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.

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. . k-Dispatch: A Workflow Management System for the Automated Execution of Biomedical Ultrasound Simulations on Remote Computing Resources. In *Proceedings of* . ACM, New York, NY, USA, 10 pages.

*corresponding author.

,
2020.

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

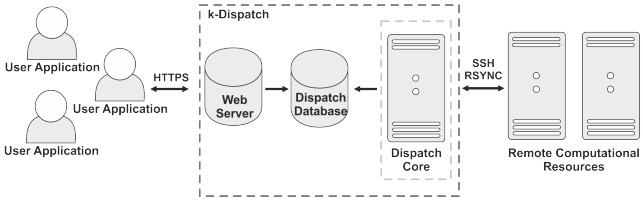


Figure 2: Simplified architecture of k-Dispatch showing three basic modules and their connection to user applications and computational resources.

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.

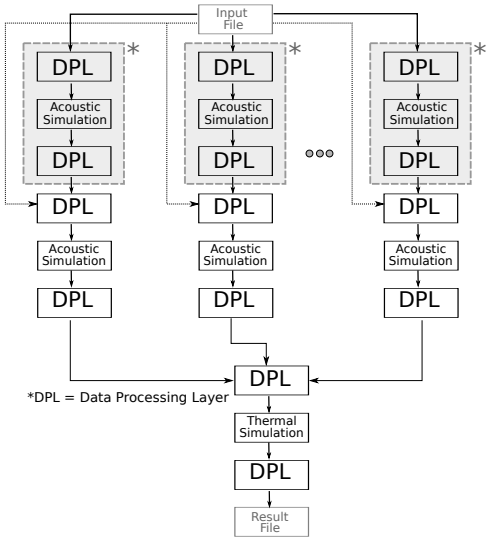
The architecture of k-Dispatch is generic and modular to enable easy system extensions by adding new workflows, computational resources, interfaces to different job schedulers, etc. Currently, k-Dispatch supports several predefined workflows hard-coded in the structure of the input file and parsing Python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

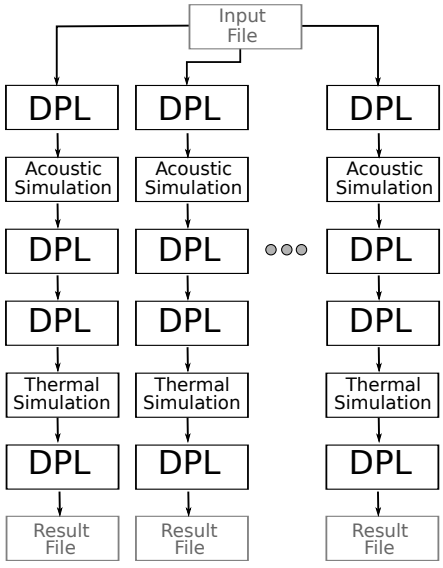
The Web server module is based on the Python Flask technology [39] and represents the only entry point to k-Dispatch. The web server communicates with the Dispatch database and with the local storage. The input files with new workflows to compute are stored in the local storage and a new record is made 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 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



(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.

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.

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

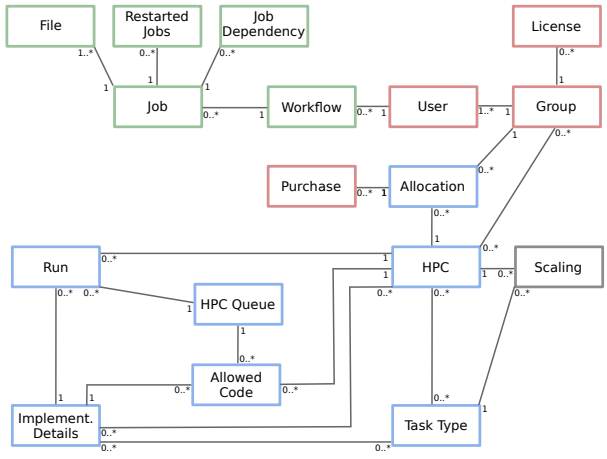


Figure 3: Simplified entity relationship diagram (ERD) for the Dispatch database.

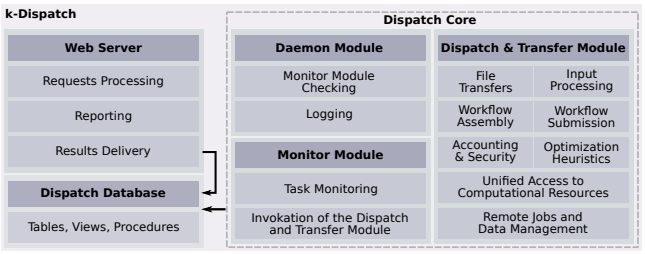


Figure 4: The architecture of k-Dispatch composed of three modules and numerous submodules.

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,
- 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.

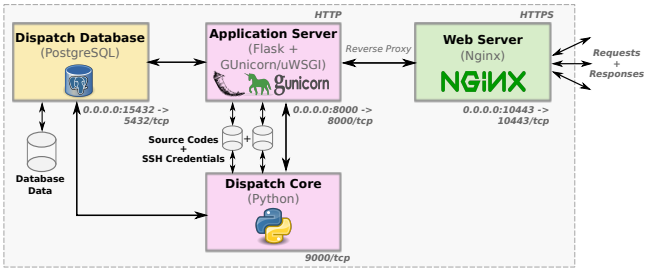


Figure 5: k-Dispatch architecture wrapped into Docker containers and volumes. The text descriptions below individual rectangles show network port mappings used in the current solution.

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.

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 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

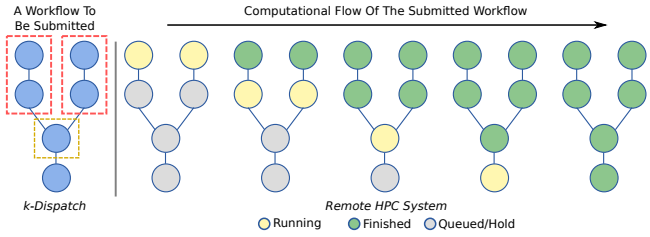


Figure 6: k-Dispatch's execution model. The blue workflow on the left hand side reveals the concurrency and dependencies between tasks. Subgraphs in the red rectangles can be executed concurrently since there are no dependencies between them, however, the tasks inside them have to be executed sequentially. 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.

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

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].

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

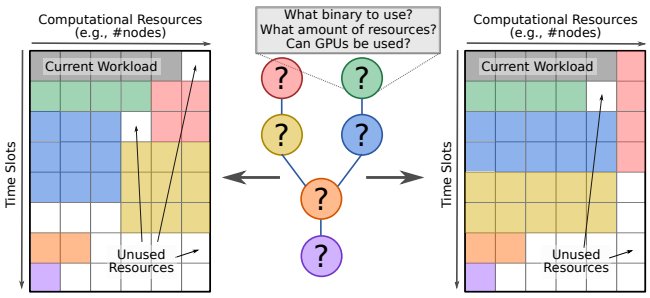


Figure 7: Two examples of the workflow mapping to computational resources and time slots under different execution parameters. Colors show resources occupied by particular tasks. Both mappings take 7 time units to complete. The mapping on the left-hand side is, however, a bit cheaper since it consumes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

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 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

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

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 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 Klusacek, 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.