CHAPTER 17

# PREDICTING THE PERFORMANCE OF SCIENTIFIC WORKFLOW TASKS FOR CLUSTER RESOURCE MANAGEMENT: AN OVERVIEW OF THE STATE OF THE ART

**Jonathan Bader**
Technische Universität Berlin, DOS Group, Berlin, Germany
jonathan.bader@tu-berlin.de

**Kathleen West**
University of Glasgow, School of Computing Science, Glasgow, UK
kathleen.west@glasgow.ac.uk

**Soeren Becker**
Technische Universität Berlin, DOS Group, Berlin, Germany
soeren.becker@tu-berlin.de

**Svetlana Kulagina**
Humboldt-Universität zu Berlin, Institute for Computer Science, Berlin, Germany
kulagins@hu-berlin.de

**Fabian Lehmann**
Humboldt-Universität zu Berlin, Institute for Computer Science, Berlin, Germany
fabian.lehmann@hu-berlin.de

**Lauritz Thamsen**
University of Glasgow, School of Computing Science, Glasgow, UK
lauritz.thamsen@glasgow.ac.uk

**Henning Meyerhenke**
Humboldt-Universität zu Berlin, Institute for Computer Science, Berlin, Germany
meyerhenke@hu-berlin.de

**Odej Kao**
Technische Universität Berlin, DOS Group, Berlin, Germany
odej.kao@tu-berlin.de

**ABSTRACT:** Scientific workflow management systems support large-scale data analysis on cluster infrastructures. For this, they interact with resource managers which schedule workflow tasks onto cluster nodes. In addition to workflow task descriptions, resource managers rely on task performance estimates such as main memory consumption and runtime to efficiently manage cluster resources. Such performance estimates should be automated, as user-based task performance estimates are error-prone.
In this book chapter, we describe key characteristics of methods for workflow task runtime and memory prediction, provide an overview and a detailed comparison of state-of-the-art methods from the literature, and discuss how workflow task performance prediction is useful for scheduling, energy-efficient and carbon-aware computing, and cost prediction.

## 17.1 Introduction

When running a scientific workflow on a computing infrastructure, scientists usually have to define task resource consumption limits, such as the available time [1] or memory [2] for task execution. Setting these resource limits correctly is necessary to enable a successful execution of workflows as exceeding runtime or memory limits results in failures, commonly through errors issued by cluster resource managers. Resource consumption estimates provided by the scientists are known to be error-prone and inaccurate [2–4]. To avoid a failure and timely re-execution of the workflow or parts of it, scientists tend to overprovision resources to ensure that their workflows are fully executed. The problem is further aggravated as different instances of the same workflow task consume varying amounts of resources [3, 5] when executed on different inputs. Assume, for example, that a task unzips its input files. Such a task's runtime will heavily depend on the size of the input files. Hence, scientists have to conservatively configure consumption limits for the largest task instances.

While a failed workflow execution due to an underestimated resource consumption directly affects scientists, an overestimation does so indirectly: It reduces the number of tasks running in parallel as more cluster resources than necessary are allocated to tasks. The cluster resource manager ensures that the requested memory is reserved for specific tasks, even if it is not fully used, leaving less memory for other tasks to run as well. This effectively decreases the level of overall parallelism and, hence, cluster throughput. Meanwhile, overestimating the runtime of tasks can lead to significantly suboptimal schedules as the scheduler assumes wrong runtimes [6, 7].

Predicting the performance of workflow tasks to set their resource consumption limits automatically can relieve scientists of the need to define limits manually. State-of-the-art approaches often focus on either runtime or memory prediction, as these are key examples of performance prediction where underestimation leads to task termination in resource managers [6, 8, 9]. Many approaches use machine learning to predict the performance of workflow tasks. Some of these approaches rely on historical data to model a task's performance [2, 10], while other approaches learn during the workflow execution and incrementally update their model online [11, 12]. In addition, some approaches use profiling to predict the performance of workflow tasks before the workflows are executed [13, 14]. The approaches also differ in their ability to predict performance for different types of infrastructure, such as homogeneous or heterogeneous infrastructures, their used prediction model, or their model input.

Figure 17.1 provides a high-level overview of the interactions among components within the scientific workflow execution environment. The Scientific Workflow Management System (SWMS) parses the abstract workflow and coordinates task submission to the resource manager, which then schedules and allocates tasks onto the cluster infrastructure, concurrently monitoring their resource usage. Workflow task performance prediction employs predictive models to generate runtime, memory, and other resource-related estimations for individual tasks. These predictions are accessible to both the SWMS and the resource manager. Utilizing predictions at the SWMS level offers the advantage of leveraging workflow graph information, which is typically unavailable to the resource manager. Conversely, the resource manager benefits from a global view, enabling it to optimize resource allocation decisions across multiple workflows and users. Recent work proposes unified solutions that leverage information from both components, enabling direct integration of task performance predictions [15, 16].

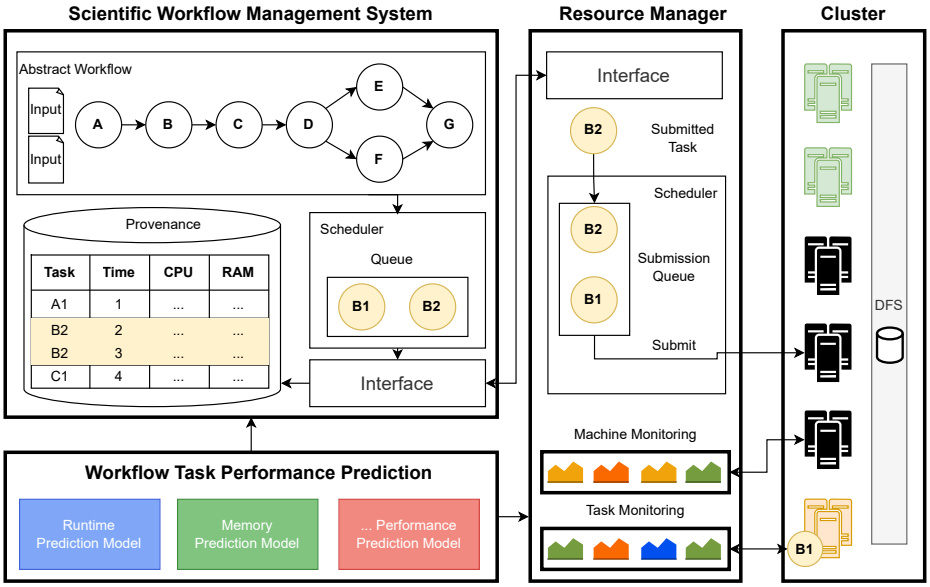


Figure 17.1: Interaction between a scientific workflow management system, a resource manager, and task performance prediction models.

In this chapter, we give an overview of state-of-the-art workflow task performance prediction methods, compare them, and discuss use cases as well as ongoing research. In particular, we compare methods that focus on **workflow task runtime** and **workflow task memory** prediction. That is, we do not include methods that only predict the total execution time or total memory consumption of entire workflow applications, as task-level predictions serve as fine-grained foundation for workflow-level predictions. Likewise, we do not include methods that are not specific to scientific workflows. The main contributions of this chapter are:

- We identify key characteristics of workflow task runtime (Section 17.2.1) and memory prediction methods (Section 17.3.1), covering general aspects such as code availability, model-specific criteria like the prediction method and applicability to heterogeneous infrastructures, as well as model inputs and evaluation settings.
- We provide an overview and a detailed comparison of state-of-the-art workflow task runtime (Section 17.2) and memory prediction methods (Section 17.3), analyzing nineteen state-of-the-art papers based on the previously defined characteristics and identifying shortcomings while highlighting future research directions.
- We exemplify how workflow task performance prediction is useful for resource management, namely scheduling, energy-efficient and carbon-aware computing, and cost prediction, and present recent research in each of these fields while also highlighting limitations (Section 17.4).

The remainder of this chapter is structured as follows. Section 17.2 describes state-of-the-art workflow task runtime prediction methods and compares them, while Section 17.3 describes state-of-the-art workflow task memory prediction methods. Section 17.4 presents applications of workflow task performance prediction for resource management and discusses the shortcomings of state-of-the-art methods. Lastly, Section 17.5 concludes the chapter.

Table 17.1: Overview of state-of-the-art methods for workflow task runtime prediction.

| | General | | Model | | | | Model Input | | | | Evaluation | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| Publication | Code Available | Domain Specific | Prediction Method | Training Time | Heterogeneous Infrastructure | GPU Support | Hardware Capacity | Hardware Performance | Task Input Size | Task Resource Usage | Workflow System | # Workflows | Experiment Type |
| [17] | ○ | ○ | Diverse | Offline | ● | ○ | ● | ● | ● | ○ | - | - | Sim. Real |
| [18] | ○ | ○ | NN | Offline | ● | ○ | ● | ● | ● | ● | Askalon | 3 | Sim. Real |
| [19] | ○ | ○ | Regr. & Clust. | Online | ○ | ○ | ○ | ○ | ● | ○ | Pegasus | 3 | Sim. Real |
| [20] | ○ | ○ | Regr. & Clust. | Online | ○ | ○ | ○ | ○ | ● | ○ | Pegasus | 5 | Sim. Real |
| [21] | ○ | ○ | Regr. | Offline | ● | ○ | ● | ● | ● | ● | - | 4 | Sim. Syn. |
| [22] | ○ | ○ | NN | Offline | ○ | ○ | ○ | ○ | ● | ○ | - | 3 | Sim. Real |
| [10] | ○ | ○ | NN | Online | ● | ○ | ● | ● | ● | ● | Diverse | 2 | Sim. Real |
| [13] | ● | ○ | Regr. | Pre | ● | ○ | ● | ● | ● | ● | Nextflow | 5 | Sim. Real |
| [23] | ○ | ○ | NN | Offline | ● | ○ | ● | ● | ○ | ● | - | - | Sim. Real |
| [14] | ● | ◑ | Regr. | Pre | ● | ○ | ● | ● | ● | ○ | Nextflow | 5 | Sim. Real |

●Yes | ○No | ◑Partially

## 17.2 Workflow Task Runtime Prediction

In this chapter, we explore the features of runtime prediction methods for workflow tasks and offer a detailed overview and thorough comparison of the latest task runtime prediction techniques found in the literature. Table 17.1 gives an overview.

### 17.2.1 Characteristics

To provide an overview and compare methods for predicting the runtime of tasks, we introduce a set of characteristics divided into four areas: general, model, input, and evaluation.

**General:** First, we describe the general characteristics of workflow task runtime prediction methods, including information about the *publication year*, *public code availability*, and whether the method is *domain-specific* or can be applied to all types of workflows.

**Model:** The model characteristics describe essential workflow task runtime prediction models' attributes. The *prediction method* attribute describes the models' underlying prediction method, such as a regression or a neural network. The *training time* attribute describes when the model is trained, either offline on historical data, online during workflow execution, or pre-workflow execution using profiling without historical data. The *heterogeneity* attribute shows whether a method is applicable to heterogeneous infrastructures or restricted to homogeneous ones. With the recent popularity of computing on GPUs, we have added a *GPU support* attribute as additional information to show whether a prediction method lends itself to running workflow tasks on both CPUs and GPUs.

**Model Input:** The model input shows the data used to perform the runtime prediction. The first attribute is the hardware capacity that shows whether static hardware information, such as the number of CPU cores, amount of memory, or disk sizes, is incorporated. Complementary, the *hardware performance* characteristic considers performance metrics in the model input, such as processor speed or I/O capabilities. An important feature for workflow task runtime prediction is the *task input data size*, which can be the general input size or a placeholder, for instance, bytes read. Finally, we include *task resource utilization* as an attribute and analyze whether the models use task resource utilization, such as CPU or memory utilization, as an input.

**Evaluation:** The attributes under the evaluation category show how the proposed methods were evaluated. The first attribute, *workflow system*, shows which workflow system was used to either run the experiments or gather the traces, followed by the *number of workflows* attribute, aiming to quantify the scope. The *experiment type* shows if the method was evaluated on real infrastructures, simulated with traces gathered from real executions, or simulated with synthetic traces.

### 17.2.2 Offline Workflow Task Runtime Prediction

Offline predictors build their model based on historical data and before the workflow execution is started, without the capabilities to improve the model online during workflow execution.

Matsunaga et al. [17] perform offline runtime prediction and evaluate several machine learning approaches for their ability to predict task runtime. Their method can be applied to heterogeneous infrastructures as it incorporates static and performance characteristics of the hardware. They conducted their evaluation on two bioinformatic tools, including an analysis of the impact of individual features on prediction accuracy, arguing for including as many features as possible and letting the algorithm decide on the specific selection. The authors also show that different algorithms perform better for different setups, i.e., the effectiveness of an algorithm depends on the task and its training data.

Malik et al. [18] leverage runtime provenance data to train a multilayer perceptron (MLP)-based neural network that predicts task execution times on grid infrastructures. Model input features include static and dynamic environment features such as a machine's CPU speed or free memory, pre-execution features such as input size, and execution features such as task resource usage. Their model is trained offline, using a feature set optimizer, a model space analyzer, an MLP generator, and a model trainer component. The authors evaluate their method with three workflows, MeteoAG, Invmod, and Wien2K, using Askalon as the workflow system, and show a prediction error between 17% and 26%.

Pham et al. [21] present an offline task runtime prediction method for cloud environments using a two-stage approach that first predicts task resource usage, followed by a second stage that uses the output of the first stage, the workflow input data, and static and dynamic hardware information for a regression to predict execution time on a target machine. Their prediction model distinguishes between pre-runtime parameters, e.g., workflow input data or VM types, and runtime parameters, such as CPU, memory, I/O operations, and bandwidth. In the first stage, pre-runtime parameters are considered to derive the runtime parameters for the execution. In the second stage, the task execution time on a target VM is predicted with a regression model using the output data from the first stage, the workflow input data, and the VM specifications. Their method is applicable to heterogeneous infrastructures and evaluated with generated data from four different workflows and diverse scientific workflow management systems.

Nadeem et al. [22] present a method for offline prediction of task execution times in grid environments using a radial basis function neural network. The learning model incorporates four specific types of information: a) workflow structure, including aspects like workflow name or dependency flow; b) application details, such as executable names or file sizes; c) execution environments, covering factors like grid locations or submission time; and d) resource state, including metrics like the number of jobs in the queue or jobs running. However, it does not account for hardware characteristics; hence it is not able to predict runtimes for heterogeneous infrastructures. For their evaluation, they used generated data from four different workflows and various scientific workflow management systems. Similar to Matsunaga et al. [17], their evaluation outcome shows which features have the greatest impact on the predicted runtime and which ones can be left out.

Huang et al. [23] propose CloudProphet, a method for predicting task execution times for public cloud infrastructures. CloudProphet operates from the view of a data center operator, also assuming that the VM where the task is running, is a black box. Thus, in a first step, the application type is identified based on task resource consumption, using a Dynamic Time Warping algorithm to normalize the temporal difference in the training traces. Next, highly correlated metrics are selected. Then, a neural network predicts the performance of the task using the highly correlated metrics as input. In addition, the workload level is predicted using a second neural network that outputs the task's workload level. A third neural network

is then used that takes the output of the first as input and predicts the performance baseline that can be used to calculate the performance degradation. Their evaluation uses five cloud benchmarks from CloudSuite and shows similar prediction accuracy to Pham et. al. [21] and Bader et al. [13].

### 17.2.3 Online Workflow Task Runtime Prediction

Da Silva et al. [19] predict task resource consumption, such as runtime, disk space, and memory consumption, for tasks in scientific workflows. Based on monitoring tools and historical data, they apply a regression tree for resource prediction. Beforehand, they identify data subsets with a high correlation by using density-based clustering. Then, they predict the expected resource usage for correlated data points based on the ratio in this specific cluster. In the case of uncorrelated data, they predict the mean runtime of previous executions. The authors evaluate their method with three workflow traces gathered on real infrastructures using the Pegasus workflow management system.

In an extension of their work, da Silva et al. [20] test a normal and a gamma distribution for uncorrelated data and sample in the case of statistical significance. They also extend their evaluation with two new workflow traces gathered from real executions.

Hilman et al. [10] employ an online incremental learning method utilizing long short-term memory networks (LSTMs) for predicting task runtimes in heterogeneous cloud environments. Their method uses two categories of metrics: pre-runtime and during runtime. The pre-runtime metrics encompass details of the task, virtual machine (VM) specifications, and the time of task submission. In contrast, the runtime metrics include parameters like CPU usage, memory consumption, and I/O operations, represented as historical time-series data. This data is continuously augmented online, during the workflow execution, and is utilized to train and refine the model after task completion. The method is evaluated on two workflows with generated tasks.

### 17.2.4 Pre-Execution Workflow Task Runtime Prediction

In our own work, we proposed the Lotaru method [13, 14], which predicts workflow task runtimes for heterogeneous infrastructures without using historical data and before the workflow is executed on the target infrastructure. Lotaru quickly profiles the local machine and the target infrastructure using a set of microbenchmarks, then runs the workflow with downsampled data on the local machine to collect task metrics. Using the hardware performance, task input, and resource usage data, tasks are assessed as CPU- or I/O-intensive, and a Bayesian linear regression model is trained to predict the runtime on target machines. For evaluation, five real-world workflows from the nf-core repository [24] were executed on six different node types using the Nextflow workflow management system, which generated experimental traces.

### 17.2.5 Discussion

Table 17.1 provides an overview of the state-of-the-art workflow task runtime prediction methods presented in the literature.

Of the ten methods, only two methods provide their source code, experimental setup scripts, or configurations as open source. With limited access to the source code, the re-

producibility of the results and also their applicability for real-world scientific workflows is limited.

One of the methods is partially domain-specific for bioinformatics and uses data attributes to downsample the input for profiling. The other methods are generally applicable, but are often evaluated only for a single domain.

For the machine learning prediction method, we can see that the majority uses neural networks and regression-based models.

Five methods predict task runtimes offline, three online, and two before workflow execution using profiling. Thus, there are different methods for each part of the workflow execution cycle.

Seven methods are able to predict runtimes for heterogeneous environments such as clouds, but none of these methods consider the use of GPUs.

The model inputs differ significantly between the different methods. All but one method make use of the task input size. Three methods furthermore only make use of the input size, while all other methods consider multiple factors for predictions, including hardware performance and capacity as well as task resource usage.

All methods are evaluated in a simulation setup, mostly using traces from real workflow executions on real infrastructures, with the most commonly used workflow systems being Pegasus and Nextflow. Evaluating these methods by integrating and evaluating them in real execution environments is an open research point. This would, for example, allow to study the two-way interactions between online task runtime predictors and schedulers that make use of predictions that can occur in real systems.

## 17.3 Workflow Task Memory Prediction

In this section, we cover the characteristics of workflow task memory prediction methods and provide a detailed overview and comprehensive comparison of state-of-the-art memory prediction methods from the literature. Table 17.2 gives a structured overview of existing memory prediction methods.

### 17.3.1 Characteristics

In this section, we describe the key features, which we divide in the three areas: General, Model, and Evaluation.

**General:** Again, the general characteristics include information about the *year of publication* and *code availability*. It also includes a classification of general or *domain-specific* applicability, as we observed that some methods require certain data properties that are only fulfilled for certain domains or use cases.

**Model:** The model characteristics include details about the underlying *prediction model*, machine learning or statistical models such as regression trees, linear regression methods, reinforcement learning, or ensemble methods that use multiple machine learning models. The model feature also encompasses the *training time*, whether memory is predicted from historical traces (i.e., offline) or online during execution and if *prediction errors* are handled for subsequent predictions. Furthermore, we examined whether machine *heterogeneity* is

Table 17.2: Overview of state-of-the-art workflow task memory prediction methods.

| | General | | Model | | | | | Evaluation | | |
|---|---|---|---|---|---|---|---|---|---|---|
| | Publication | Code Available | Domain Specific | Prediction Method | Online/Offline | Heterogeneous Infrastructure | Error Handling | Task Input Size | Workflow System | # Number Workflows | Experiment Type |
| [19] | ○ | ○ | Regr. & Clust. | Online | ○ | ○ | ● | Pegasus | 3 | Sim. |
| [20] | ○ | ○ | Regr. & Clust. | Online | ○ | ○ | ● | Pegasus | 5 | Sim. |
| [12] | ● | ○ | Analytical | Online | ○ | ● | ○ | Makeflow | 3 | Sim. |
| [2] | ● | ○ | Regr. | Offline | ○ | ● | ● | IceProd | 1 | Sim. |
| [11] | ○ | ○ | Regr. | Online | ○ | ● | ◑ | Pegasus | 5 | Sim Syn. |
| [25] | ○ | ○ | RL | Online | ○ | ● | ○ | Nextflow | 5 | Real |
| [9] | ● | ● | Analytical | Online | ○ | ● | ○ | Coffea | 1 | Real |
| [26] | ● | ○ | Time Series Regr. | Online | ○ | ● | ● | Nextflow | 2 | Sim. Real |
| [27] | ● | ○ | Multiple | Online | ○ | ● | ● | Nextflow | 6 | Sim. Real |
| [5] | ● | ○ | Rule & Regr. | Online | ○ | ● | ● | Nextflow | 4 | Real |
| [28] | ● | ○ | Time Series Regr. | Online | ○ | ● | ● | Nextflow | 2 | Sim. Real |

●Yes | ○No | ◑Partially

considered, such as different processors, memory configurations, and memory access architectures (e.g., UMA vs. NUMA). We do not divide this characteristic further, as we found no support in existing state-of-the-art methods. Finally, we analyze whether the prediction model assumes a correlation between *input data size* and memory usage.

**Evaluation:** Again, we examine whether the *experiment type* uses real executions, a simulation on real data, or a simulation on synthetic data. We also look at the *number of workflows* used and the *workflow management system* used either to execute the workflow or to collect traces.

### 17.3.2 Offline Task Memory Prediction

Witt et al. [2] propose a method that optimizes the resource wastage instead of the prediction error. The authors show a correlation between the size of the input data and the memory consumption of a task and accordingly develop a linear prediction model that incorporates the input sizes as a feature. Their method doubles the predicted memory on a task failure. The method is evaluated using a simulation with the traces of a Neutrino Observatory experiment log. The authors' method achieves less wastage than user estimates and the baseline of Tovar el al. [12]. In their experiments, they further investigate different failure handling strategies, i.e., how to handle task failures due to memory underestimation, and show a significant impact on wasted memory resources, but also that there is no best strategy for all kinds of tasks.

### 17.3.3 Online Task Memory Prediction

The method developed by da Silva et al. [19] and its extension [20] can also be applied to memory prediction for correlated data points based on the ratio in this specific cluster. We provided a comprehensive overview of this method in the previous section 17.2.3.

Tovar et al. [12] introduce a strategy for predicting task memory in scientific workflows. The authors develop a predictive model that estimates the peak memory demand of a given task using an online analytical approach during workflow execution. The model is adjustable, focusing on either maximizing throughput or minimizing resource wastage. Their optimization is based on the slow-peaks model, which assumes the scenario where task failures occur towards the end of the execution. The authors suggest using a two-step policy that first allocates the predicted amount of memory, followed by allocating the maximum available resources in case a task execution fails. While the authors acknowledge the potential for a more complex multi-phase failure handling policy, they note it requires more in-depth investigation. Their method is evaluated by a simulation using traces from three workflow executions with the Makeflow workflow management system, showing an overall reduction in memory wastage and an increase in throughput.

Tovar et al. [9] present a domain-specific solution to predict memory for task executions and splitting tasks into smaller slices. The first five executed tasks will be assigned the maximum memory of a node. Afterward, the actual used memory is monitored, and the maximum ever observed is assigned plus a small margin. When a task fails, it is split into two equally sized tasks and resubmitted, leading to potentially multiple subsequent retries and splits. However, the authors also mention that the possibility of task splitting is due to the nature of independent events that comprise the tasks, i.e., the domain knowledge. Initially,

the task size is determined by the linear correlation between the chunk size of a task and its resource usage. Task splitting has also been proposed in the field of genomics, but is not generally applicable to black-box tasks [29]. Their evaluation uses the coffea framework and runs the TopEFT workflow on a cluster of 40 commodity nodes. The results show the efficiency of their method, reducing wasted memory resources.

Witt et al. [11] present a feedback-based peak memory allocation method. The authors propose two different predictors. The first predictor is a linear regression model that uses the sum of the task's input files' size as a feature to predict peak memory. They dynamically offset their prediction to avoid an underallocation. For example, the "LR mean ±" strategy incorporates the standard deviation as an offset, while the "LR mean -" method accounts solely for negative prediction errors. As a second predictor, they use the percentile of historical peak memory usage, such as the median or the $99^{\text{th}}$ percentile. In this case, no offset is selected. For failed task executions, the previously predicted peak memory will be multiplied by a factor of two. The methods are evaluated using the DynamicCloudSim simulation environment with five synthetic traces from the Pegasus Workflow Generator. The authors also examine the relationship between wasted memory and scheduling policies and show that the two mutually influence each other.

We [25] present two reinforcement learning methods that predict the peak memory requirements of tasks. Our first method is based on gradient bandits that must learn a preference for a particular memory configuration. The agent's reward function includes a penalty for memory underallocations to discourage the agent from choosing such actions. Their failure strategy ensures that no allocations smaller than the failed one are chosen. The second method uses Q-learning agents that increase, decrease, or maintain memory allocation. The agents' reward function includes an artificial minimum and maximum amount of memory to allocate and discourages insufficient memory allocations. We evaluate the reinforcement learning approaches using five workflows from the nf-core repository and ran them on real infrastructures, showing that the methods are able to reduce memory wastage compared to the workflow defaults.

Lehmann et al. [5] propose a memory sizing strategy called Ponder to dynamically decide whether a linear relationship between a task's data input size and the memory consumption exists. Ponder aims to reduce the chance of out-of-memory failures and uses a rule-based method to predict a task's memory. If a relationship is detected, linear regression is applied; otherwise, the highest observed memory value is used. The method also adjusts predictions to ensure they are reasonable, particularly when they exceed historical values. Additionally, Ponder includes a weighted offset and a static offset to account for variability in memory consumption with the same input data and task across different runs. The authors evaluate their approach for four different nf-core workflows on Kubernetes.

We [27] recently introduced an online method that uses multiple machine learning methods and dynamically selects the best-performing one. During workflow execution, the model is continuously retrained and updated to incorporate metrics from recent task executions. A novel resource allocation quality score is used to continuously assess the goodness of a model's prediction, which is based on accuracy and efficiency. We use not one but many offsetting techniques and dynamically select the one that would have caused the lowest resource wastage. The method is evaluated on traces of six real-world workflows gathered with the workflow management system Nextflow. The experimental results significantly outperform existing state-of-the-art peak memory prediction methods from the literature.

We [26, 28] also propose a memory prediction approach that uses time series monitoring

data in order to provide a dynamic memory prediction, i.e., a memory prediction over time. Due to the dynamic approach, the method first predicts the runtime of a workflow task by training a linear model and offsets the predicted time. Then, the expected runtime of the task is segmented according to predefined values. Now, for each of the segments, a linear regression model is trained to predict the maximum memory usage for each segment. The prediction is offset for each segment to avoid memory underallocation. Since our model has two aspects, runtime and memory, we discuss and propose different error-handling methods. For example, the selective retry strategy, which adjusts only the failed segment, or the partial retry strategy, which adjusts all segments from the one that failed. The method is evaluated in a simulation using traces from two real-world workflows from the publicly available nf-core repository and outperforms state-of-the-art methods in terms of reducing memory wastage.

### 17.3.4 Discussion

Table 17.2 gives an overview of eleven state-of-the-art methods for predicting memory usage for workflow tasks from the literature. We divided the characteristics we included in our comparison into three categories: General, Model, and Evaluation.

Of the eleven methods presented, seven make their source code publicly available. Compared to workflow task runtime prediction methods from the literature, this is a higher percentage. This makes it easier to use existing methods as baselines for newer approaches if prototype implementations are readily available.

One of the presented methods can only be applied to a specific domain, all others are generally applicable. The prediction models are very heterogeneous and use different machine learning methods.

One method is applied offline, while ten methods run online. This is a clear difference to task runtime prediction methods, where only a couple of methods learned prediction models during the runtime of workflows.

Strikingly, none of the methods considers a heterogeneous infrastructure. It can be assumed that this is the case because, from our experience, the memory consumption of the same workflow task on different machines is often quite stable, especially when using the same operating system and the same underlying processor architecture, unlike the runtime of the workflow task, so that there is less need to evaluate methods across different machines.

Nine of the eleven methods perform error handling, i.e., they adjust the prediction after a task failure. Many of the papers emphasized that the use of error handling has a significant impact on wasted memory resources and is thus almost as important as the initial memory prediction.

Eight methods are evaluated using simulations, while three evaluate their method on real systems. We believe that most methods are simulated for practical reasons: The execution of large workflows is time-consuming, and re-running parts due to memory failures further increases the experiment time while consuming valuable cluster resources and energy. In addition, simulations make it easy to try out a variety of error-handling strategies and method configurations. However, once a well-working method has been found, it would be beneficial to evaluate the methods in real-world systems, as this would increase the integration in real scientific workflow management systems.

Overall, Pegasus was a commonly used workflow management system for evaluation in older publications, while newer publications often use Nextflow.

## 17.4 Applications of Task Performance Prediction for Scientific Workflow Resource Management

Many advanced resource management methods make use of workflow task resource predictions. Prominent examples include workflow scheduling methods that aim to minimize overall workflow execution time (modeled by the makespan) [30, 31], methods for energy- and carbon-efficient workflow execution [32, 33], and methods that aim to predict and optimize the cost of executing workflows [34, 35]. Figure 17.2 provides an overview of a typical execution environment for scientific workflows and shows how resource prediction can be employed. When scheduling to minimize the makespan, the resource manager uses predicted resources, such as memory and runtime, to determine the task submission order and the best node for each task. Energy-efficient and carbon-efficient workflow execution incorporates information about the energy efficiency of available cluster machines and the availability of low-carbon energy. Finally, cost prediction considers the price of using cluster machines and uses resource estimates to select the most cost-effective machine for a task. In the following, we describe each of these resource management methods in more detail and summarize relevant work.

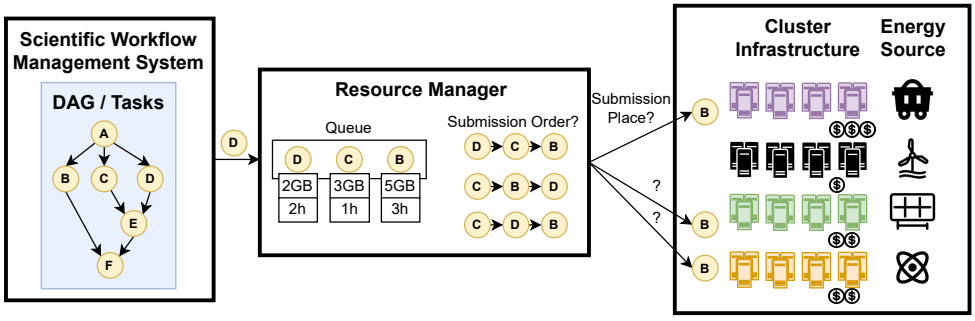


Figure 17.2: Typical execution environment where task performance predictions are used for scheduling, energy-efficient and carbon-aware execution, and cost prediction. The cluster can take different forms, such as an on-premise system, a virtual cluster in the cloud, or a hybrid solution.

### 17.4.1 Workflow Scheduling

**Introduction:** Scheduling tasks has been a central topic in workflow research for decades [30, 36]. For large-scale workflows, which are often executed on parallel and distributed platforms, scheduling involves deciding where and when to execute each task within the workflow. These decisions are guided by specific objectives, typically aiming to optimize performance or resource utilization.

Scheduling approaches can typically be categorized into online and plan-based strategies. Online scheduling assigns tasks to resources as they become available, which provides flexibility but often limits global optimization. Plan-based strategies, by contrast, create a full schedule before execution begins, leveraging a broader decision horizon to achieve better results for objectives like makespan. In this section, we focus on plan-based strategies as

they benefit more from accurate runtime estimates.

A common objective in task scheduling is minimizing the makespan, which serves as a proxy for the total execution time of the workflow [37]. However, other objectives and constraints, such as fault tolerance [38], throughput or latency [39], and energy efficiency (see Section 17.4.2), have also been studied. These objectives often conflict with each other; for instance, reducing energy consumption may lead to longer execution times. Addressing such trade-offs often requires multi-objective optimization, adding complexity to the scheduling problem.

The computational complexity of finding optimal schedules for large scientific workflows, particularly on heterogeneous or distributed platforms, has motivated extensive research into heuristic approaches. These methods provide practical solutions by approximating optimal schedules. This section focuses on heuristic approaches, summarizing selected plan-based works, highlighting their contributions to the field of workflow task scheduling.

**Selected Works:** Creating a plan-based schedule is NP-hard, even on homogeneous platforms [40], motivating the use of heuristic optimization methods in practice. Among these, two common approaches are list scheduling and partitioning-based heuristics. List scheduling algorithms can vary in their approach, with many employing greedy strategies, while partitioning-based methods divide the workflow into blocks to reduce the search space.

HEFT [30] is a seminal list-scheduling algorithm, which has been extended in various ways to address different scheduling problem formulations, including PEFT [41]. Further extensions include the work of Shi and Dongarra [42], as well as dynamic variants like DVR HEFT [43] and P-HEFT [44]. Additionally, cloud-oriented adaptations, such as E-HEFT [31], demonstrate the flexibility of this approach across diverse execution environments.

Partitioning-based heuristic schedulers, including dagP [45], DagHetPart [46], and the framework by Viil and Srirama [47], are particularly beneficial for large-scale workflows. By grouping tasks into blocks and assigning these blocks to nodes, these methods reduce the computational complexity compared to scheduling individual tasks.

Dealing with multiple objectives adds further complexity. One common approach is to combine multiple objectives into a single one, as demonstrated by Liu et al. [48]. Alternatively, Pareto-optimal solutions can be sought, as done by Su et al. [49]. For example, Bathie et al. [50] propose a dynamic ILP-based scheduling algorithm that accounts for memory constraints. However, these exact approaches are computationally expensive and thus mostly limited to smaller workflows.

Another crucial challenge is the impact of imprecise task runtime estimations. Chirkin et al. [51] explore stochastic scheduling methods to mitigate such inaccuracies, emphasizing the importance of preparation steps in improving scheduling performance. Additionally, practical systems have begun integrating monitoring tools that track resource consumption and refine scheduling decisions over time [16, 52, 53]. These systems highlight the growing importance of dynamic, data-driven scheduling in real-world applications.

**Conclusion:**

- Scheduling algorithms differ in their objective functions and solution approaches. Among these, minimizing the overall makespan of the workflow remains the most commonly pursued objective in workflow scheduling.

- Most scheduling approaches depend on accurate estimates of task resource requirements. Inaccurate estimates can lead to suboptimal schedules, impacting both performance and efficiency.
- Plan-based techniques, which often achieve better makespans due to their broader decision horizon, are particularly sensitive to errors in task resource predictions.
- To address uncertainties in task resource estimations, practical systems increasingly integrate adaptive techniques, such as runtime monitoring and feedback mechanisms, to refine scheduling decisions dynamically.

### 17.4.2 Energy Efficiency and Carbon Awareness

**Introduction:** Executing workflows more energy-efficiently reduces their energy consumption. This can often be achieved by right-sizing, only allocating for example the amount of memory required by workflow tasks. Similarly, scheduling can reduce energy consumption by ensuring that cluster resources are utilized well by task instances and little energy goes to idling resources. In contrast, improving the carbon efficiency of a workflow reduces its carbon footprint without necessarily reducing its energy consumption. This is possible by aligning a workflow's execution and, therefore, its load and energy consumption with the availability of low-carbon energy from variable renewable sources such as wind and solar. All these optimizations have in common that knowledge of the runtime and resource utilization of scientific workflow tasks is useful, so that they directly benefit from accurate performance predictions.

In the past, research has predominantly focused on improving the energy efficiency of workflows [32, 54, 55]. More recently, however, research has started to optimize specifically for carbon awareness of delay-tolerant batch processing workloads [56–58]. In this section, we first summarize selected related work that improves the energy efficiency and carbon awareness of workflows, before we draw a conclusion.

**Selected Works:** A range of methods has been proposed to improve energy efficiency and carbon awareness for workflows. Below, we summarize key contributions in both areas.

GreenHEFT [32] and MOHEFT [54] are widely recognized algorithms for energy-efficient workflow scheduling. GreenHEFT assigns tasks to nodes based on the lowest estimated energy consumption, while MOHEFT considers a trade-off between predicted runtime and energy consumption. Both methods rely on performance predictions using models trained on historical data.

Reddy et al. [55] introduce a scheduling framework that clusters tasks based on their runtime and assigns clusters to virtual machines (VMs) according to predicted resource availability. Their simulation results show reductions in resource consumption, workflow makespan, and energy usage. However, as task clustering depends on runtime estimates, the approach is sensitive to inaccuracies in these predictions.

Wen et al. [59] propose an adaptive scheduling approach for industrial workflows using a genetic algorithm. The method minimizes monetary costs and maximizes the use of low-carbon energy by scheduling tasks across geographically distributed data centers. This approach demonstrates the potential for reducing carbon emissions while maintaining low additional execution costs.

More recent carbon-aware execution methods align workloads with the variable availability of low-carbon energy in a single location, leveraging temporal load shifting and resource

scaling. This goes beyond the previous approach, which migrates workloads to locations where more renewable energy is available, as such migration is not always possible due to security/privacy constraints and data transmission costs.

Wiesner et al. [56] examine the potential of shifting the execution of delay-tolerant workloads to times when more low-carbon energy is available to reduce carbon emissions. The simulations assume that the duration of tasks and iterations is known prior to executing jobs. In our work [14], we address this assumption by using a task runtime prediction model for carbon-aware scheduling and, thereby, demonstrate the relevance of accurate runtime predictions in reducing carbon emissions.

Souza et al. [33] move beyond simulations by experimentally evaluating carbon savings from simple carbon-aware execution mechanisms, such as applying them to workflows like BLAST. These experiments assume that task runtimes are known in advance, which may not reflect real-world conditions.

Hanafy et al. [60] develop an algorithm to minimize a workload's carbon emissions through carbon-aware scaling. This varies the resource usage of a given batch processing application according to variations in the carbon intensity of grid energy, allocating more resources when more low-carbon energy is available. The approach assumes knowledge of the application's runtime and relies on a measured scaling factor obtained through short profiling runs. As such, the method would be affected by inaccurate runtime estimates and also assumes that the application's scaling behavior is constant even for long-running applications.

We note that none of these recent methods are specific to scientific workflows, despite workflows being extraordinarily suited to carbon-aware scheduling and scaling, as they are commonly significantly delay tolerant, efficiently interruptible, and highly scalable in terms of allocations for both individual tasks and entire workflows, leading to a substantial potential for carbon-optimized execution [61].

**Conclusion:**

- State-of-the-art methods are able to reduce resource consumption, workflow makespan, energy consumption, and carbon emissions using knowledge of workflow task resource requirements.
- However, these methods assume that accurate information on the expected task runtime and resource usage are available, which may not be the case in real-world systems.
- Furthermore, there are few methods that are workflow-specific and also few that simultaneously aim at energy and carbon efficiency by, for example, first targeting high resource utilization and then aligning execution with the availability of low-carbon energy.

### 17.4.3 Cost Optimization and Prediction

**Introduction:** Cost prediction and optimization for large-scale workflows is crucial for scientists and enterprises, as they regularly work with limited budgets [35, 62]. Workflow task performance predictions are a prerequisite for this, providing key information such as expected task runtimes and resource requirements. Costs can be monetary if the target infrastructure is a cloud environment, but they can also be execution slots on shared clusters that are not directly monetary. As cost optimization and prediction is a broad topic, we

provide representative examples covering workflow cost prediction for cloud services and minimizing cost under deadline constraints. In this section, we summarize selected related work on predicting and optimizing the cost of executing a workflow.

**Selected Works:** Rosa et al. [35, 62] propose provisioning services for cloud federations that can report the cost of workflow execution beforehand. This is achieved by predicting execution times using a multiple linear regression model based on historical data. Validated with two real-world bioinformatic workflows, this approach not only ensures accuracy but also empowers users to optimize the cost and efficiency of utilizing federated cloud resources for scientific workflows.

Alkhanak et al. [34] provide an overview of scheduling methods for scientific workflows, emphasizing cost optimization and offering a taxonomy of cost-centric metrics. The authors note that the cost of executing a workflow on an infrastructure and the time it takes to do so are inversely proportional, as faster nodes tend to be more expensive, but these can be conflicting optimizations. As a result, the scheduling mechanism must balance the cost of execution and the timing of the workflow, both of which are influenced by inaccuracies in resource predictions.

Malawski et al. [63] propose a method to minimize the cost of executing scientific workflows on cloud infrastructures. Their model employs mathematical programming languages to formulate a mixed integer programming problem, utilizing knowledge about task runtimes as one of the inputs. The models are evaluated with synthetic workflows and the real-world Montage workflow running on AWS, demonstrating optimization for cost-efficiency.

Zhou et al. [64] propose two approaches that focus on minimizing the cost of scheduling workflows while adhering to deadline constraints, as well as an approach that aims to optimize cost and makespan at the same time. Their approaches use genetic algorithms, including chromosome encoding, evaluation and selection, and crossover and mutation. Their simulated evaluation shows that their approach significantly reduces monetary costs compared to existing algorithms under the same deadline constraints.

**Conclusion:**

- State-of-the-art methods from the literature can achieve significant cost optimizations in cloud environments and accurately predict execution costs using detailed information about the resource usage of workflow tasks.
- However, these methods typically assume accurate resource prediction, an assumption that may not hold in real-world systems and leaves room for research that accounts for these uncertainties.
- State-of-the-art methods predominantly target cost prediction and optimization in cloud environments, with limited focus on shared cluster environments, where costs are measured in execution slots or time rather than monetary terms.

## 17.5 Summary

In this chapter, we have presented an overview of state-of-the-art runtime and memory prediction methods for workflow tasks. First, we identified key characteristics of task performance prediction methods. Second, we described and compared state-of-the-art methods in detail. Third, we explained how such runtime and memory prediction methods are useful for

advanced resource management methods such as workflow scheduling, energy-efficient and carbon-aware workflow execution, and workflow cost prediction and optimization.

Our overview of state-of-the-art runtime prediction methods has shown that many methods account for hardware heterogeneity, but none targets task execution on GPUs, a class of devices that is becoming widely used for AI/ML applications. Meanwhile, heterogeneity is not considered for any of the memory prediction methods. We assume that this is due to less variability in memory consumption assuming the same operating system and architecture, but it leaves room for further research if such assumptions are not met. Furthermore, most methods for runtime and memory prediction have only been evaluated via simulations based on workflow execution traces. Finally, we have observed that the majority of publications do not publish their code, impeding reproducibility and reuse.

## Acknowledgments

This work was supported by the German Research Foundation (DFG) as part of the Collaborative Research Center 1404 "FONDA – Foundations of Workflow Systems for Large Scale Scientific Data Analysis", project ID 414984028.

## Bibliography

[1] C. Bailey Lee, Y. Schwartzman, J. Hardy, and A. Snavely, "Are user runtime estimates inherently inaccurate?," in *Int. Workshop on Job Scheduling Strategies for Parallel Processing*, pp. 253–263, 2005.

[2] C. Witt, J. van Santen, and U. Leser, "Learning low-wastage memory allocations for scientific workflows at icecube," in *Int. Conf. on High Performance Computing & Simulation (HPCS)*, 2019.

[3] T. S. Phung, L. Ward, K. Chard, and D. Thain, "Not all tasks are created equal: Adaptive resource allocation for heterogeneous tasks in dynamic workflows," in *Workshop on Workflows in Support of Large-Scale Science (WORKS)*, pp. 17–24, 2021.

[4] A. Hirales-Carbajal, A. Tchernykh, R. Yahyapour, J. L. González-García, T. Röblitz, and J. M. Ramírez-Alcaraz, "Multiple workflow scheduling strategies with user run time estimates on a grid," *Journal of Grid Computing*, vol. 10, pp. 325–346, 2012.

[5] F. Lehmann, J. Bader, N. De Mecquenem, X. Wang, V. Bountris, F. Friederici, U. Leser, and L. Thamsen, "Ponder: Online prediction of task memory requirements for scientific workflows," in *Int. Conf. on e-Science*, 2024.

[6] R. Ramírez-Velarde, A. Tchernykh, C. Barba-Jimenez, A. Hirales-Carbajal, and J. Nolazco-Flores, "Adaptive resource allocation with job runtime uncertainty," *Journal of Grid Computing*, vol. 15, pp. 415–434, 2017.

[7] A. Ilyushkin and D. Epema, "The impact of task runtime estimate accuracy on scheduling workloads of workflows," in *Int. Symp. on Cluster, Cloud and Grid Computing (CCGRID)*, pp. 331–341, 2018.

[8] C. Witt, M. Bux, W. Gusew, and U. Leser, "Predictive performance modeling for distributed batch processing using black box monitoring and machine learning," *Information Systems*, vol. 82, pp. 33–52, 2019.

[9] B. Tovar, B. Lyons, K. Mohrman, B. Sly-Delgado, K. Lannon, and D. Thain, "Dynamic task shaping for high throughput data analysis applications in high energy physics," in *Int. Parallel and Distributed Processing Symposium (IPDPS)*, 2022.

[10] M. H. Hilman, M. A. Rodriguez, and R. Buyya, "Task runtime prediction in scientific workflows using an online incremental learning approach," in *Int. Conf. on Utility and Cloud Computing (UCC)*, pp. 93–102, 2018.

[11] C. Witt, D. Wagner, and U. Leser, "Feedback-based resource allocation for batch scheduling of scientific workflows," in *Int. Conf. on High Performance Computing & Simulation*, 2019.

[12] B. Tovar, R. F. da Silva, G. Juve, E. Deelman, W. Allcock, D. Thain, and M. Livny, "A job sizing strategy for high-throughput scientific workflows," *IEEE Transactions on Parallel and Distributed Systems*, vol. 29, no. 2, 2017.

[13] J. Bader, F. Lehmann, L. Thamsen, J. Will, U. Leser, and O. Kao, "Lotaru: Locally estimating runtimes of scientific workflow tasks in heterogeneous clusters," in *Int. Conf. on Scientific and Statistical Database Management*, 2022.

[14] J. Bader, F. Lehmann, L. Thamsen, U. Leser, and O. Kao, "Lotaru: Locally predicting workflow task runtimes for resource management on heterogeneous infrastructures," *Future Generation Computer Systems*, vol. 150, 2024.

[15] D. H. Ahn, N. Bass, A. Chu, J. Garlick, M. Grondona, S. Herbein, H. I. Ingólfsson, J. Koning, T. Patki, T. R. Scogland, *et al.*, "Flux: Overcoming scheduling challenges for exascale workflows," *Future Generation Computer Systems*, vol. 110, pp. 202–213, 2020.

[16] F. Lehmann, J. Bader, F. Tschirpke, L. Thamsen, and U. Leser, "How workflow engines should talk to resource managers: A proposal for a common workflow scheduling interface," in *Int. Symp. on Cluster, Cloud and Internet Computing (CCGrid)*, 2023.

[17] A. Matsunaga and J. A. Fortes, "On the use of machine learning to predict the time and resources consumed by applications," in *Int. Conf. on Cluster, Cloud and Grid Computing*, pp. 495–504, 2010.

[18] M. J. Malik, T. Fahringer, and R. Prodan, "Execution time prediction for grid infrastructures based on runtime provenance data," in *Workshop on Workflows in Support of Large-Scale Science*, pp. 48–57, 2013.

[19] R. Ferreira Da Silva, G. Juve, E. Deelman, T. Glatard, F. Desprez, D. Thain, B. Tovar, and M. Livny, "Toward fine-grained online task characteristics estimation in scientific workflows," in *Workshop on Workflows in Support of Large-Scale Science*, pp. 58–67, 2013.

[20] R. Ferreira Da Silva, G. Juve, M. Rynge, E. Deelman, and M. Livny, "Online task resource consumption prediction for scientific workflows," *Parallel Processing Letters*, vol. 25, no. 03, 2015.

[21] T.-P. Pham, J. J. Durillo, and T. Fahringer, "Predicting workflow task execution time in the cloud using a two-stage machine learning approach," *IEEE Transactions on Cloud Computing*, vol. 8, no. 1, pp. 256–268, 2017.

[22] F. Nadeem, D. Alghazzawi, A. Mashat, K. Fakeeh, A. Almalaise, and H. Hagras, "Modeling and predicting execution time of scientific workflows in the grid using radial basis function neural network," *Cluster Computing*, vol. 20, no. 3, pp. 2805–2819, 2017.

[23] D. Huang, L. Costero, A. Pahlevan, M. Zapater, and D. Atienza, "Cloudprophet: A machine learning-based performance prediction for public clouds," *IEEE Transactions on Sustainable Computing*, pp. 1–16, 2024.

[24] P. A. Ewels, A. Peltzer, S. Fillinger, H. Patel, J. Alneberg, A. Wilm, M. U. Garcia, P. Di Tommaso, and S. Nahnsen, "The nf-core framework for community-curated bioinformatics pipelines," *Nature Biotechnology*, vol. 38, no. 3, pp. 276–278, 2020.

[25] J. Bader, N. Zunker, S. Becker, and O. Kao, "Leveraging reinforcement learning for task resource allocation in scientific workflows," in *Int. Conf. on Big Data*, 2022.

[26] J. Bader, N. Diedrich, L. Thamsen, and O. Kao, "Predicting dynamic memory requirements for scientific workflow tasks," in *Int. Conf. on Big Data*, 2023.

[27] J. Bader, F. Skalski, F. Lehmann, D. Scheinert, J. Will, L. Thamsen, and O. Kao, "Sizey: Memory-efficient execution of scientific workflow tasks," in *Int. Conf. on Cluster Computing*, 2024.

[28] J. Bader, A. Lößer, L. Thamsen, B. Scheuermann, and O. Kao, "Ks+: Predicting workflow task memory usage over time," in *Int. Conf. on e-Science*, pp. 1–7, 2024.

[29] S. Mohammadi, L. PourKarimi, M. Zschäbitz, T. Aretz, N. De Mecquenem, U. Leser, and K. Reinert, "Optimizing job/task granularity for metagenomic workflows in heterogeneous cluster infrastructures.," in *EDBT/ICDT Workshops*, 2024.

[30] H. Topcuoglu, S. Hariri, and M.-Y. Wu, "Performance-effective and low-complexity task scheduling for heterogeneous computing," *IEEE Transactions on Parallel and Distributed Systems*, vol. 13, no. 3, 2002.

[31] Y. Samadi, M. Zbakh, and C. Tadonki, "E-heft: Enhancement heterogeneous earliest finish time algorithm for task scheduling based on load balancing in cloud computing," in *Int. Conf. on High Performance Computing & Simulation (HPCS)*, pp. 601–609, 2018.

[32] J. J. Durillo, V. Nae, and R. Prodan, "Multi-objective energy-efficient workflow scheduling using list-based heuristics," *Future Generation Computer Systems*, vol. 36, 2014.

[33] A. Souza, N. Bashir, J. Murillo, W. Hanafy, Q. Liang, D. Irwin, and P. Shenoy, "Ecovisor: A virtual energy system for carbon-efficient applications," in *Int. Conf. on Architectural Support for Programming Languages and Operating Systems*, 2023.

[34] E. N. Alkhanak, S. P. Lee, R. Rezaei, and R. M. Parizi, "Cost optimization approaches for scientific workflow scheduling in cloud and grid computing: A review, classifications, and open issues," *Journal of Systems and Software*, vol. 113, pp. 1–26, 2016.

[35] M. J. Rosa, C. G. Ralha, M. Holanda, and A. P. Araujo, "Computational resource and cost prediction service for scientific workflows in federated clouds," *Future Generation Computer Systems*, vol. 125, pp. 844–858, 2021.

[36] M. Weske, G. Vossen, and C. B. Medeiros, "Scientific workflow management: WASA architecture and applications," *Technical Report, Universität Münster, Germany*, 1996.

[37] J. Liu, E. Pacitti, and P. Valduriez, "A survey of scheduling frameworks in big data systems," *International Journal of Cloud Computing*, vol. 7, no. 2, pp. 103–128, 2018.

[38] A. Benoit, V. Le Fèvre, L. Perotin, P. Raghavan, Y. Robert, and H. Sun, "Resilient scheduling of moldable parallel jobs to cope with silent errors," *IEEE Transactions on Computers*, vol. 71, no. 7, pp. 1696–1710, 2021.

[39] A. Benoit, V. Rehn-Sonigo, and Y. Robert, "Multi-criteria scheduling of pipeline workflows," in *Int. Conf. on Cluster Computing*, pp. 515–524, 2007.

[40] M. R. Garey and D. S. Johnson, *Computers and Intractability: A Guide to the Theory of NP-Completeness*. W. H. Freeman and Company, 1979.

[41] H. Arabnejad and J. G. Barbosa, "List scheduling algorithm for heterogeneous systems by an optimistic cost table," *IEEE Transactions on Parallel and Distributed Systems*, vol. 25, no. 3, pp. 682–694, 2013.

[42] Z. Shi and J. J. Dongarra, "Scheduling workflow applications on processors with different capabilities," *Future Generation Computer Systems*, vol. 22, no. 6, pp. 665–675, 2006.

[43] S. Sandokji and F. Eassa, "Dynamic variant rank heft task scheduling algorithm toward exascle computing," *Procedia Computer Science*, vol. 163, pp. 482–493, 2019.

[44] J. G. Barbosa and B. Moreira, "Dynamic scheduling of a batch of parallel task jobs on heterogeneous clusters," *Parallel computing*, vol. 37, no. 8, pp. 428–438, 2011.

[45] M. Y. Özkaya, A. Benoit, B. Uçar, J. Herrmann, and Ü. V. Çatalyürek, "A scalable clustering-based task scheduler for homogeneous processors using DAG partitioning," in *Int. Parallel and Distributed Processing Symposium (IPDPS)*, pp. 155–165, 2019.

[46] S. Kulagina, H. Meyerhenke, and A. Benoit, "Mapping large memory-constrained workflows onto heterogeneous platforms," in *Int. Conf. on Parallel Processing*, 2024.

[47] J. Viil and S. N. Srirama, "Framework for automated partitioning and execution of scientific workflows in the cloud," *The Journal of Supercomputing*, vol. 74, pp. 2656–2683, 2018.

[48] J. Liu, E. Pacitti, P. Valduriez, D. De Oliveira, and M. Mattoso, "Multi-objective scheduling of scientific workflows in multisite clouds," *Future Generation Computer Systems*, vol. 63, pp. 76–95, 2016.

[49] S. Su, J. Li, Q. Huang, X. Huang, K. Shuang, and J. Wang, "Cost-efficient task scheduling for executing large programs in the cloud," *Parallel Computing*, vol. 39, no. 4-5, pp. 177–188, 2013.

[50] G. Bathie, L. Marchal, Y. Robert, and S. Thibault, "Dynamic DAG scheduling under memory constraints for shared-memory platforms," *International Journal of Networking and Computing*, vol. 11, no. 1, pp. 27–49, 2021.

[51] A. M. Chirkin, A. S. Belloum, S. V. Kovalchuk, M. X. Makkes, M. A. Melnik, A. A. Visheratin, and D. A. Nasonov, "Execution time estimation for workflow scheduling," *Future Generation Computer Systems*, vol. 75, pp. 376–387, 2017.

[52] J. Bader, L. Thamsen, S. Kulagina, J. Will, H. Meyerhenke, and O. Kao, "Tarema: Adaptive resource allocation for scalable scientific workflows in heterogeneous clusters," in *Int. Conf. on Big Data*, pp. 65–75, 2021.

[53] J. Liu, E. Pacitti, P. Valduriez, and M. Mattoso, "Scientific workflow scheduling with provenance data in a multisite cloud," *Transactions on large-scale data-and knowledge-centered systems XXXIII*, pp. 80–112, 2017.

[54] J. J. Durillo, R. Prodan, and J. G. Barbosa, "Pareto tradeoff scheduling of workflows on federated commercial clouds," *Simulation Modelling Practice and Theory*, vol. 58, 2015.

[55] P. V. Reddy and K. G. Reddy, "A Multi-Objective Based Scheduling Framework for Effective Resource Utilization in Cloud Computing," *IEEE Access*, vol. 11, 2023.

[56] P. Wiesner, I. Behnke, D. Scheinert, K. Gontarska, and L. Thamsen, "Let's wait awhile: How temporal workload shifting can reduce carbon emissions in the cloud," in *International Middleware Conference*, 2021.

[57] A. Radovanović, R. Koningstein, I. Schneider, B. Chen, A. Duarte, B. Roy, D. Xiao, M. Haridasan, P. Hung, N. Care, S. Talukdar, E. Mullen, K. Smith, M. Cottman, and W. Cirne, "Carbon-Aware Computing for Datacenters," *IEEE Transactions on Power Systems*, vol. 38, no. 2, pp. 1270–1280, 2023.

[58] L. Lin and A. A. Chien, "Adapting Datacenter Capacity for Greener Datacenters and Grid," in *Int. Conf. on Future Energy Systems*, 2023.

[59] Z. Wen, S. Garg, G. S. Aujla, K. Alwasel, D. Puthal, S. Dustdar, A. Y. Zomaya, and R. Ranjan, "Running Industrial Workflow Applications in a Software-Defined Multi-cloud Environment Using Green Energy Aware Scheduling Algorithm," *IEEE Transactions on Industrial Informatics*, vol. 17, no. 8, pp. 5645–5656, 2021.

[60] W. A. Hanafy, Q. Liang, N. Bashir, D. Irwin, and P. Shenoy, "CarbonScaler: Leveraging Cloud Workload Elasticity for Optimizing Carbon-Efficiency," *Proc. of the ACM on Measurement and Analysis of Computing Systems*, vol. 7, no. 3, pp. 57:1–57:28, 2023.

[61] K. West, F. Lehmann, V. Bountris, U. Leser, Y. Elkhatib, and L. Thamsen, "Exploring the Potential of Carbon-Aware Execution for Scientific Workflows," in *Int. Symp. on Cluster, Cloud and Internet Computing*, IEEE, 2025.

[62] M. J. Rosa, A. P. Araüjo, and F. L. Mendes, "Cost and time prediction for efficient execution of bioinformatics workflows in federated cloud," in *Int. Conf. on Bioinformatics and Biomedicine (BIBM)*, pp. 1703–1710, 2018.

[63] M. Malawski, K. Figiela, M. Bubak, E. Deelman, and J. Nabrzyski, "Scheduling multilevel deadline-constrained scientific workflows on clouds based on cost optimization," *Scientific Programming*, vol. 2015, 2015.

[64] J. Zhou, T. Wang, P. Cong, P. Lu, T. Wei, and M. Chen, "Cost and makespan-aware workflow scheduling in hybrid clouds," *Journal of Systems Architecture*, vol. 100, p. 101631, 2019.