to be used. Although several heuristics has been proposed to optimize the execution parameters [7,14,27], they have

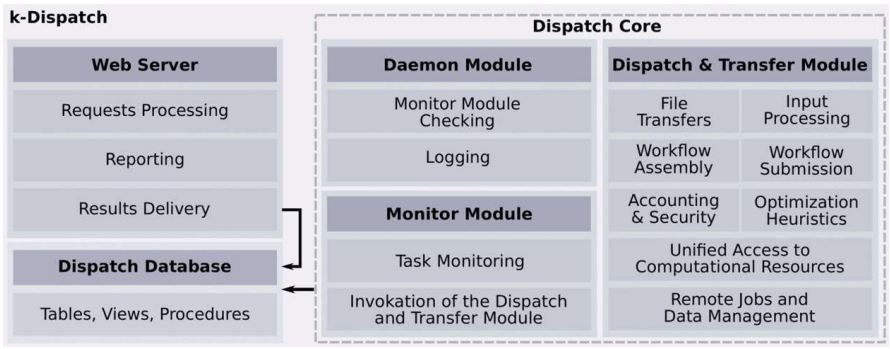


Fig. 1. k-Dispatch's modules and a brief description of the actions each module is responsible for. Arrows show the communication between Dispatch Core, Web Server and Dispatch Database.

strong limitations such as no dependencies among tasks or monotonic strong scaling. The genetic algorithm allows to solve any instance of the optimization problem.

The Estimator is responsible for estimating the execution time for particular tasks based on their input data and the amount of required resources. The Estimator incorporates various interpolation heuristics to reckon up missing values in strong and weak scaling.

The Evaluator uses a simplified simulator of job scheduler called Tetrisator [16], which takes a candidate schedule, simulates its execution on a given cluster and calculates the workflow makespan and cost. Tetrisator is a one-pass simulator of an HPC system with a predefined number of uniform computing nodes. It is inspired by the default strategy of the PBS job scheduler. In this paper, its functionality was extended by the backfilling technique allowing smaller jobs to overtake larger ones if no delays is introduced. The tasks are submitted to the simulator in the order defined in the candidate solution. Workflows may contains multiple dependencies among inner tasks, and the initial cluster workload may be defined, i.e., the cluster is not empty at the workflow submission time.

As soon as a satisfactory solution is found, the workflow is submitted to the real cluster and executed. Upon finishing the execution, the execution times for all tasks are collected by the Collector and stored in the performance database. This data is used to gradually improve the accuracy of the Estimator.

2.3 Estimator Module and Interpolation Techniques

There are many factors that may affect the execution time of a given task. Obviously, the most important ones are the size of the problem stored in the input file and the amount of resources assigned to the task. However, there might be many additional aspects significantly impacting the execution time such as data distribution and load balance, varying time complexity of the algorithms

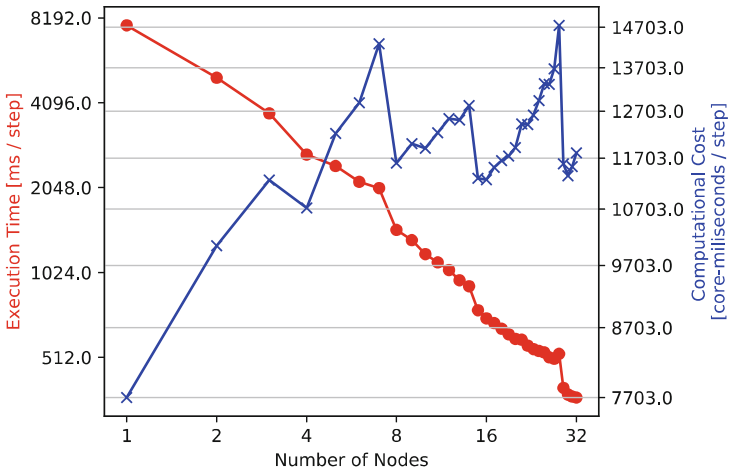


Fig. 2. Red line shows the strong scaling of the k-Wave code measured for a domain size of 1024^3 grid points on the Barbora cluster. Blue line shows the evolution of the computational cost when more nodes are added. (Color figure online)

used, additional task parameters and the amount of data being stored during the task execution.

As a practical example, let us talk about the MPI implementation of the k-Wave toolbox [15] simulating (non)-linear propagation of ultrasound wave through a heterogeneous absorbing medium. The scaling of the execution time and cost for one specific problem instance on the Barbora cluster with 36 processor cores per node can be seen in Fig. 2. Here, a domain of 1024^3 grid points is partitioned into 2D slabs and distributed over various numbers of compute nodes (1 to 32). The red curve shows the execution time per one simulation time step (the whole simulation usually executes tens of thousands of time steps).

Although this strong scaling curve looks almost ideal, several sudden drops in the execution time can be observed. These drops are the consequences of well balanced workload distribution. For example, if we cut the domain into 512 slices, we can distribute the work over 512 ranks mapped onto 512 cores. Since k-Wave is a memory and network bound application, it is often advantageous to undersubscribe the computing nodes and use higher aggregated memory and network bandwidth. On the Barbora cluster, we can spread 512 ranks over 15 to 28 nodes in a round robin fashion. Since the efficiency of such distribution is decreasing, the scaling curve is flattening toward 28 nodes. However, when 29 nodes are allocated to the task, the domain can be cut into 1024 slices leading to a much better workload distribution and significantly lower execution time. This imperfect workload distribution also renders into the simulation cost since there is a direct proportion between the parallel efficiency and the related cost. The blue curve shows several local minima and maxima in the execution cost which provide very suitable execution parameters or should be avoided, respectively.

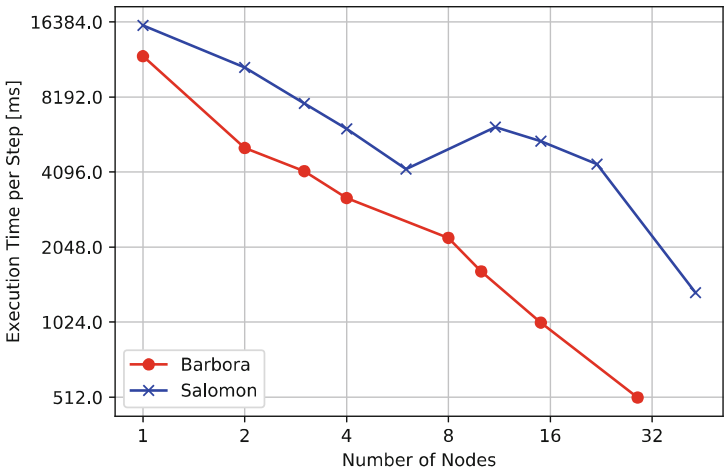


Fig. 3. Strong scaling of the k-Wave code execution time measured for 1024^3 domain size on the Barbora (36 cores/node) and Salomon (24 cores/node) cluster.

Let us note that having a complete performance dataset with all possible input sizes, numbers of nodes, and other tens of simulation parameters is computationally intractable. When having incomplete performance datasets where some points on the scaling curve are missing, the interpolation should rather overestimate the execution time to prevent premature task termination. Even more important question is how the scaling curve changes when a previously unseen domain size is used. In this situation, it is necessary to estimate both the shape and the position of the scaling curve from measured strong and weak scaling. As interpolation functions, linear and quadratic interpolation were used.

Finally, the scaling curves may change significantly among different machines. One such an example can be seen in Fig. 3 where the same problem is solved on Barbora (36 Cascade Lake cores per node) and Salomon (24 Haswell cores per node). Not only is the curve shifted due to a lower node performance, but it has a very different shape in the second half. This may be the effect of a different interconnection network topology, but also current cluster utilization. In this case, it may be very hard to use any interpolation. Thus when a new cluster is connected to k-Dispatch, a few benchmark runs for the most typical simulation settings are performed to get a minimum amount of performance data.

2.4 Evaluator Module Improvement

Since our previous work [16], Tetrisator has been extended by implementing the backfilling technique [21] to simulate the real batch scheduler more accurately.

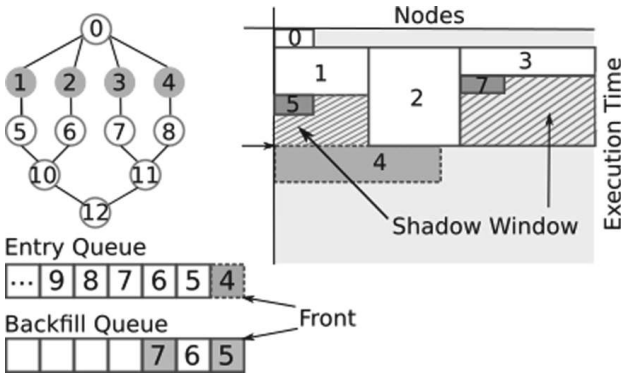


Fig. 4. The state of the entry and backfill queues when the task no 4 is about to be executed. Workflow tasks came to the entry queue in the execution order. Tasks 0–3 have already been executed. When task 4 is to be executed, the scheduler (depicted as schedule on the right) gets short of free nodes, and allows task 5 and 7 to overtake task 4 and fill the *shadow window*.

Tetrisator schedules tasks in the same order as they come to the HPC system (breadth first traversal of the workflow) [16]. The tasks are waiting in an entry queue until executed. A task at the front of the entry queue is ready for execution when all the task dependencies have been fulfilled and there are enough free resources. If this condition is not met, backfilling may find its place. Provided that the execution time of the waiting task will not be postponed and any task dependencies will not be offended, tasks requiring smaller amount of resources may overtake the waiting task. The task to be backfilled is calculated in a so called *shadow window*. The implemented algorithm is depicted in Fig. 4.

Real batch schedulers[2] may implement more sophisticated criteria for backfilling. For example, user and task priorities may be taken into account when deciding which tasks may overtake the waiting ones (fair-share policy). Since we mainly aim at static allocations where users do not compete, this calculation is omitted, i.e., all tasks and users have the same priority and only task dependencies are considered.

The implemented backfilling algorithm considers a queue of jobs that could be possibly backfilled, i.e., the *backfill queue* of length n and width of 1. n stands for a positive number of tasks that have the capability to be backfilled. Width of 1 means that the dependent tasks on the direct candidates to backfill are not considered. In other words, let us have task A actually being calculated, task B waiting for task A and other two tasks C and D. Task D depends on task C. Task C is not dependant on any other task and since not offending the execution time of task B, it can be directly added to the backfill queue. Task D could be also executed and finished within the *shadow window* as task C as well as still not running out of available resources, however, since dependent on task C it is not added to the *backfill queue* (attacking width of 2).

[2] https://docs.it4i.cz/general/job-priority/.

3 Experiment Setup

This paper follows the experimental setup presented in [16] to evaluate the developed workflow schedules under incomplete performance database. For the makespan and cost evaluation, the Tetrisator simulator worked with a 54 node cluster. The validation of the final schedules was performed on the Barbora cluster, where a static allocation was created to ensure the same initial conditions for all tests.

3.1 Investigated Workflows

This paper uses two typical biomedical ultrasound workflows applied in the ultrasound neurostimulation and photoacoustic imaging, see Fig. 5. Both workflows consist of two types of tasks. The simulation tasks (ST) executing the k-Wave MPI solver represent heavy parallel jobs running for a few hours. The ST tasks were limited to use between 1 and 32 nodes (36 - 1152 cores). The data processing tasks (PT) perform data pre-processing, post-processing, aggregation, etc. The PT tasks have a linear time complexity and almost perfect scaling. Since their runtime is on the order of minutes, only one or two nodes depending on the amount of memory requested are used.

The first workflow starts with a single PT task generating input files for the ST tasks. Consequently, a few independent trains of ST-PT-ST tasks are executed. Finally, the results from all trains are aggregated using a parallel reduction tree composed of PT tasks. The second workflow starts by running a few ST tasks operating on the same input file, but with different parameters. The results are aggregated into a single output file using a parallel tree reduction. But this time, the result is used by the following wave of ST tasks. In practise, this workflow is repeated in a loop until some error metric calculated by the last PT task is satisfied.

3.2 Used Datasets

Let us here define the datasets used in our experiments along with their short description:

- **Dataset A**. Reference strong scaling of the k-Wave code measured on a domain size of $1024 \times 1024 \times 1024$ grid points using 1–32 nodes.
- **Dataset 1A**. Based on *Dataset A* but having only 16 values including peaks and values in between them.
- **Dataset 2A**. Based on *Dataset A* but having only 8 values excluding peaks.
- **Dataset B**. Reference strong scaling of the k-Wave code measured on a domain size of $810 \times 810 \times 810$ grid points using 1–32 nodes.
- **Dataset 1B**. $810 \times 810 \times 810$ domain interpolated for 1–32 nodes using the quadratic interpolation from the known domain sizes: $512 \times 512 \times 512$, $648 \times 648 \times 648$, $1024 \times 1024 \times 1024$.

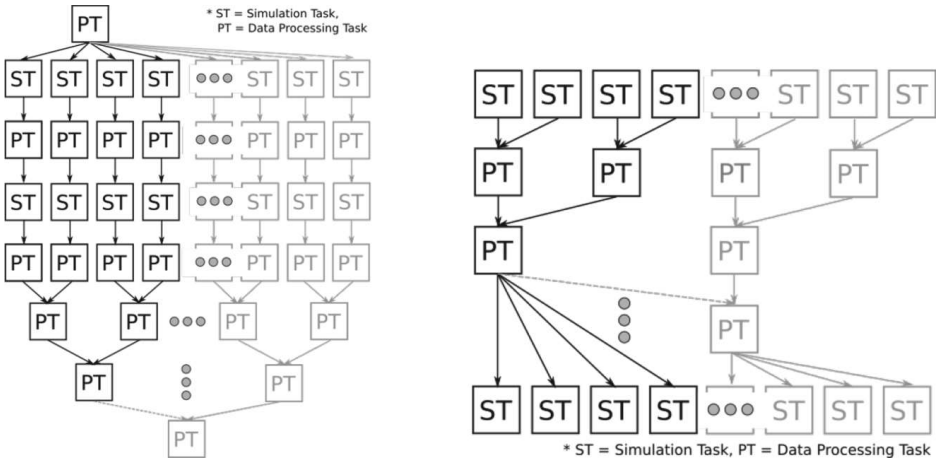


Fig. 5. The structure of investigated workflows. The heavy simulation tasks are interleaved with light data processing tasks. The parts highlighted in black show the minimal workflow structure consisting of 20 and 11 tasks, respectively. The parts displayed in grey show how the workflow structure can grow. (Color figure online)

3.3 Tetrisator Validation Against Real Cluster

To compare the simulator output with the real execution carried out in a dedicated queue comprising 54 nodes of the Barbora cluster, an artificial schedule based on the first workflow type was created. This workflow contained 20 tasks, (8 heavy STs alternated with 12 light PTs). The execution times of particular tasks were taken from the *Dataset A*. The number of simulation time steps inside the ST tasks were reduced to make the workflow finish in less that 1 h. To prevent premature termination, a safety cap of 10% calculated from the estimated execution time was added to each task. The real execution time actually covers net computing time as well as overheads such as the computing node initialization. Performed experimental scenario expects no initial workload, i.e., the cluster was empty when the workflow was submitted and executed. The obtained experimental results are then compared against two evolved execution plans employing Tetrisator with backfilling switched on and off.

3.4 Workflow Schedule Quality Measures

The quality of the developed workflow schedules is evaluated by a fitness function the Optimizer calls after the execution trace has been created by Tetrisator. This work investigates two different fitness functions: GODA and GOSA.

GODA (Global Optimization of the workflow on systems with on-Demand Allocations) calculates the makespan over the longest critical path including queueing times. However, the execution cost considers only truly consumed resources. This is a typical cluster operation with users competing for resources. Since having two contradictory criteria, a user-defined scalarization parameter α is used to balance between the execution time and cost. The algorithm cannot

perform a true multi-objective optimization because there is no further feedback from the user that could select the preferred solution from the Pareto front. Contrary, the most suitable solution has to be chosen autonomously and submitted to the cluster as soon as possible (before the cluster background workload changes significantly).

GOSA (Global Optimization of the workflow on systems with Static Allocations) expects the user holds a dedicated part of the cluster and thus has to pay for the whole allocation no matter some nodes may remain idle. Although this is a more expensive solution, it usually reduces the queueing time. Since the makespan and cost are directly proportional, no scalarization coefficient is needed and only the makespan is considered.

3.5 Evaluation of Interpolation Techniques

To estimate missing execution time for a particular task, domain size, and number of nodes, two different interpolation techniques from the Python's *scipy* package [25] were used. After a thorough investigation in [17] and new experiments performed in the paper, a linear and quadratic `interp1d` interpolations were chosen. Very similar results to the quadratic interpolation were also obtained by cubic spline `CubicSpline` with the `bc_type` parameter set to `natural`. Unfortunately, the use of the default value of `bc_type` caused high oscillations and strong underestimations of the execution time. Therefore, we decided to use a quadratic interpolation instead.

Three different experiments with the interpolation functions were conducted. The goals of particular experiments were

- to estimate missing points on the strong scaling curve for a domain size of 1024^3 grid points defined by the points with ideal scaling ($N\%(P*36) \approx 0$), where N is the domain size and P is the number of nodes, see Fig. 8.
- to estimate missing points on the strong scaling curve for a domain size of 1024^3 grid points when having also points in the middle of the intervals between two points with ideal scaling, see Fig. 8.
- to reconstruct a completely unknown scaling curve for an unseen domain size from the data stored in the performance database. In this example, scaling curves for 512^3, 648^3 and 1024^3 were used to estimate the one for 810^3 grid points, see Fig. 9. The domain sizes chosen progressively double the total number of grid points.

As the measure of the interpolation quality, a mean relative error was used, see Eq. (1).

$$meanError = \frac{1}{N}\sum_{i=0}^{N}\left(\frac{|a_i - b_i|}{a_i}\right) \tag{1}$$

where a denotes the measured execution time, b the interpolated execution time, and N is the total number of the compute nodes (32).

In all cases, we can tolerate a small overestimation but shall avoid underestimation which leads to premature job termination and necessary resubmission with prolonged execution time.

4 Experimental Results

This section presents and discusses (1) the similarity of the workflow execution schedule to the one executed on a real HPC cluster, and (2) the error reached by the interpolation techniques.

4.1 Simulated Execution Plans Reliability

The following figures point out the differences between simulated execution plans created by Tetrisator and the real executions performed in the dedicated queue on Barbora. Figure 6 shows the scenario where no initial workload is expected and all 54 nodes are fully available at submission time. As expected, the simulated makespan by Tetrisator with backfilling switched off is a bit pessimistic causing the overestimation by 15%. On the contrary, it can be seen that the simulated makespan by Tetrisator using backfilling is underestimated by 3%. This underestimation is, however, caused by cumulative error produced by slight delays of individual task execution times on the cluster.

Our observations suggest that the real PBS cluster scheduler works in the same manner as Tetrisator. This means the tasks within the workflow are submitted to the real cluster in the same order as they are processed by the Tetrisator, and their submission time is more or less the same. Thus, the tasks are also executed one by one in the same manner as arriving to the cluster. The changes in the order happen when a task has to wait for free resources (Fig. 7).

4.2 Interpolation Functions Accuracy

Figure 8 shows the measured and interpolated strong scaling curves on a domain composed of 1024^3 grid points. Inspecting the scaling curve created by a linear interpolation, a very close match can be seen. When interpolating using values where the scaling is close to the optimal, the mean interpolation error reaches 4%. After adding the values from the middle of particular intervals, the error drops below 0.8%. Unfortunately, the interpolated values for sparser training data are mostly underestimated, which can be corrected by a small bias or picking the points with the worst instead of best workload distribution.

When repeating the same experiment with a cubic spline and a quadratic interpolation, the mean error gets higher up to the level of 12% and 7%, respectively, depending on the number of known values. The high error is caused by several oscillations, and more specifically, by the extrapolation error where the execution time is extremely underestimated.

The 4% error of the linear interpolation reaches the level of uncertainty of real execution time measurement on clusters due to unstable node, network and I/O performance. The suitability of the linear interpolation can be also attributed to a very good scaling of the ST tasks without any significant anomalies. Since parallel codes have to show good scaling to be deployed in production runs, linear interpolation is expected to work well for most such codes.

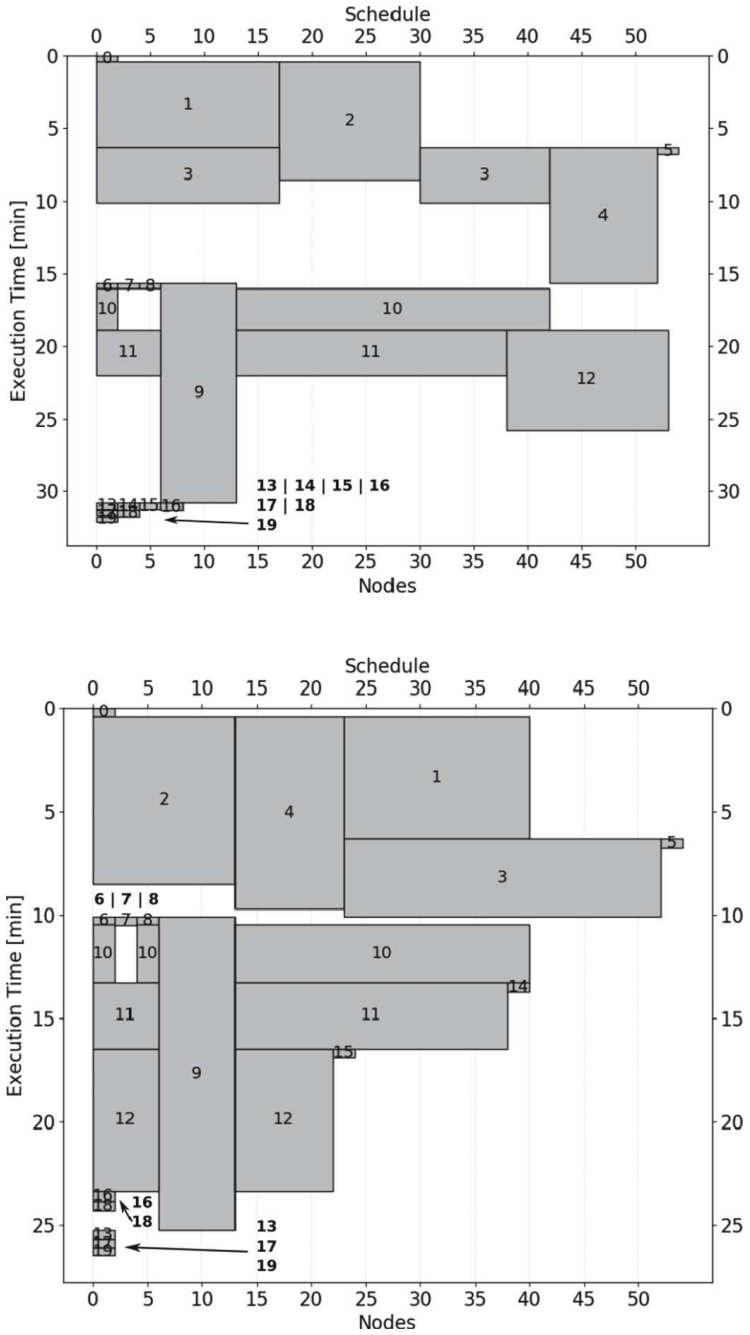


Fig. 6. Two simulated execution schedules. The top one with backfilling switched-off and the makespan reaches 32.1 min, and the bottom one implementing backfilling and finishing earlier in 26.4 min.

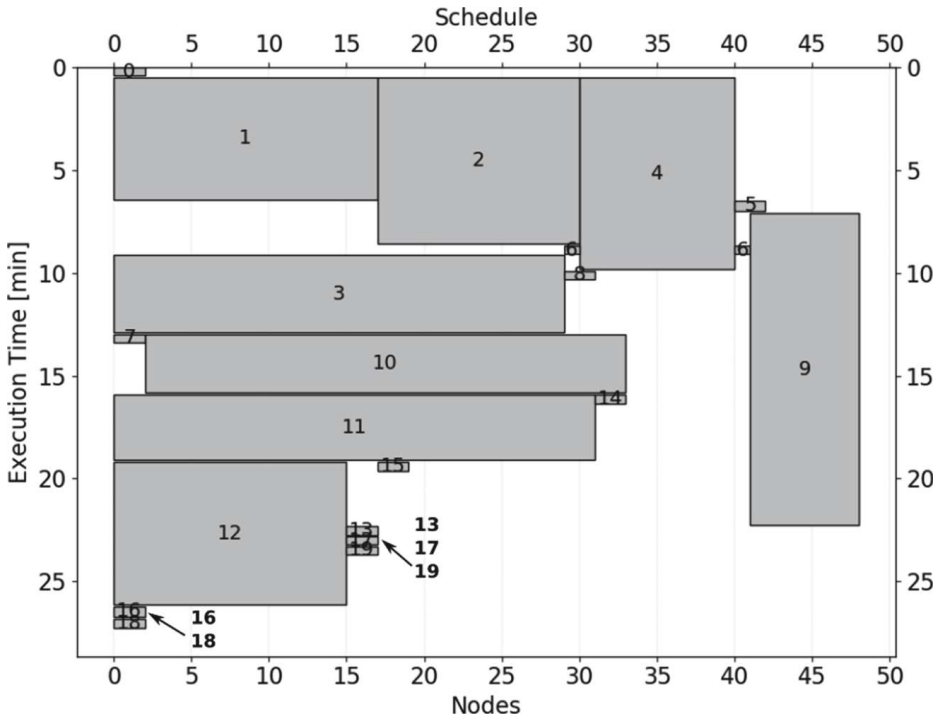


Fig. 7. Real execution of the workflow on Barbora finishing in 27.3 min.

The second experiment attempts to estimate the strong scaling for an unknown domain size, see Fig. 9. The figure reveals that the interpolation method rather overestimate the scaling curve. When repeating this experiment with a linear and a natural cubic spline interpolations, we got the mean error of 25.4% and 13.5%, respectively, while the quadratic interpolation and the cubic spline with `bc_type` parameter set to default produced better estimates reaching the mean error at a level of 10.5%. The explanation is quite simple. While the strong scaling of the ST tasks on a given domain size is almost linear, the algorithm has an asymptotic time complexity of $O(n \log n)$. Moreover, the ST tasks heavily employ fast Fourier transform which is very sensitive to the domain size and its prime factors. The quadratic interpolation thus better capture the nature of ST tasks.

The conclusion is to use a linear interpolation to estimate values on known scaling curves while using a quadratic interpolation when the domain size has not been seen before. It is important to say that the k-Wave code is highly tuned and scales very well. Employing a code the scaling of which is more "wild" with many peaks or a dramatic slowdowns may become a challenge. When using different

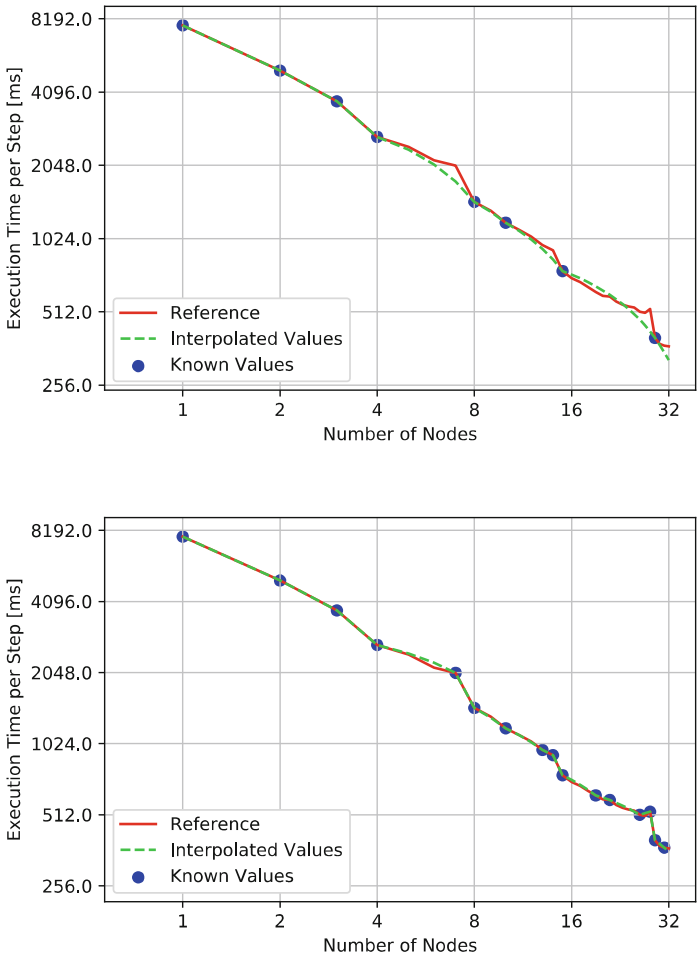


Fig. 8. Reference and interpolated strong scaling of ST tasks for a domain size of 1024^3 grid points with a linear interpolation calculated from 8 and 16 known values, respectively. In the top figure, values in unexpected peaks were selected intentionally to see how much the value would be underestimated.

parallel codes, it may be beneficial to use a different interpolation for unknown domain sizes corresponding to the asymptotic time complexity. Moreover, if the scaling is relatively stable, it may be possible to construct a scaling equation and use a fitting methods to set its coefficients using known performance data. Alternatively, we may try to interpolate the known points using a various polynomial interpolations and based on the error make a decision about a selection of the interpolation method.

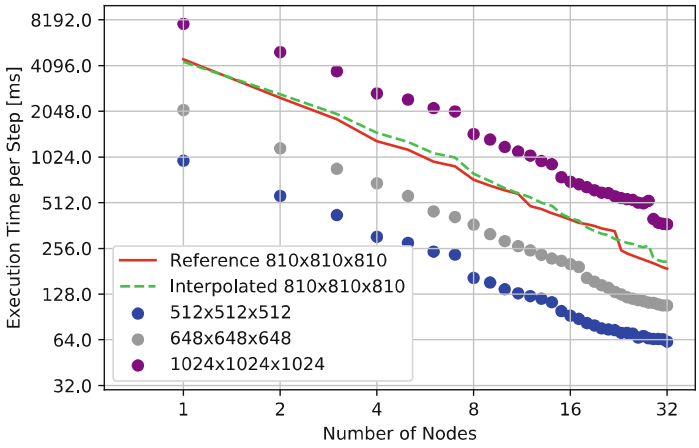


Fig. 9. Reference and interpolated strong scaling of ST tasks for an unknown domain size of 810^3 grid points with a quadratic interpolation.

4.3 Impact of Interpolation on Schedule Makespan and Cost

This section investigates the quality and accuracy of the developed schedules when using the performance database containing all data, only a subset, or no data for particular domain size.

Figure 10 shows the makespan and cost of the best workflow schedules developed for the GODA situation on a known domain size of 1024^3 grid points, with all, 8 and 16 performance values. These experiments also use different values of the α scalarization coefficient (only three values of α are used in figure for better visibility). The schedules were collected over twenty independent runs of the genetic algorithm. The Pareto fronts (lines in the plot) for the same values of α are close to each other confirming that by employing interpolation methods on incomplete datasets we are able to achieve very similar results. When using *Dataset 2A* containing only 8 performance values, the solutions found may be deflected from that ones evolved using dense dataset. This actually does not mean that found solutions are bad, they just overlap the area where solutions for different value of α would be expected. Next, it can be seen that solutions for different α form isolated clusters. This implies we can affect the execution plan to prioritize different criteria. At this point, it is important to note that the execution plan may be adjusted in makespan by a factor of 10.0 while in computational cost by a factor of 1.7. The factors vary and the cost factor is such small due to the highly optimised code used. This is a very promising result showing that when the interpolation is reasonably accurate, the impact on the best solution developed by the Optimizer is rather small.

Table 1 summarizes conducted experiments of GOSA expressing the quality of the execution schedules as makespan. The table may be divided into two parts. The left one is for the domain size of 1024^3 where missing strong scaling values were completed by a linear interpolation. The right one is for the domain size of 810^3 which was fully interpolated using a quadratic interpolation. The

difference between the achieved makespan for the full performance dataset and interpolated datasets is given by an interpolation error (investigated in Sec. 4.2) and performance fluctuations of cluster's nodes. The experiments were provided with both backfilling switch on and off. It turned out backfilling did not impact the results significantly, causing differences between 0.09% and 4%. This suggests the genetic algorithm finds such good workflow schedules that minimize the amount of unused resources so that the backfilling has only a limited space for schedule improvements. Next, a workflow structure also influences how good the workflow could be mapped.

Table 1. The results show GOSA applied on the domain of 1024^3 on the left and 810^3 on the right. Experiments were performed using (1) the full performance dataset without interpolation, (2) the partial performance dataset of 8 and 16 known values, respectively, and completed using linear interpolation, and (3) the full performance dataset created using quadratic interpolation. The table depicts average (Avg), minimum (Min) and maximum (Max) obtained values of makespan in minutes. The percentage difference between experiments with partial and full performance datasets is also depicted.

1024 x 1024 x 1024		40 Tasks		80 Tasks		810 x 810 x 810		40 Tasks		80 Tasks	
		Makespan [min]	Diff. [%]	Makespan [min]	Diff. [%]			Makespan [min]	Diff. [%]	Makespan [min]	Diff. [%]
GOSA	**Avg**	**29.70**	**-**	**58.31**	**-**	**GOSA**	**Avg**	**14.82**	**-**	**30.05**	**-**
with no	**Min**	27.75	-	55.74	-	**with no**	**Min**	14.07	-	28.32	-
interp.	**Max**	35.10	**-**	61.07	**-**	**interp.**	**Max**	16.88	**-**	31.76	**-**
GOSA with	**Avg**	**29.19**	**1.72**	**59.23**	**1.57**	**GOSA**	**Avg**	**17.08**	**15.25**	**33.11**	**10.18**
linear interp.	**Min**	27.29	1.65	55.27	0.84	**with quadratic**	**Min**	15.44	9.70	31.27	10.41
(Dataset A1)	**Max**	33.25	5.27	65.47	7.21	**interpolation**	**Max**	18.85	11.64	36.67	15.44
GOSA with	**Avg**	**26.74**	**9.98**	**51.06**	**12.44**						
linear interp.	**Min**	24.87	10.36	49.05	12.00						
(Dataset A2)	**Max**	30.33	13.58	56.46	7.55						

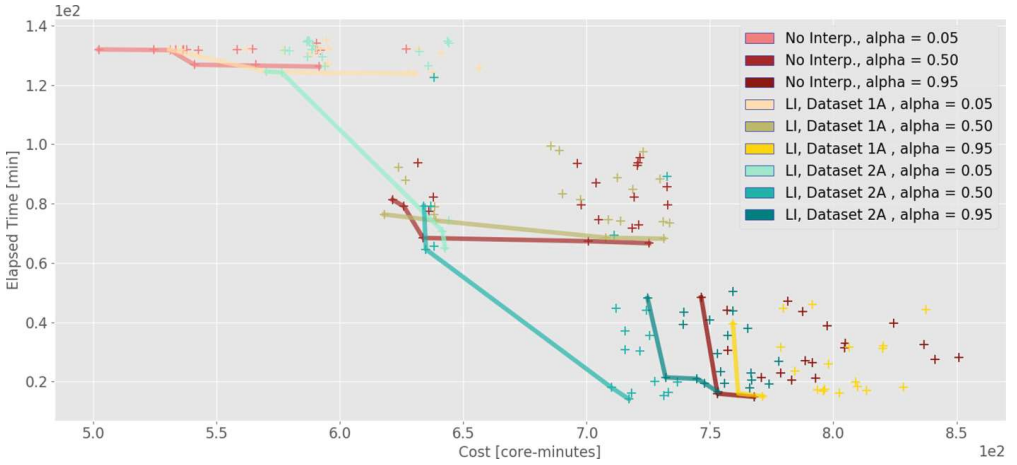


Fig. 10. Pareto front together with dominated solutions showing the evolved schedules for workflows of 11 tasks not requiring interpolation, and two experiments both using linear interpolation (LI) but differing in the content of the performance dataset.

5 Conclusions

The paper has investigated the optimization of moldable ultrasound workflow executions under incomplete performance database where the execution times for some combination of tasks, input data and amount of resources are not known and have to be estimated. Consequently, the paper has proven the workflow execution on a cluster can be simulated and this simulator can be integrated in the k-Dispatch's optimization module. Although being a one-pass PBS-based simulator, the estimations provided are sensible. The simulator gives accurate estimations especially for workflows executed on dedicated resources where other workload is known. The cross validation of an artificial and the real schedules created by the PBS job scheduler on Barbora show a good general match.

The experimental results indicate that linear interpolation works well in situations the input data has been seen before and the task has already been executed using a few execution parameters configurations. In such cases, the missing performance data can be calculated with a very small error below 4%. From our experience, linear interpolations appear to be generally applicable on parallel codes with good strong scaling. On the other hand, if the input data has not been seen before, the execution time has to be estimated from similar inputs by interpolating between known strong/weak scaling curves. In this case, a quadratic interpolation worked sufficiently well for our codes, however, the error may reach 10%. This can be attributed to used codes having $O(N \log N)$ time complexity. For codes with different time complexity, higher polynomial interpolations may produce better results.

The paper also confirms that it is possible to find different schedules that prioritize various criteria using the trade-off parameter α. The proposed optimization algorithm constructs the Pareto front offering different suitable schedules. Users, however, (1) are not aware of what tasks are executed within the workflow, (2) may not know what solution to choose, and finally (3) the Pareto fronts are calculated just before the workflow execution and this information is not available at submission time to k-Dispatch. This is the reason why a multi-criteria optimization is transformed to an easier form where users can express their preferences between two criteria (makespan vs. computational cost) using, e.g., a slider bar just before workflow submission to k-Dispatch.

The developed schedules tend to overestimate the execution time, which is partially caused by imperfect interpolation, and a reserve of 10% added to the workflow to avoid premature termination. Nevertheless, the error between developed and real schedules fits within a 15% margin, which is considered to be acceptable for most users.

5.1 Future Work

There are two directions we would like to follow in our future work. First, we would like to include the information about the actual cluster utilization into the cluster simulator. This will allow us to better simulate workflow execution in on-demand allocations where the user competes with others. It may have an impact

on the shape of the developed schedules because tasks asking for more resources sit longer in the queue. Using smaller amounts of resources thus may improve the workflow makespan. Second, we would like to examine more advanced machine learning techniques to improve the interpolation accuracy once the performance database includes tens of thousands of records.

Acknowledgments. This work was supported by the Ministry of Education, Youth and Sports of the Czech Republic through the e-INFRA CZ (ID:90140). This work was supported by Brno University of Technology under project numbers IGA FIT/FSI-J-22-7980 Acceleration of Selected Evolutionary Communication Techniques for Solving Combinatoric Tasks and FIT-S-20-6309 Design, Optimization and Evaluation of Application Specific Computer Systems.

References

1. Amdahl, G.M.: Validity of the single processor approach to achieving large scale computing capabilities. In: Proceedings of the April 1820 1967 Spring Joint Computer Conference, vol. 23(4), pp. 483–485 (1967)
2. Bharathi, S., Chervenak, A., Deelman, E., Mehta, G., Su, M.-H., Vahi, K.: Characterization of scientific workflows. In: Third Workshop on Workflows in Support of Large-Scale Science, pp. 1–10. IEEE (2008)
3. Bleuse, R., Hunold, S., Kedad-Sidhoum, S., Monna, F., Mounie, G., Trystram, D.: Scheduling independent moldable tasks on multi-cores with GPUs. IEEE Trans. Parallel Distrib. Syst. **28**(9), 2689–2702 (2017)
4. Chan, T.F., Mathew, T.P.: Domain decomposition algorithms. Acta Numer **3**, 61–143 (1994)
5. Chirkin, A.M., et al.: Execution time estimation for workflow scheduling. Future Generat. Comput. Syst. **75** (2017)
6. Deelman, E., Vahi, K., Juve, G., et al.: Pegasus: a workflow management system for science automation. Future Generat. Comput. Syst. (2014)
7. Dutot, P.-F., Netto, M.A.S., Goldman, A., Kon, F.: Scheduling moldable BSP tasks. In: Feitelson, D., Frachtenberg, E., Rudolph, L., Schwiegelshohn, U. (eds.) JSSPP 2005. LNCS, vol. 3834, pp. 157–172. Springer, Heidelberg (2005). https://doi.org/10.1007/11605300_8
8. Feitelson, D.G., Rudolph, L.: Toward convergence in job schedulers for parallel supercomputers. In: Feitelson, D.G., Rudolph, L. (eds.) JSSPP 1996. LNCS, vol. 1162, pp. 1–26. Springer, Heidelberg (1996). https://doi.org/10.1007/BFb0022284
9. Gad, A.F.: Geneticalgorithmpython: Building genetic algorithm in python (2021)
10. Goldberg, D.E.: Genetic algorithms in search, optimization, and machine learning. Addison-Wesley Longman, Boston (1989)
11. Henderson, R.L.: Job scheduling under the portable batch system. In: Feitelson, D.G., Rudolph, L. (eds.) JSSPP 1995. LNCS, vol. 949, pp. 279–294. Springer, Heidelberg (1995). https://doi.org/10.1007/3-540-60153-8_34
12. Hovestadt, M., Kao, O., Keller, A., Streit, A.: Scheduling in HPC resource management systems: queuing vs. planning. In: Feitelson, D., Rudolph, L., Schwiegelshohn, U. (eds.) JSSPP 2003. LNCS, vol. 2862, pp. 1–20. Springer, Heidelberg (2003). https://doi.org/10.1007/10968987_1
13. Izadkhah, H.: Learning based genetic algorithm for task graph scheduling. Applied Computational Intelligence and Soft Computing (2019)

14. Jansen, K., Land, F.: Scheduling Monotone Moldable Jobs in Linear Time. In: 2018 IEEE International Parallel and Distributed Processing Symposium (IPDPS), pp. 172–181. IEEE, May 2018
15. Jaros, J., Rendell, A.P., Treeby, B.E.: Full-wave nonlinear ultrasound simulation on distributed clusters with applications in high-intensity focused ultrasound. Int. J. High Perfor. Comput. Appli. **30**(2), 137–155 (2016)
16. Jaros, M., Jaros, J.: Performance-cost optimization of moldable scientific workflows. In: Klusáček, D., Cirne, W., Rodrigo, G.P. (eds.) JSSPP 2021. LNCS, vol. 12985, pp. 149–167. Springer, Cham (2021). https://doi.org/10.1007/978-3-030-88224-2_8
17. Jaros, M., Sasak, T., Treeby, B.E., Jaros, J.: Estimation of execution parameters for k-Wave simulations. In: Kozubek, T., Arbenz, P., Jaroš, J., Říha, L., Šístek, J., Tichý, P. (eds.) HPCSE 2019. LNCS, vol. 12456, pp. 116–134. Springer, Cham (2021). https://doi.org/10.1007/978-3-030-67077-1_7
18. Jaros, M., Treeby, E.B., Jaros, J., Georgiou, P.: k-dispatch: A workflow management system for the automated execution of biomedical ultrasound simulations on remote computing resources. In: Platform for Advanced Scientific Computing Conference, pp. 1–10. ACM (2020)
19. Omara, F.A., Arafa, M.M.: Genetic algorithms for task scheduling problem. J. Paral. Distrib. Comput. **70**(1), 13–22 (2010)
20. Poudel, J., Lou, Y., Anastasio, M.A.: A survey of computational frameworks for solving the acoustic inverse problem in three-dimensional photoacoustic computed tomography. Phys. Med. Biol. (2019)
21. Rajaei, H., Dadfar, M.: Comparison Of backfilling algorithms for job scheduling in distributed memory parallel system. In: 2006 Annual Conference and Exposition Proceedings, ASEE Conferences, pp. 11.339.1-11.339.12 (2007)
22. Robert, Y.: Task Graph Scheduling. In: Encyclopedia of Parallel Computing, pp. 2013–2025 (2011)
23. Szabo, T.L.: Diagnostic Ultrasound Imaging: Inside Out (2014)
24. Treeby, B., Cox, B.: k-wave: Matlab toolbox for the simulation and reconstruction of photoacoustic wave fields. J. Biomed. Opt. **15**, 021314 (2010)
25. Virtanen, P., Gommers, R., Oliphant, T. E. A. O.: Fundamental algorithms for scientific computing in python. SciPy 1.0. Nat. Methods **17**, 261–272 (2020)
26. Wolstencroft, K., Haines, R., Fellows, D., et al.: The taverna workflow suite: designing and executing workflows of web services on the desktop, web or in the cloud. Nucleic Acids Res. **41**(W1), W557–W561 (2013)
27. Ye, D., Chen, D.Z., Zhang, G.: Online scheduling of moldable parallel tasks. J. Sched. **21**(6), 647–654 (2018). https://doi.org/10.1007/s10951-018-0556-2
28. Yoo, A.B., Jette, M.A., Grondona, M.: SLURM: simple linux utility for resource management. In: Feitelson, D., Rudolph, L., Schwiegelshohn, U. (eds.) JSSPP 2003. LNCS, vol. 2862, pp. 44–60. Springer, Heidelberg (2003). https://doi.org/10.1007/10968987_3