Optimization of Execution Parameters of Moldable Workflows under Incomplete Performance Data

Author 1
Author 2
author1@email.com
author2@email.com
Institution
City, Country

ABSTRACT

Complex scientific workflows describing challenging real-world problems are composed of many computational tasks requiring high performance computing or cloud facilities to be computed in a sensible time. Most of these tasks are usually written as moldable parallel programs being able to run across various numbers of compute nodes. The amount of resources assigned to particular tasks may strongly affect the overall execution and queuing time of the whole workflow (makespan) as well as the total computational cost.

For this purpose, this paper employs a genetic algorithm that searches for a good resource distribution over the particular tasks, and a cluster simulator that evaluates makespan and cost of the developed workflow execution schedule. Since the exact execution time cannot be measured for every possible combination of task, input data size, and assigned resources, several interpolation techniques are used to predict the task duration for a given amount of compute resources. The best execution schedules are eventually submitted to a real cluster with a PBS scheduler to validate the whole technique.

The experimental results confirm the proposed cluster simulator corresponds to a real PBS job scheduler with a sufficient fidelity. The investigation of the interpolation techniques showed that incomplete performance data can be successfully completed by linear and quadratic interpolations making a maximum mean error below 10%. Finally, the paper shows it is possible to implement a user defined parameter which instructs the genetic algorithm to prefer either the makespan or cost, or find a suitable trade-off.

CCS CONCEPTS

• **Computer systems organization** → **Embedded systems**; *Redundancy*; Robotics; • **Networks** → Network reliability.

PASC '22, June 27–29, 2022, Basel, Switzerland

ACM ISBN 978-1-4503-XXXX-X/18/06…$15.00
https://doi.org/10.1145/1122445.1122456

KEYWORDS

task graph scheduling, workflow, genetic algorithm, moldable tasks, makespan estimation, performance scaling interpolation

ACM Reference Format:
Author 1 and Author 2. 2018. Optimization of Execution Parameters of Moldable Workflows under Incomplete Performance Data. In *PASC '22: Platform for Advanced Scientific Computing , June 27–29, 2022, Basel, Switzerland*. ACM, New York, NY, USA, 10 pages. https://doi.org/10.1145/1122445.1122456

1 INTRODUCTION

All fields of science and engineering use computers to reach new findings, while the most compute power demanding problems require High Performance Computing (HPC) or Cloud systems to give answers to their questions. The problems being solved nowadays are often very complex and comprise a lot of various tasks with mutual dependencies describing different aspects of the investigated problem. Their computation can be formally described using scientific workflows [2], also referred to as task graphs [16]. There is immense number of scientific workflows spread around various fields [14], yet they have one thing in common. They are all desired to be computed in the shortest time for the lowest possible cost.

The execution of scientific workflows on HPC systems is performed via communication with the HPC front-end, also referred to as the job scheduler [11]. After the workflow data has been uploaded to the cluster, the workflow tasks are submitted to the computational queues where being waiting until the system has enough free resources, and all task dependencies have been resolved (predecessor tasks have been finished).

Modern HPC schedulers implement advanced techniques for efficient task and resource management [12]. However, the queuing time, computation time and related cost depend on the task execution parameters provided at submission. These parameters include the required execution time accompanied by the number of compute nodes, cores and accelerators, the amount of main memory and storage space, and more and more frequently, the frequency and power cup of various hardware components. In most cases, only experienced users are endowed by sufficient knowledge to estimate these parameters appropriately knowing the size of the input data for particular tasks. In other cases, default parameters may be chosen leading to inefficient workflow processing.

Complex compute tasks are usually written as moldable distributed programs being able to exploit various amounts and types of computing resources, i.e., they can run on different numbers of compute nodes. However, the moldability is often limited by many factors, the most important of which being the domain decomposition [4] and parallel efficiency (strong scaling) [1]. The goal of the workflow execution optimization is posed as the assignment of suitable amount of compute resources to individual tasks in order to minimize the overall computation time and cost.

While the field of rigid workflow optimization, where the amount of resources per task is fixed or specified by the user before the workflow submission, has been thoroughly studied and is part of common job schedulers such as PBSPro [11] or Slurm [20], the autonomous optimization and scheduling of moldable workflows has still been an outstanding problem, although firstly opened two decades ago in [8].

During the last decade, many papers have focused on the prediction of rigid workflow execution time and enhancing the HPC resource management. For example, Chirkin et al. [5] introduces a makespan estimation algorithm that may be integrated into job schedulers. Robert et al. [16] gives an overview of task graph scheduling algorithms. The usage of genetic algorithms addressing the task scheduling problems has also been introduced, e.g., a task graph scheduling on homogeneous processors using genetic algorithm and local search strategies [13], and performance improvement of the used genetic algorithm [15]. However, a handful works have taken into the consideration the moldability and strong scaling behavior of particular tasks, their dependencies and the current cluster utilization [3, 7, 19].

In all cases, the estimation of the makespan and optimization of the tasks execution parameters rely on the performance database storing strong and weak scaling. However, it is often not possible to benchmark the execution time for all possible combinations of the task type, task inputs and execution parameters. If a task has already been executed with given inputs and execution parameters, the execution time can be retrieved from the performance database. However, for unseen combinations, some kind of interpolation or machine learning techniques have to be used.

In our previous work [DOUBLE-BLIND], Genetic Algorithms (GA) [10] and a simple cluster simulator were used to find optimal execution parameters for various workflows on systems with on-demand and static allocations. This paper follows up with our previous work and its main goals are to (1) prove that GA is able to find execution plans for different workflows when using incomplete performance datasets, (2) prove a trade-off parameter to find different solutions meeting contradictory optimization criteria can be introduced, and finally (3) compare the outcomes from the cluster simulator with the real workflow execution also considering the initial cluster workload. The resilience of the optimization techniques will be investigated on several scenarios and validated against the real workflow makespan measured on the Barbora supercomputer[1].

2 AUTOMATIC OPTIMIZATION OF WORKFLOW EXECUTION PARAMETERS

The selection of suitable execution parameters for workflow tasks plays the crucial role in the optimization and scheduling process aiming to reduce the computational cost and the overall processing time from the workflow submission to the results delivery, also referred to as makespan. A naive selection of the execution parameters often leads to various unpleasant situations such as unnecessarily long waiting times if default execution time and/or high amounts of compute resources were chosen to ensure the task completion without premature termination, task crashes if the amount of compute resources was not sufficient, task hangs up when unsatisfactory amount or resources is requested, and so on.

Even if users have enough experience with applications used within the workflow, they might see it very difficult and tedious to set the execution parameters properly to get good performance. For the less experienced ones, the default values are usually the first and the only choice. Although batch schedulers implement several optimization methods and heuristics to maintain high cluster utilization and low queueing times, bad execution parameters spoil their submission schedules, e.g., when tens of tasks enter the queue asking for 24 hour allocations but actually finishing after an hour.

2.1 k-Dispatch Workflow Management System

Complex scheduling of scientific workflows goes beyond the capabilities of common batch job schedulers which treat tasks independently only paying attention to their dependencies. For the workflow scheduling, a workflow management system sitting in between the end user and the batch job scheduler is required. k-Dispatch [DOUBLE-BLIND] is a Workflow Management System (WMS) [6, 18] allowing the end users to submit complex workflows with associated data via a simple web interface and have them automatically executed on remote HPC facilities. Although oriented on the ultrasound community and the popular k-Wave acoustic toolbox [DOUBLE-BLIND], its general design allows simple adaptation to other workflows and toolboxes.

k-Dispatch consists of three main modules depicted in Fig. 1: Web server, Dispatch database and Dispatch core. The user applications, e.g., a stand-alone medical GUI, Web application, or Matlab interface, communicate with the Web server using the secured HTTPS protocol and REST API. The Dispatch database holds all necessary information about the users, submitted workflows, jobs, computational resources, available binaries and the performance data collected over

[1] IT4Innovations, Czech republic, https://docs.it4i.cz/barbora/introduction/

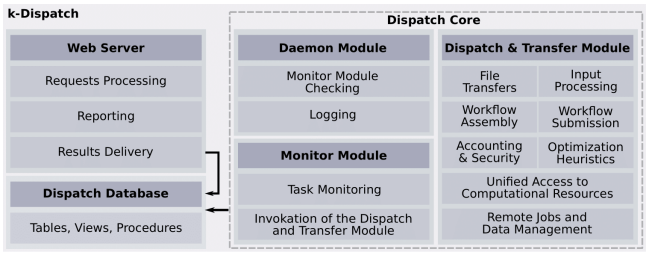

Figure 1: k-Dispatch's modules and a brief description of the actions each module is responsible for. Arrows show the communication between Dispatch Core, Web Server and Dispatch Database.

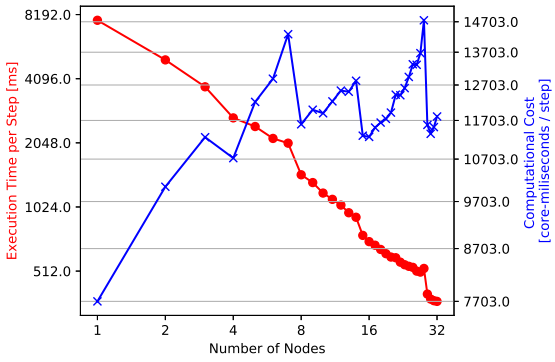

Figure 2: Red line shows the strong scaling of the k-Wave code measured for 1024^3 domain size on the Barbora cluster. Blue line shows the evolution of the computational cost when more nodes are added.

all executed tasks suitable for the execution time estimation. The Dispatch core is responsible for planning, executing and monitoring submitted workflows. The communication with HPC and cloud facilities is done via SSH and RSYNC protocols. For more information, please refer to [**?**].

2.2 Workflow Optimization within k-Dispatch

The optimization algorithm providing suitable parameters for particular tasks of the workflow is integrated inside the Dispatch core. It is composed of four modules: Optimizer, Estimator, Evaluator and Collector. The Optimizer is based on a Genetic Algorithm implemented in the PyGAD library [9] and its parameter settings have been thoroughly investigated in [DOUBLE-BLIND]. The goal of the Optimizer is to generate high quality candidate solutions, each of which holding a list of execution parameters for all tasks in the workflow. In the simplest case, a candidate solution is a vector where the position of the task is given by a breath first traversal through the workflow task graph and the value determines the number of compute nodes to be used. The Optimizer is based on a Genetic Algorithm implemented in the PyGAD library [9] and its parameter settings have been thoroughly investigated in [DOUBLE-BLIND]. The goal of the Optimizer is to generate high quality candidate solutions, each of which holding a list of execution parameters for all tasks in the workflow. In the simplest case, a candidate solution is a vector where the position of the task is given by a breath first traversal through the workflow task graph and the value determines the number of compute nodes to be used.

The Estimator is responsible for estimating the execution time for particular tasks based on their input data and the amount of required resources. The Estimator incorporates various interpolation heuristics to reckon up missing performance data.

The Evaluator uses a simplified simulator of job scheduler called Tetrisator [DOUBLE-BLIND], which takes a candidate solution, simulates its execution on a given cluster and calculates the workflow makespan and cost. Tetrisator is a one-pass simulator of an HPC system with a predefined number of uniform computing nodes. It is inspired by the default strategy of the PBS job scheduler without the backfilling support. The tasks are submitted to the simulator in the order defined in the candidate solution. Workflows may contains multiple dependencies among inner tasks, and the initial cluster workload may be defined, i.e., the cluster is not empty at the workflow submission time.

As soon as a satisfactory solution is found, the workflow is submitted to the real cluster and executed. Upon finishing the execution, the execution times for all tasks are collected by the Collector and stored in the performance database. This data is used to gradually improve the accuracy of the Estimator.

2.3 Estimator Module and Interpolation Techniques

There are many factors that may affect the execution time of a given task. Obviously, the most important ones are the size of the problem stored in the input file, and the amount of resources assigned to the task. As a practical example, let us talk about the MPI implementation of the k-Wave toolbox [DOUBLE-BLIND] simulating (non)-linear propagation of ultrasound wave through a heterogeneous absorbing medium. The scaling of the execution time and cost for one specific problem instance on the Barbora cluster with 36 processor cores per node can be seen in Fig. 2. Here, a domain of 1024^3 grid points is partitioned into 2D slabs and distributed over various numbers of compute nodes (1 to 32). The red curve shows the execution time per one simulation time step (the whole simulation usually executes tens of thousands of time steps).

Although this strong scaling curve looks almost ideal, several sudden drops in the execution time can be observed. These drops are the consequences of well balanced workload distribution. For example, if we cut the domain into 512 slices, we can distribute the work over 512 ranks mapped onto 512 cores. Since k-Wave is a memory and network bound application, it is often advantageous to undersubscribe the

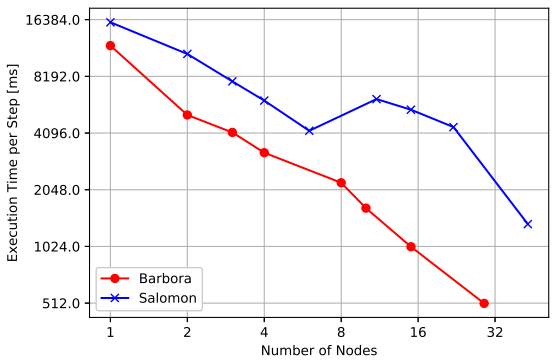

Figure 3: Strong scaling of the k-Wave code execution time measured for 1024^3 domain size on the Barbora (36 cores / node) and Salomon (24 cores / node) cluster.

computing nodes and use higher aggregated memory and network bandwidth. On the Barbora cluster, we can spread 512 ranks over 15 to 28 nodes in a round robin fashion. Since the efficiency of such distribution is decreasing, the scaling curve is flattening toward 28 nodes. However, when 29 nodes are allocated to the task, the domain can be cut into 1024 slices leading to a much better workload distribution and significantly lower execution time. This imperfect workload distribution also renders into the simulation cost since there is a direct proportion between the parallel efficiency and the related cost. The blue curve shows several local minima and maxima in the execution cost which provide very suitable execution parameters or should be avoided, respectively.

Let us note that having a complete performance dataset with all possible input sizes, numbers of nodes, and other tens of simulation parameters is computationally intractable. When having incomplete performance datasets where some points on the curve are missing, the interpolation should rather overestimate the execution time to prevent early task termination. Even more important question is how the scaling curve changes when a previously unseen domain size is used. In this situation, it is necessary to estimate both the shape and the position of the scaling curve from measured strong and weak scaling. As interpolation functions, linear and quadratic interpolation were used.

Finally, the scaling curves may change significantly among different machines. One such an example can be seen in Fig. 3 where the same problem is solved on Barbora (36 Cascade Lake cores per node) and Salomon (24 Haswell cores per node). Not only is the curve shifted due to a lower node performance, but it has a very different shape in the second half. This may be the effect of a different interconnection network topology, but also current cluster utilization. In this case, it is very hard to use any interpolation. Thus when a new cluster is connected to k-Dispatch, a few benchmark runs for the most typical simulation settings are performed to get a minimum amount of performance data.

3 EXPERIMENT SETUP

This paper follows the experimental setup presented in [DOUBLE-BLIND] to evaluate the developed workflow schedules under incomplete performance database. For the makespan and cost evaluation, the Tetrisator simulator worked with a 64 node cluster. The validation of the final schedules was performed on the Barbora cluster, where a static allocation with 54 nodes[2] was created to ensure the same initial conditions for all tests.

3.1 Investigated Workflows

This paper uses two typical biomedical ultrasound workflows applied in the ultrasound neurostimulation and photoacoustic imaging, see Fig. 4. Both workflows consist of two types of tasks. The simulation tasks (ST) executing the k-Wave MPI solver represent heavy parallel jobs running for a few hours. The ST tasks were limited to use between 1 and 32 nodes. The data processing tasks (PT) perform data pre-processing, post-processing, aggregation, etc. These tasks usually use one or two nodes depending on the amount of memory requested, and finishes within a few tens of minutes.

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

3.2 Used Datasets

Let us here define the datasets used in our experiments along with their short description:

- **Dataset A**. Reference strong scaling of the k-Wave code measured on a domain size of $1024 \times 1024 \times 1024$ grid points using 1-32 nodes.
- **Dataset 1A**. Based on *Dataset A* but having only 16 values including peaks and values in between them.
- **Dataset 2A**. Based on *Dataset A* but having only 8 values excluding peaks.
- **Dataset B**. Reference strong scaling of the k-Wave code measured on a domain size of $810 \times 810 \times 810$ grid points using 1-32 nodes.
- **Dataset 1B**. $810 \times 810 \times 810$ domain interpolated for 1-32 nodes using the quadratic interpolation from the known domain sizes: $512 \times 512 \times 512$, $648 \times 648 \times 648$, $1024 \times 1024 \times 1024$.

[2] A 64 node allocation had been requested but due to a cluster failure, experiments had to be performed on only 54 nodes. Thankfully, this did not affect the obtained results significantly.

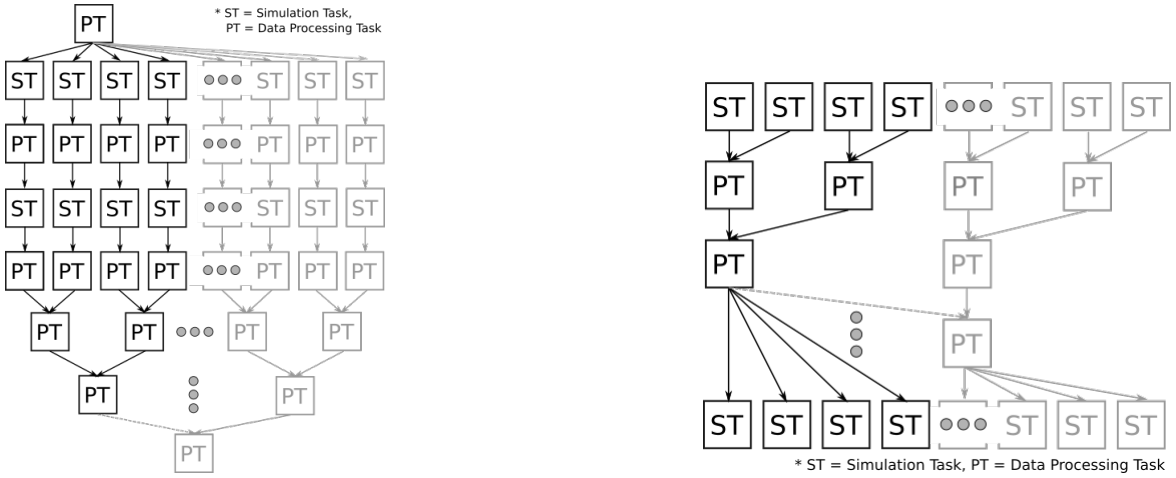

Figure 4: The structure of investigated workflows. The heavy simulation tasks are interleaved with light data processing tasks. The parts highlighted in black show the minimal workflow structure consisting of 20 and 11 tasks, respectively. The parts displayed in grey show how the workflow structure can grow.

3.3 Tetrisator Validation against Real Cluster

To compare the simulator output with the real execution carried out in a dedicated queue comprising 54 nodes of the Barbora cluster, an artificial schedule based on the first workflow type was created. This workflow contained 20 tasks, (8 heavy STs alternated with 12 light PTs). The execution times of particular tasks were taken from the *Dataset A*. The number of simulation time steps inside the ST tasks were reduced to make the workflow finish in less that 1 hour. To prevent premature termination, a safety cup of 10% calculated from the estimated execution time was added to each task. The real execution time actually covers net computing time as well as overheads such as the computing node initialization. Two experimental scenarios were performed: (1) no initial workload, i.e., the cluster was empty when the workflow was submitted and executed, and (2) predefined initial workload, i.e., not all nodes were available for some time which impacts the tasks execution order and may cause some delays. Since, Tetrisator is inspired by PBS but does not perfectly implements all its features like backfilling, this experiment also tries to capture the features that may be beneficial for future simulator extensions. It is expected that the real makespan may be shorter due to backfilling. However, a bit pessimistic prediction is always better than the undervalued one.

3.4 Workflow Schedule Quality Measures

The quality of the developed workflow schedules is evaluated by a fitness function the Optimizer calls after the execution trace has been created by Tetrisator. This work investigates two different fitness functions: GODA and GOSA.

GODA (Global Optimization of the workflow on systems with on-Demand Allocations) calculates the makespan over the longest critical path including queueing times. However, the execution cost considers only truly consumed resources. This is a typical cluster operation with users competing for resources. Since having two contradictory criteria, a user-defined scalarization parameter α is used to balance between the execution time and cost. The algorithm cannot perform a true mutli-objective optimization because there is no further feedback from the user that could select the preferred solution from the Pareto front. Contrary, the most suitable solution has to be chosen autonomously and submitted to the cluster as soon as possible (before the cluster background workload changes significantly).

GOSA (Global Optimization of the workflow on systems with Static Allocations) expects the user holds a dedicated part of the cluster and thus has to to pay for the whole allocation no matter the some nodes may be idle. Although this is a more expensive solution, it usually reduces the queueing time. Since the makespan and cost are directly proportional, no scalarization coefficient is needed and only the makespan is considered.

3.5 Evaluation of Interpolation Techniques

To estimate missing execution time for a particular task, domain size, and number of nodes, two different interpolation techniques from the Python's *scipy* package [17] were used. After a thorough investigation in [DOUBLE-BLIND] and new experiments performed in the paper, a linear and quadratic `interp1d` interpolations were chosen. Very similar results to the quadratic interpolation were also obtained by cubic spline `CubicSpline` with the `bc_type` parameter set to `natural`. Unfortunately, the use of the default value of `bc_type` caused high oscillations and strong underestimations of the execution time. Therefore, we decided to use the quadratic interpolation instead.

Three different experiments with the interpolation functions were conducted. The goals of particular experiments were

- to estimate missing points on the strong scaling curve for a domain size of 1024^3 grid points defined by the points with ideal scaling ($N\%(P*36) \approx 0$), where N is the domain size and P is the number of nodes, see Fig. 7.
- to estimate missing points on the strong scaling curve for a domain size of 1024^3 grid points when having also points in the middle of the intervals between two points with ideal scaling, see Fig. 7.
- to reconstruct a completely unknown scaling curve for an unseen domain size from the data stored in the performance database. In this example, scaling curves for 512^3, 648^3 and 1024^3 were used to estimate the one for 810^3 grid points, see Fig. 8. The domain sizes chosen progressively doubles the total number of grid points.

As the measure of the interpolation quality, a mean relative error was used, see Eq. (1).

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

where a denotes the measured data, b the interpolated data, and N is the total number of the nodes (32).

In all cases, we can tolerate a small overestimation but shall avoid underestimation which leads to premature job termination and necessary resubmission with prolonged execution time.

4 EXPERIMENTAL RESULTS

This section presents and discusses (1) the similarity of the workflow execution schedule to the one executed on a real HPC cluster, and (2) the error reached by the interpolation techniques.

4.1 Simulated Execution Plans Reliability

The following figures point out the differences between simulated execution plans created by Tetrisator and the real executions performed in the dedicated queue on Barbora. Figure 5 shows the first scenario where no initial workload is expected and all 54 nodes are fully available at submission time. As expected, the simulated makespan is a bit pessimistic causing the overestimation by 15%. The second scenario illustrated in Fig. 6 expects initial workload consuming 40 nodes for the first 18 minutes delaying the heavy simulation tasks. In this case, the prediction error reached 10%. In both cases, the overestimation is higher since the backfilling is not implemented. This difference in plans is visible at the bottom of both figures. When there is not enough free resources for task 3, the cluster scheduler lets other tasks to overtake this one while Tetirsator postpones the execution of all following tasks.

Our observations suggest that the real PBS cluster scheduler works in the same manner as Tetrisator. This means the tasks within the workflow are submitted to the real cluster in the same order as they are processed by the Tetrisator, and their submission time is more or less the same. Thus, the tasks are also executed one by one in the same manner as arriving to the cluster. The changes in the order happen when a task has to wait for free resources. We may also say that at least in the dedicated queue, the task executions do not suffer from delays caused by the scheduler's refresh time.

4.2 Interpolation Functions Accuracy

Figure 7 shows the measured and interpolated strong scaling curves on a domain composed of 1024^3 grid points. Inspecting the scaling curve created by a linear interpolation, a very close match can be seen. When interpolating using values where the scaling is close to the optimal, the mean interpolation error reaches 4%. After adding the values from the middle of particular intervals, the error drops below 0.8%. Unfortunately, the interpolated values for sparser training data are mostly underestimated, which can be corrected by a small bias or picking the points with the worst instead of best workload distribution.

When repeating the same experiment with a cubic spline and a quadratic interpolation, the mean error gets higher up to the level of 12% and 7%, respectively, depending on the number of known values. The high error is caused by several oscillations, and more specifically, by the extrapolation error where the execution time is extremely underestimated.

The 4% error of the linear interpolation reaches the level of uncertainty of real execution time measurement on clusters due to unstable node, network and I/O performance. The suitability of the linear interpolation may be also attributed to a very good scaling of the ST tasks without any significant anomalies.

The second experiment attempts to estimate the strong scaling for an unknown domain size, see Fig. 8. The figure reveals that the interpolation method rather overestimate the scaling curve. When repeating this experiment with a linear and a natural cubic spline interpolations, we got the mean error of 25.4% and 13.5%, respectively, while the quadratic interpolation and the cubic spline with `bc_type` parameter set to default produced better estimates reaching the mean error at a level of 10.5%. The explanation is quite simple. While the strong scaling of the ST tasks on a given domain size is almost linear, the algorithm has an asymptotic time complexity of $O(n \log n)$. Moreover, the ST tasks heavily employ fast Fourier transform which is very sensitive to the domain size and its prime factors. The quadratic interpolation thus better capture the nature of ST tasks.

The conclusion is to use a linear interpolation to estimated values on known scaling curves while using a quadratic interpolation when the domain size has not been seen before. It is important to say that the k-Wave code is highly tuned and scales very well. Employing a code the scaling of which is be more "wild" with many peaks or a dramatic slowdowns may become a challenge. On the other hand, if the scaling is relatively stable, it may be possible to construct a scaling

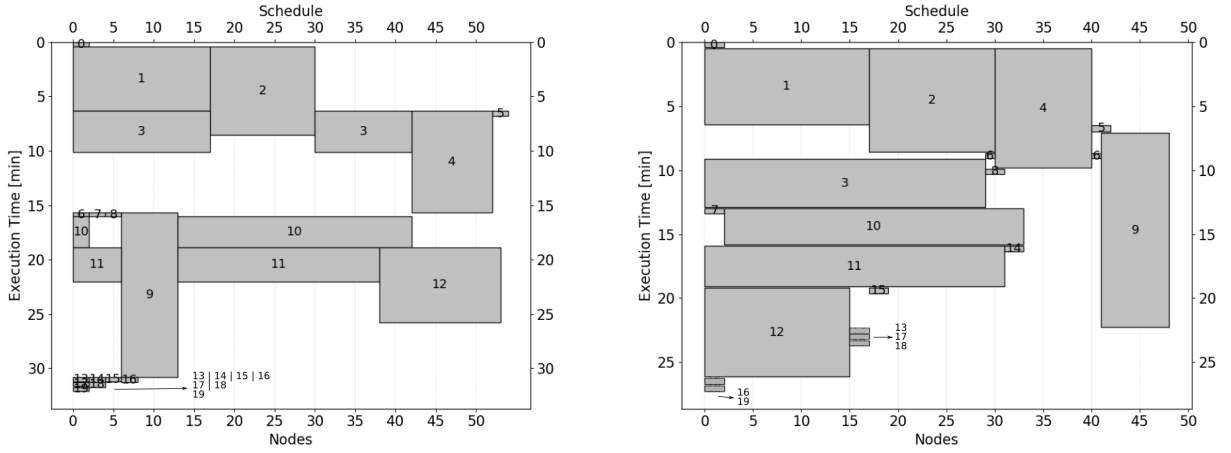

Figure 5: Simulated execution plan (left) finishing in 32.1 minutes while the real execution on Barbora (right) finishing in 27.3 minutes.

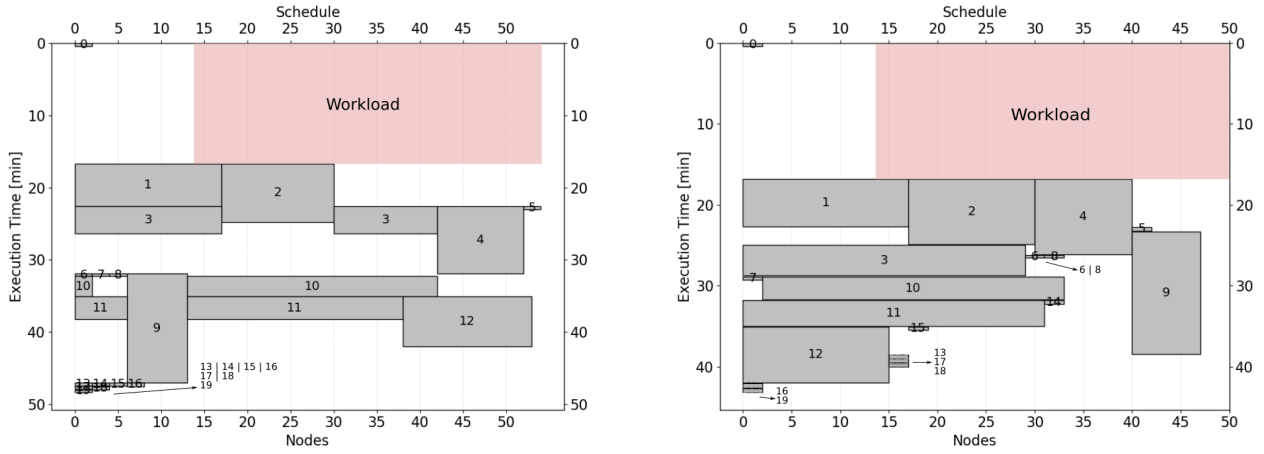

Figure 6: Simulated execution plan (left) finishing in 48.3 minutes and the real execution on Barbora (right) finishing in 43.2 minutes. Pink rectangle denotes cluster workload at the workflow submission time.

equation and use a fitting methods to set its coefficients using known performance data. Alternatively, we may try to interpolate the known points using a various polynomial interpolations and based on the error make a decision about a selection of the interpolation method.

4.3 Interpolation Impact on Schedule Makespan and Cost

This section investigates the quality and accuracy of the developed schedules when using the performance database containing all data, only a subset, or no data for particular domain size.

Figure 9 shows the makespan and cost of the best workflow schedules developed for the GODA situation on a known domain size of 1024^3 grid points, with all, 8 and 16 performance values. These experiments also use different values of the α scalarization coefficient (only three values of α are used in figure for better visibility). The schedules were collected over twenty independent runs of the genetic algorithm. The Pareto fronts (lines in the plot) for the same values of α are close to each other confirming that by employing interpolation methods on incomplete datasets we are able to achieve very similar results. When using *Dataset 2A* containing only 8 performance values, the solutions found may be deflected from that ones evolved using dense dataset. This actually does not mean that found solutions are bad, they just overlap the area where solutions for different value of α would be expected. Next, it can be seen that solutions for different α form isolated clusters. This implies we can affect the execution plan to prioritize different criteria. At this point, it is

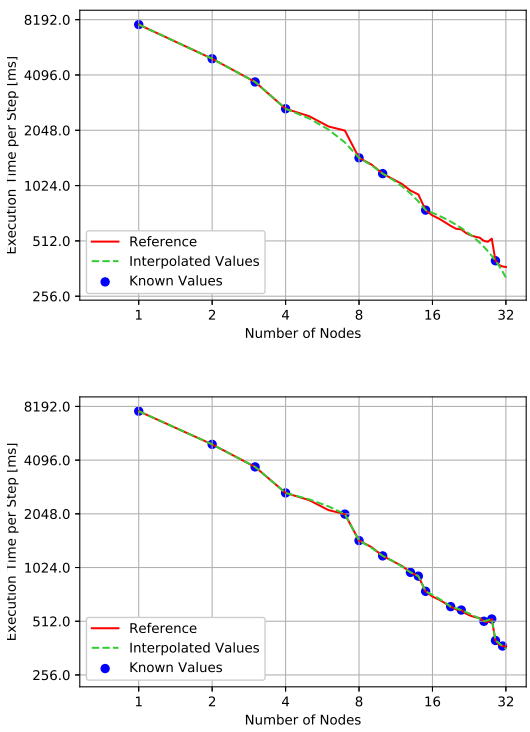

Figure 7: Reference and interpolated strong scaling of ST tasks for a domain size of 1024^3 grid points with a linear interpolation calculated from 8 and 16 known values, respectively. In the top figure, values in unexpected peaks were selected intentionally to see how much the value would be underestimated.

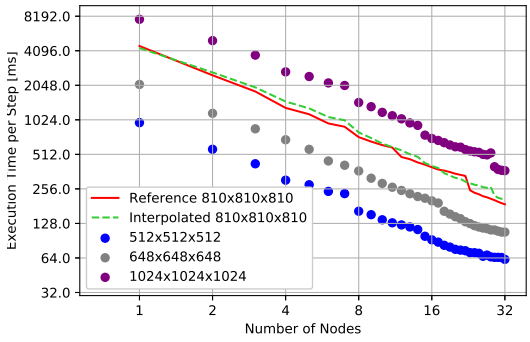

Figure 8: Reference and interpolated strong scaling of ST tasks for an unknown domain size of 810^3 grid points with a quadratic interpolation.

important to note that the execution plan may be adjusted in makespan by a factor of 10.0 while in computational cost by a factor of 1.7. The factors vary and the cost factor is such small due to the highly optimised code used. This is a very promising result showing that when the interpolation is reasonably accurate, the impact on the best solution developed by the Optimizer is rather small.

Table 1 summarizes conducted experiments of GOSA expressing the quality of the execution schedules as makespan. The table may be divided into two parts. The left one is for the domain size of 1024^3 where missing strong scaling values were completed by a linear interpolation. The right one is for the domain size of 810^3 which was fully interpolated using a quadratic interpolation. The difference between the achieved makespan for the full performance dataset and interpolated datasets is given by an interpolation error (investigated in Sec. 4.2) and performance fluctuations of cluster's nodes.

5 CONCLUSIONS

The paper has investigated the optimization of moldable scientific workflow executions under incomplete performance database, i.e., the execution times for some combination of tasks, input data and amount of resources are not known and have to be interpolated from already known data. Consequently, the paper has proved that we can simulate the workflow execution in the real cluster and this simulator can be integrated in the k-Dispatch's optimization module. Although being a one-pass PBS-based simulator without backfilling technique, the estimations provided are sensible. The simulator gives accurate estimations especially for workflows executed on dedicated resources where other workload is known. The cross validation of an artificial and the real schedules created by the PBS job scheduler on Barbora show a good general match.

The experimental results indicate that linear interpolation works well in situations the input data has been seen before and the task has been executed using a few execution parameters configurations. In such cases, the missing performance data can be calculated with a very small error below 4%. However, if the input data has not been seen before, it is necessary to estimate the execution time from similar inputs. In this case, a quadratic interpolation worked sufficiently well, however, the error may reach 10%.

The paper also confirms it is possible to find different schedules that prioritize various criteria using the trade-off parameter α. The proposed optimization algorithm constructs the Pareto front offering different reasonable solutions, i.e. schedules. Users, however, (1) are not aware of what tasks are executed within the workflow, (2) do not know what solution to choose, and finally (3) the Pareto fronts are calculated just before the workflow execution and this information is not available at submission time to k-Dispatch. This is the reason why the multi-criteria optimization is transformed to an easier form where users can express their preferences between two criteria (makespan vs. computational cost) using, e.g., a slider bar. This is provided to users at the workflow submission time and their preference is considered during the optimization process. The experiments show that the scaling

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

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

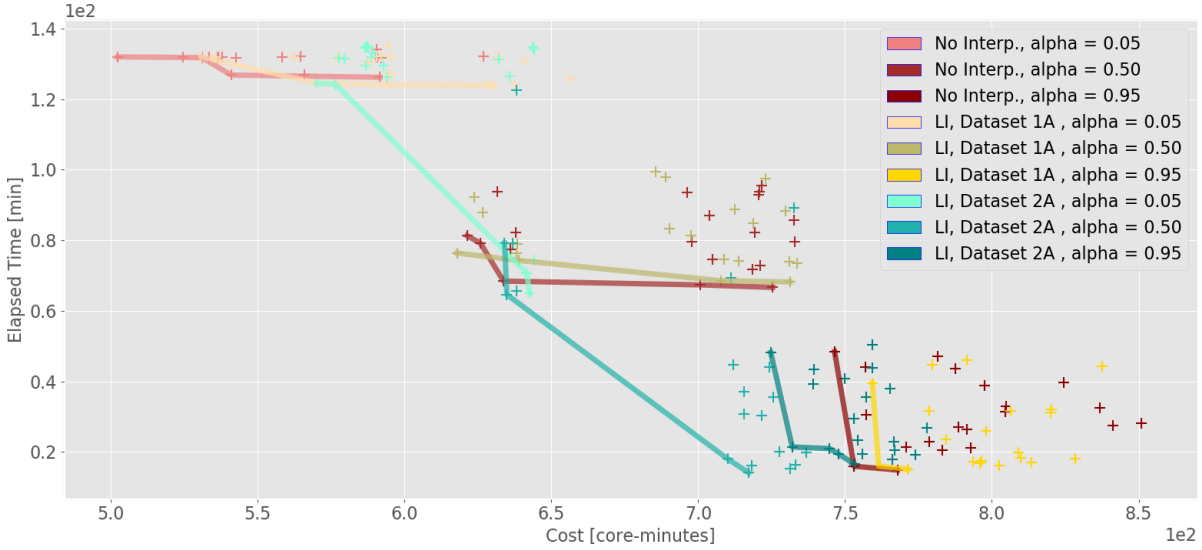

Figure 9: Pareto front together with dominated solutions showing the evolved schedules for workflows of 11 tasks not requiring interpolation, and two experiments both using linear interpolation (LI) but differing in the content of the performance dataset.

factor of each criterion may differ. In our case, it is mostly given by highly optimised and tuned codes.

The developed schedules tend to overestimate the execution time, which is partially caused by imperfect interpolation, by missing support for backfilling which allows short jobs to overtake the longer ones, and a reserve of 10% added to the workflow to avoid premature termination. Nevertheless, the error between developed and real schedules fits within a 15% margin, which is considered to be acceptable for most users.

5.1 Future Work

There are three directions we would like to follow in our future work. First, we would like to include the information about the actual cluster utilization into the cluster simulator. This will allow us to better simulate workflow execution in on-demand allocations where the user competes with others. It may have an impact on the shape of the developed schedules because tasks asking for more resources sit longer in the queue. Using smaller amounts of resources thus may improve the workflow makespan. Second, we would like to implement backfilling technique into our cluster simulator to better

predict the time the task is started. Finally, we would like to examine more advanced machine learning techniques to improve the interpolation accuracy once the performance database includes tens of thousands of records.

6 ACKNOWLEDGMENTS

Identification of funding sources and other support, and thanks to individuals and groups that assisted in the research and the preparation of the work should be included in an acknowledgment section, which is placed just before the reference section in your document.

REFERENCES

[1] G 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

[2] Shishir Bharathi, Ann Chervenak, Ewa Deelman, Gaurang Mehta, Mei-Hui Su, and Karan Vahi. 2008. Characterization of scientific workflows. In *Third Workshop on Workflows in Support of Large-Scale Science*. IEEE, 1–10. https://doi.org/10.1109/WORKS.2008.4723958

[3] Raphael Bleuse, Sascha Hunold, Safia Kedad-Sidhoum, Florence Monna, Gregory Mounie, and Denis Trystram. 2017. Scheduling Independent Moldable Tasks on Multi-Cores with GPUs. *IEEE Transactions on Parallel and Distributed Systems* 28, 9 (sep 2017), 2689–2702. https://doi.org/10.1109/TPDS.2017.2675891

[4] Tony F. Chan and Tarek P. Mathew. 1994. Domain decomposition algorithms. *Acta Numerica* 3 (jan 1994), 61–143. https://doi.org/10.1017/S0962492900002427

[5] Artem M. Chirkin, Adam S.Z. Belloum, Sergey V. Kovalchuk, Marc X. Makkes, Mikhail A. Melnik, Alexander A. Visheratin, and Denis A. Nasonov. 2017. Execution time estimation for workflow scheduling. *Future Generation Computer Systems* 75 (2017). https://doi.org/10.1016/j.future.2017.01.011

[6] Ewa Deelman, Karan Vahi, Gideon Juve, et al. 2014. Pegasus: a Workflow Management System for Science Automation. *Future Generation Computer Systems* (2014).

[7] Pierre-François Dutot, Marco A. S. Netto, Alfredo Goldman, and Fabio Kon. 2005. Scheduling Moldable BSP Tasks. In *Lecture Notes in Computer Science*. 157–172. https://doi.org/10.1007/11605300_8

[8] Dror G. Feitelson and Larry Rudolph. 1996. Toward convergence in job schedulers for parallel supercomputers. In *Lecture Notes in Computer Science*. 1–26. https://doi.org/10.1007/BFb0022284

[9] Ahmed F. Gad. 2021. GeneticAlgorithmPython: Building Genetic Algorithm in Python.

[10] David E. Goldberg. 1989. *Genetic algorithms in search, optimization, and machine learning*. Addison-Wesley Longman, Boston, MA. 372 pages. https://doi.org/10.5555/534133

[11] Robert L. Henderson. 1995. Job scheduling under the Portable Batch System. In *Lecture Notes in Computer Science*. 279–294. https://doi.org/10.1007/3-540-60153-8_34

[12] Matthias Hovestadt, Odej Kao, Axel Keller, and Achim Streit. 2003. Scheduling in HPC resource management systems: Queuing vs. planning. *Lecture Notes in Computer Science* 2862, June (2003), 1–20. https://doi.org/10.1007/10968987_1

[13] Habib Izadkhah. 2019. Learning based genetic algorithm for task graph scheduling. *Applied Computational Intelligence and Soft Computing* (2019). https://doi.org/10.1155/2019/6543957

[14] Anna-Lena Lamprecht and Kenneth J. Turner. 2016. Scientific workflows. *J. on Software Tools for Technology Transfer* 18, 6 (nov 2016), 575–580. https://doi.org/10.1007/s10009-016-0428-z

[15] Fatma A. Omara and Mona M. Arafa. 2010. Genetic algorithms for task scheduling problem. *J. Parallel and Distrib. Comput.* 70, 1 (2010), 13–22. https://doi.org/10.1016/j.jpdc.2009.09.009

[16] Yves Robert. 2011. Task Graph Scheduling. *Encyclopedia of Parallel Computing* (2011), 2013–2025. https://doi.org/10.1007/978-0-387-09766-4_42

[17] Pauli Virtanen, Ralf Gommers, and Travis E. andand others Oliphant. 2020. SciPy 1.0: Fundamental Algorithms for Scientific Computing in Python. *Nature Methods* 17 (2020), 261–272. https://doi.org/10.1038/s41592-019-0686-2

[18] Katherine Wolstencroft, Robert Haines, Donal Fellows, et al. 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

[19] Deshi Ye, Danny Z. Chen, and Guochuan Zhang. 2018. Online scheduling of moldable parallel tasks. *Journal of Scheduling* 21, 6 (2018), 647–654. https://doi.org/10.1007/s10951-018-0556-2

[20] Andy B. Yoo, Morris A. Jette, and Mark Grondona. 2003. SLURM: Simple Linux Utility for Resource Management. In *Lecture Notes in Computer Science*. 44–60. https://doi.org/10.1007/10968987_3