Optimizing Biomedical Ultrasound Workflow Scheduling Using Cluster Simulations

Marta Jaros[1][0000−0002−7775−8106] and Dalibor Klusáček[2] and Jiri Jaros[1][0000−0002−0087−8804]

[1] Brno University of Technology,
Faculty of Information Technology,
Centre of Excellence IT4Innovations,
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 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 Section 2, the considered workflow scheduling problem is discussed thoroughly. Section 3 describes the Alea simulator and its new workflow-related functionality. Next, Section 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 Section 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 2.1). 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. 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

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]

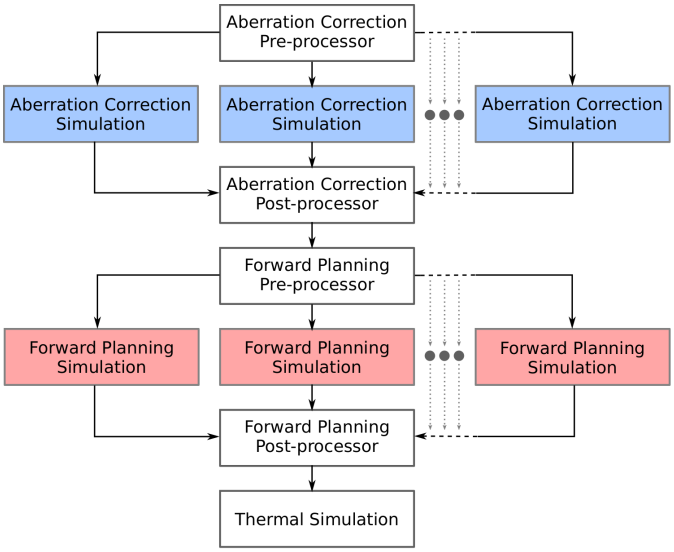


Fig. 1. Typical neurostimulation simulation workflow using the reverse focusing for aberration corrections. After pre-processing, reverse ultrasound propagation simulation from particular targets are executed. After aberration correction, forward ultrasound simulations are executed to calculate energy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

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 Gbit 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

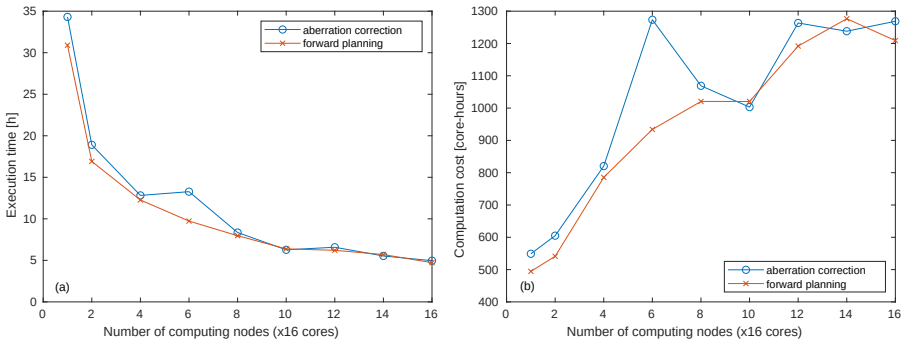


Fig. 2. Strong scaling of the (a) execution time and (b) execution cost for aberration correction and forward planning simulations. The anomalies in the plots are caused by unbalanced work distribution over compute nodes.

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 minutes. 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.

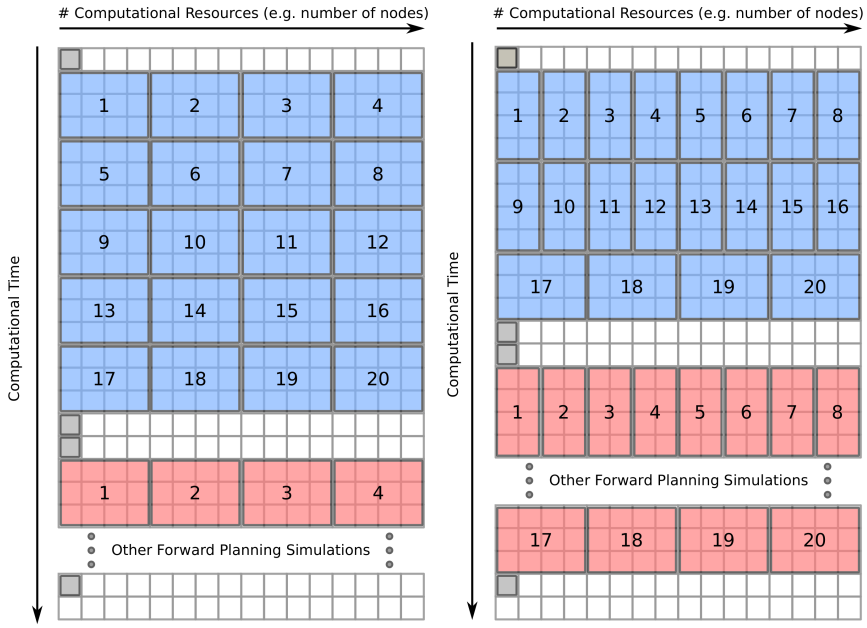


Fig. 3. Example of two different execution plans of the neurostimulation workflow on a 16-node cluster. On the left, every job was optimized independently neglecting the queuing times. On the right, the complete workflow was optimized leading to different resources allocations for particular simulations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

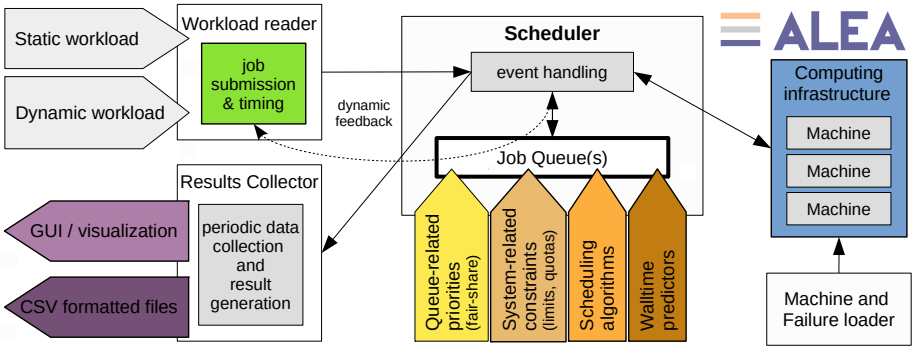


Fig. 4. Main components of the Alea jobs scheduling simulator.

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 Section 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

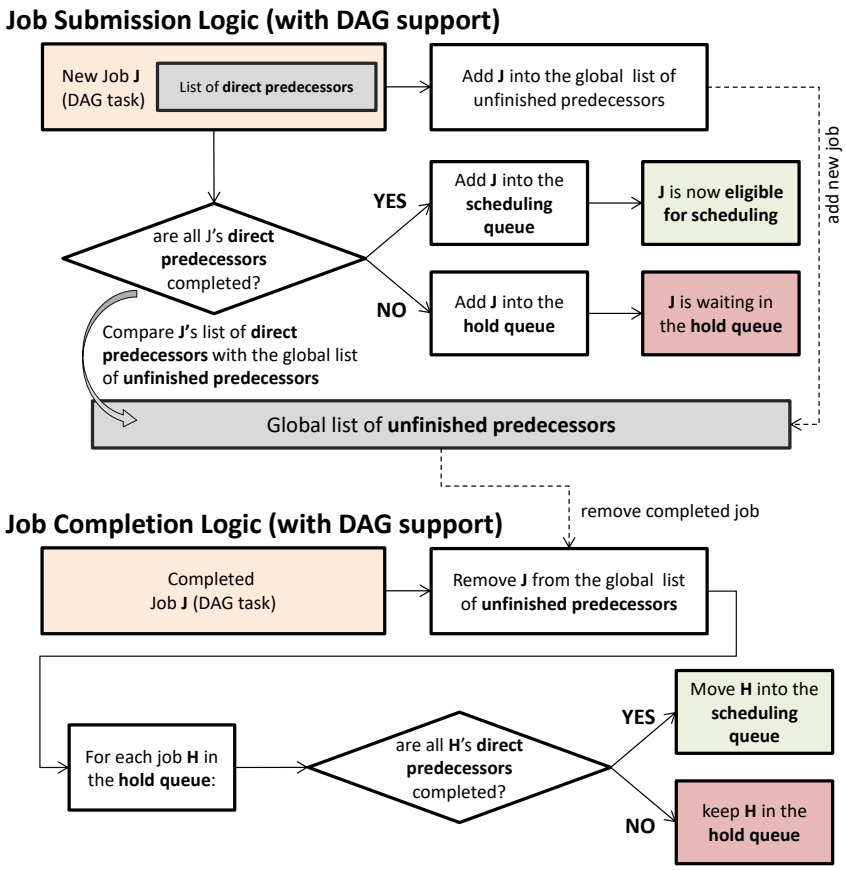


Fig. 5. Added logic handling correct execution order of DAG-like workflows within Alea simulator. Job dependencies are checked during new job arrival (top) and updated once a job completes its execution (bottom). At this point, waiting jobs from the *hold queue* are moved to the *scheduling queue* if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

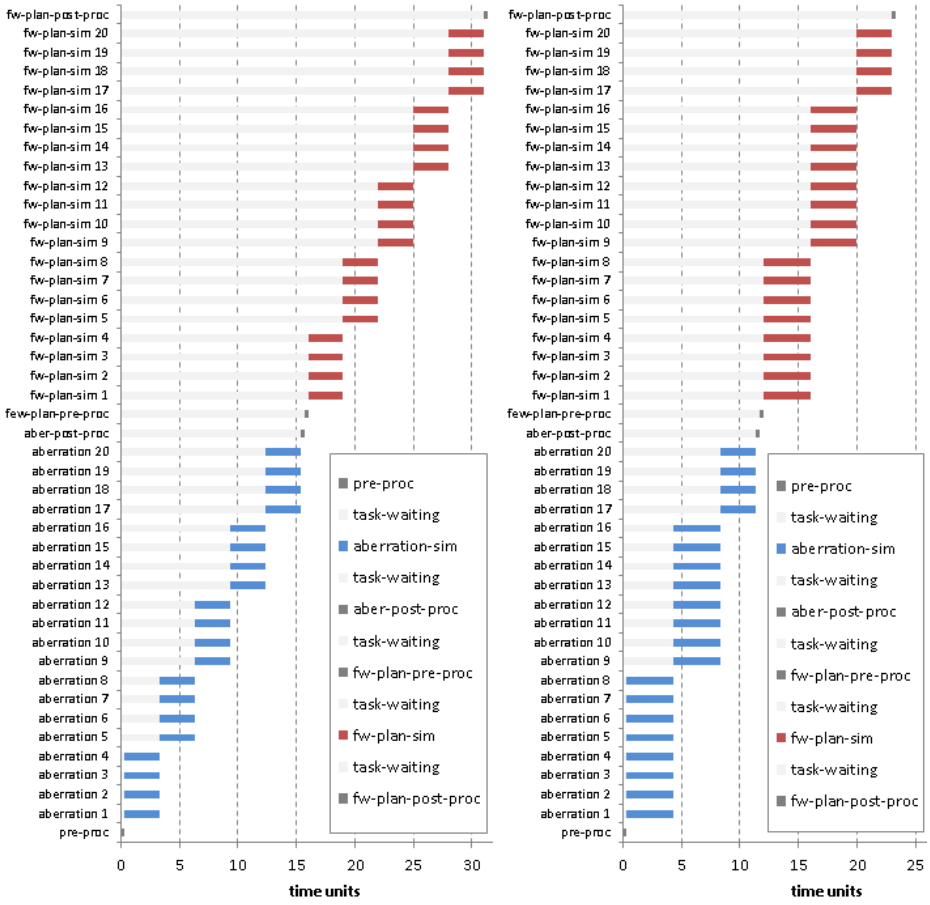


Fig. 6. Alea simulator used to measure the impact of task-level (left) vs. workflow-level (right) optimization on the total workflow execution time (makespan).

time (X-axis). Clearly, these results correspond to the illustration used in Fig. 3. 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

[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.

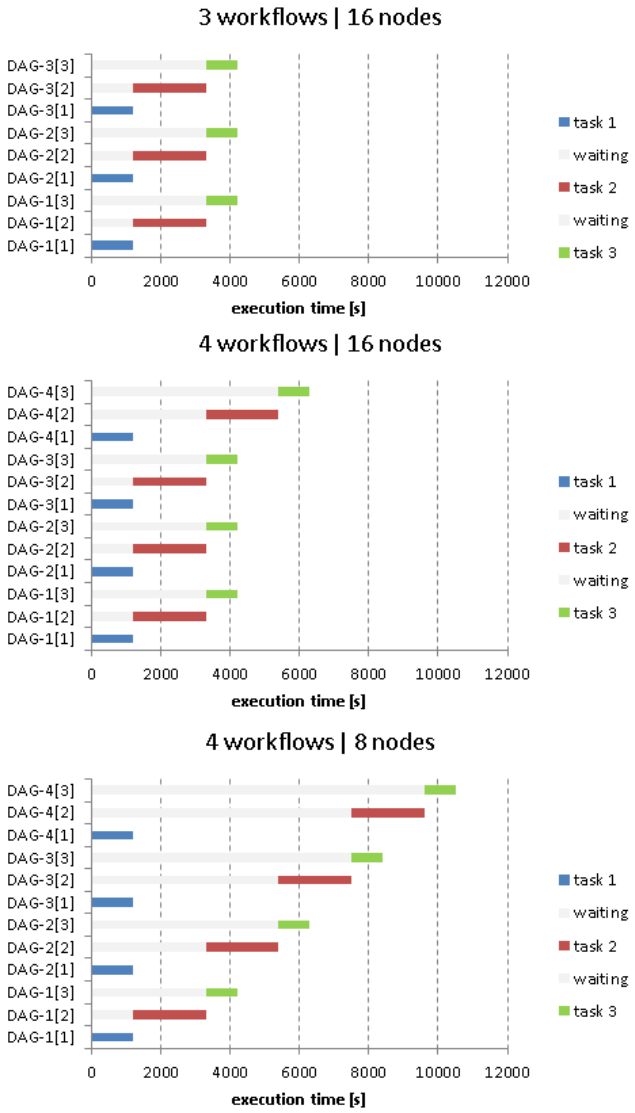


Fig. 7. Makespan and wait time (queuing time) as impacted by the number of concurrently executed workflows and the size of the infrastructure.

the most demanding setup (64 DAGs, each having 64 tasks per DAG) the total simulation time is below 2.5 seconds.

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.

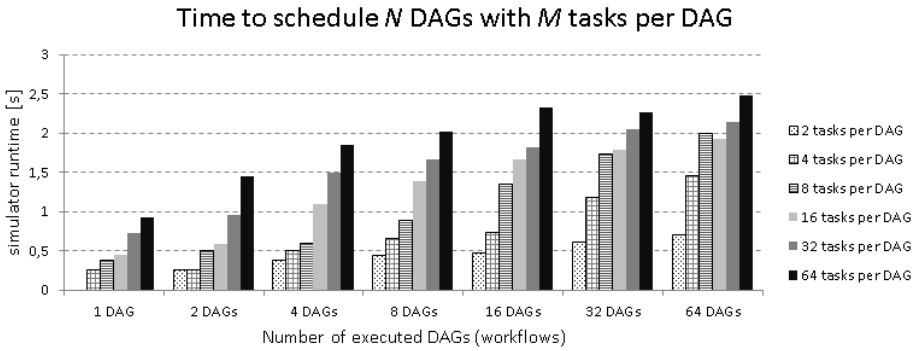


Fig. 8. The time needed to execute one simulation with respect to the number and complexity of simulated workflows (DAGs).

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 Edu-

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

cation, Youth and Sports of the Czech Republic. Computational resources were supplied by 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 IT4-Innovations 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 Unions 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. F. Azevedo, D. Klusáček, and F. Suter. Improving fairness in a large scale HTC system through workload analysis and simulation. In R. Yahyapour, editor, *Euro-Par 2019: Parallel Processing*, pages 129–141. Springer International Publishing, 2019.
3. E. Deelman, K. Vahi, G. Juve, M. Rynge, S. Callaghan, P. J. Maechling, R. Mayani, W. Chen, R. Ferreira da Silva, M. Livny, and K. Wenger. Pegasus: a workflow management system for science automation. *Future Generation Computer Systems*, 2014.
4. D. G. Feitelson. Parallel workloads archive, May 2020. `http://www.cs.huji.ac.il/labs/parallel/workload/`.
5. D. G. Feitelson. The standard workload format, May 2020. `https://www.cse.huji.ac.il/labs/parallel/workload/swf.html`.
6. P. S. Georgiou, J. Jaros, H. Payne, C. Allen, T. T. Shah, H. U. Ahmed, E. Gibson, D. Barratt, and B. E. Treeby. Beam distortion due to gold fiducial markers during salvage high-intensity focused ultrasound in the prostate. *Medical Physics*, 44(2):679–693, 2017.
7. D. Godlove. Singularity. In *Proceedings of the Practice and Experience in Advanced Research Computing on Rise of the Machines (learning)*, pages 1–4, New York, NY, USA, jul 2019. ACM.
8. HTCondor. HTCondor – high throughput computing, 2019.
9. M. Jaros, B. E. Treeby, P. Georgiou, and J. Jaros. 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, New York, NY, USA, 2020. Association for Computing Machinery.
10. D. Klusáček, M. Soysal, and F. Suter. Alea – complex job scheduling simulator. In R. Wyrzykowski, E. Deelman, J. Dongarra, and K. Karczewski, editors, *Parallel Processing and Applied Mathematics*, pages 217–229. Springer International Publishing, 2020.

11. D. Klusáček and Š. Tóth. On interactions among scheduling policies: Finding efficient queue setup using high-resolution simulations. In F. Silva, I. Dutra, and V. S. Costa, editors, *Euro-Par 2014*, volume 8632 of *LNCS*, pages 138–149. Springer, 2014.
12. E. E. Konofagou. Trespassing the Barrier of the Brain with Ultrasound. *Acoustics Today*, 13(4):21–26, 2017.
13. D. Merkel. Docker: Lightweight linux containers for consistent development and deployment. *Linux Journal*, 2014(239), March 2014.
14. Y. Robert. Task graph scheduling. In D. Padua, editor, *Encyclopedia of Parallel Computing*, pages 2013–2025. Springer US, Boston, MA, 2011.
15. V. Sarkar. Partitioning and scheduling parallel programs for multiprocessors. In *Research Monographs in Parallel and Distributed Computing*, pages 1–183. MIT Press, 1989.
16. A. Sulistio, U. Cibej, S. Venugopal, B. Robic, and R. Buyya. A toolkit for modelling and simulating data Grids: an extension to GridSim. *Concurrency and Computation: Practice & Experience*, 20(13):1591–1609, 2008.
17. T. L. Szabo. Diagnostic Ultrasound Imaging: Inside Out, 2014.
18. B. E. Treeby and B. T. Cox. k-Wave: MATLAB toolbox for the simulation and reconstruction of photoacoustic wave fields. *Journal of Biomedical Optics*, 15(2):021–314, 2010.
19. B. E. Treeby, J. Jaros, E. Martin, and B. T. Cox. From biology to bytes: Predicting the path of ultrasound waves through the human body. *Acoustics Today*, 15(2):36–44, 2019.