Predicting Dynamic Memory Requirements for Scientific Workflow Tasks

Jonathan Bader*‡, Nils Diedrich*‡, Lauritz Thamsen§, and Odej Kao‡
‡ {firstname.lastname}@tu-berlin.de, Technische Universität Berlin, Germany
§ lauritz.thamsen@glasgow.ac.uk, University of Glasgow, United Kingdom

***Abstract*—With the increasing amount of data available to scientists in disciplines as diverse as bioinformatics, physics, and remote sensing, scientific workflow systems are becoming increasingly important for composing and executing scalable data analysis pipelines. When writing such workflows, users need to specify the resources to be reserved for tasks so that sufficient resources are allocated on the target cluster infrastructure. Crucially, underestimating a task's memory requirements can result in task failures. Therefore, users often resort to overprovisioning, resulting in significant resource wastage and decreased throughput.**

In this paper, we propose a novel online method that uses monitoring time series data to predict task memory usage in order to reduce the memory wastage of scientific workflow tasks. Our method predicts a task's runtime, divides it into k equally-sized segments, and learns the peak memory value for each segment depending on the total file input size. We evaluate the prototype implementation of our method using workflows from the publicly available nf-core repository, showing an average memory wastage reduction of 29.48% compared to the best state-of-the-art approach.

***Index Terms*—Resource Management, Scientific Workflow, Memory Prediction, Cluster Computing, Machine Learning**

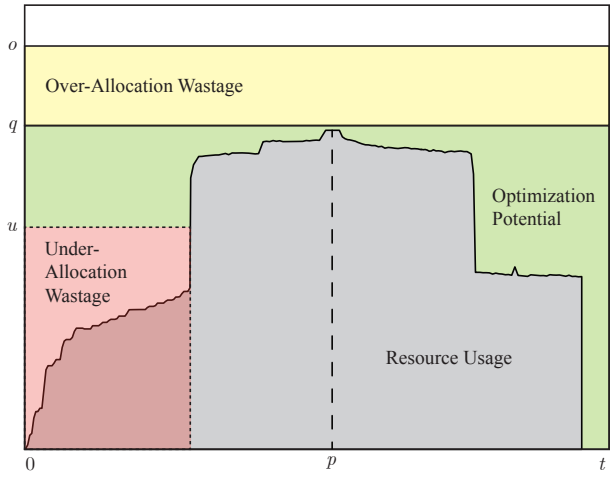


Fig. 1: The figure shows a task's memory usage over time, the optimal, under-, and over-allocation when predicting a single peak memory value, as well as the associated optimization potential.

I. Introduction

Recent years have shown an increasing amount of generated data in all fields of science. For instance, the Square Kilometre Array (SKA) telescopes generate terabytes of raw data per second [1] and the Large Hadron Collider (LHC) retains petabytes of data per year [2]. Due to the increasing amount of data, the manual handling of tasks as a sequence of scripts is no longer feasible, and running them on a single scientist's personal computer is impractical. As a result, workflows are frequently employed to automate, parallelize, and monitor the data analysis process [3]–[5].

Such workflows consist of a set of tasks that act as a wrapper for an arbitrary application and transform data inputs into data outputs. The tasks' execution order is defined by their interdependencies, defining the workflow's dataflow. Due to the amount of data and the associated long runtimes, workflows are frequently executed on large cluster infrastructures administrated by resource managers such as Slurm [6] or Kubernetes [7]. Resource managers rely on resource estimates, e.g., the peak memory or the number of cores, to allocate each task to a suitable machine [8] and to ensure user resource limits. Usually, these estimates are provided by the user and are, therefore, error-prone and inaccurate [9]–[11]. Due to the risk of bottlenecked or even failed task executions caused by assigning too little memory [12], [13], scientists often lean towards overprovisioning memory [14]. This leads to a wastage of resources, reducing the cluster's throughput and increasing the cost [15].

There are methods available for predicting a task's memory requirements without the need for scientists to provide manual estimates. State-of-the-art methods approach this problem by predicting a task's peak memory with analytic methods [15], regression models [11], [16], or reinforcement learning [17]. However, they neglect the fact that a task's resource consumption varies over time. For instance, Figure 1 shows a task's memory consumption function f over time that peaks at point p and flattens out afterward. Allocating q GB of memory leads to a successful task execution as $f(p)$ is the global maximum and $f(p) < q$. However, as memory usage fluctuates, allocating a fixed amount of memory during a task's complete lifetime does not necessarily reflect the task's actual memory usage needs. This observation indicates significant optimization potential currently overlooked by existing static memory estimation approaches.

*equal contribution

Our paper addresses this by leveraging time series monitoring data to predict the memory consumption of scientific workflow tasks over time at a fine-grained level. Our method employs a two-step approach, separating the dynamic memory prediction into distinct runtime and static memory prediction steps. First, we predict a task's runtime with a linear regression model depending on the task's data input size. To this prediction, we add a negative offset in order to prevent task failure due to a delayed increase of memory caused by an expected longer task runtime. Second, we divide the time series into k segments and train k linear regression models, predicting the peak memory usage for each respective segment. To safeguard task executions against prediction errors, we offset predictions with a buffer based on the observed prediction errors.

The contributions of this paper are:

- We propose the k-Segments method that uses time series data to predict a task's memory usage over time. To this end, we separately predict a task's runtime and peak memory values over k segments, merging the results into a memory prediction function.
- We provide a prototype implementation of our method[1] and publish our experimental traces, which showcase task resource usages[2].
- We evaluate our k-Segments method with two real-world workflows consisting of 33 different tasks in total. Our experimental results show an average memory wastage reduction of 29.48% compared to the best state-of-the-art method.

Outline. The remainder of the paper is structured as follows. Section II explains how state-of-the-art methods handle memory prediction for workflow tasks. Section III presents our time series-based memory prediction method. Section IV examines our method through a comparison with state-of-the-art baselines and an analysis of the results obtained. Section V summarizes and concludes our paper.

II. Related Work

This section commences with covering related research on workflow task memory prediction. Subsequently, we provide a comparative analysis, highlighting distinctions between our method and previous approaches.

A. Workflow Task Memory Prediction

This subsection covers research on workflow task memory prediction.

Tovar et al. [15] presented a task sizing strategy for high-throughput scientific workflows. In their work, they propose a model that predicts the peak resource usage for a specific task using an analytical model. The model can be tweaked towards two different objectives: either maximizing throughput or minimizing wastage. Both objectives optimize resource usage under the assumption of a slow-peaks model. The slow-peaks model assumes a worst-case scenario where tasks fail at the end of their execution. The authors suggest using a two-step policy that initially allocates the predicted amount of resources, followed by the maximum available resources in case the first allocation results in an under-allocation. The authors shortly discuss that a multi-step policy is possible but needs further evaluation. Their results show that their approach achieves an overall increase in throughput and a decrease in resource wastage.

Witt et al. [16] used an online feedback loop-based resource allocation method to minimize resource wastage. To learn the resource usage, their predictor uses the sum of the task's input files and trains different linear regression models for peak memory usage prediction. In contrast, Tovar et al. [15] do not use a predictor and base their prediction only on the historical peak usages. Witt et al. dynamically offset the linear regression to achieve an overprediction and avoid task failure through underprovisioning. There are multiple offset strategies proposed. The offset strategy LR mean ± adds the standard deviation as an offset, whereas the LR mean - approach only considers negative prediction errors. The LR max offset strategy adds the largest observed underprediction as an offset. The results yield that their work improves resource utilization and is even able to outperform the work by Tovar et al. [15].

Addressing the memory allocation problem, Witt et al. [11] propose a second method that minimizes the wastage and not the prediction error. The authors assume a relationship between input data size and a task's resource usage and train a linear model based on this assumption. They examine different failure handling strategies, i.e., how to address task failures due to underestimating memory. In their implementation, they decide on a strategy that restarts the task with twice the amount of requested memory. Their evaluation shows that selecting the appropriate failure-handling strategy has a considerable impact on wasted resources.

In our own related work [17], we proposed two different reinforcement learning approaches based on gradient bandits and Q-learning. Both methods have the objective of minimizing resource wastage. Contrary to the previously discussed methods, the two reinforcement learning bandits did not implement an offset technique. We examined actions, state spaces, policies, and reward functions for the two reinforcement learning methods, showing how to circumvent limitations such as discrete action spaces. The evaluation shows that out of the two proposed reinforcement learning methods, the gradient bandit performs better, outperforming the feedback loop-based approach by Witt et al. [16] and significantly reducing memory wastage compared to the default configuration.

B. Comparison with Own Method

In this subsection, we compare the previously related work with our own k-Segments method.

The presented methods predict the peak memory consumption of a single task instance. In contrast, our method predicts k peaks over the time a task is executed. Therefore, we use a new linear model for all k segments in the time series. This can potentially lead to lower memory wastage as a task's memory

[1] https://github.com/dos-group/k-Segments

[2] https://github.com/dos-group/k-Segments-traces

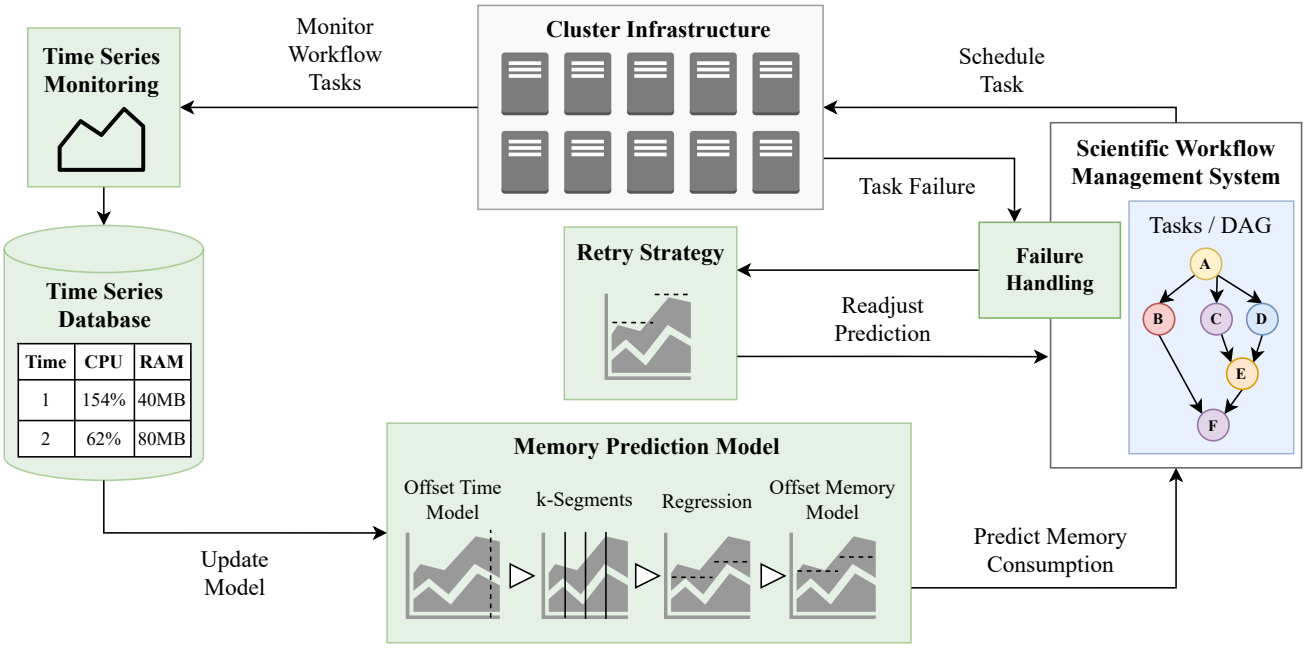


Fig. 2: The figure provides a high-level overview of our k-Segments method (green) as applied in a scientific workflow environment. The model of our method learns time-dependent memory allocations by underpredicting the runtime, dividing a task's time series into k equally distributed segments, learning the peak memory value per segment, and offsetting it. The predictions are updated during the runtime of a workflow execution and provided to the workflow management engine.

usage can be granularly defined but opens the space for more frequent task failures due to underprovisioning memory. Thus, similar to the related work, we employ an offset strategy to avoid such failures. However, we have to employ a more conservative strategy to avoid task failures, using the largest historical prediction error. In addition, we need to consider offsetting only failed segments or each segment.

III. Approach

This section provides a comprehensive overview of our method, including a detailed description of its runtime and memory prediction components, their integrated application for dynamic memory consumption prediction, and our approach to addressing prediction failures.

A. Overview

Figure 2 provides an overview of the execution environment in which our method is applied. The scientific workflow management system (SWMS) is responsible for submitting the workflow tasks to the cluster infrastructure. During the execution of the workflow tasks, our monitoring component collects time series monitoring data such as memory usage, CPU usage, or file events and stores it in a database. On task completion, our method retrieves the monitoring data from the database and builds or updates its task resource prediction model. The model partitions the time series into k segments and trains a linear regression model for each segment, predicting the segment's memory usage. Whenever the SWMS submits a known workflow task, it receives a predicted resource allocation function for the respective task from our model. Our method works in an online fashion, meaning that the task resource prediction model is trained during the execution of the workflow. This especially benefits workflow and task executions without historical data, as an offline method cannot provide resource predictions in such cases, leading to the use of user defaults.

B. k-Segments Model Creation

Figure 3 shows the model creation steps, which we cover in the following in detail. Contrary to the traditional approaches that predict peak memory usage, our time series-based method needs to take into account differences in task runtimes. Therefore, as a first step, our model predicts the task's expected runtime. We use a linear regression that assumes a relationship between the task's input size and its runtime. Given a dataset $D = \{(x_1, Y_1), ..., (x_n, Y_n)\}$ of a single task type, where x is a scalar representing the total size of the input file and Y is a list representing the memory usage over time of a single task execution. Then, a linear model can be trained with the total input file size x as the independent variable and the runtime r as the dependent variable, where $r = j \cdot f$, $j = length(Y)$, and f is equals to the length of the monitoring interval. Then, we subtract the largest negative historical prediction error of our prediction as an offset.

Next, we define $k - 1$ change points that are evenly distributed across the time series, creating k segments in total. Again, given a dataset $D = \{(x_1, Y_1), ..., (x_n, Y_n)\}$ of a task type, where x is a scalar, the total size of the input file, and Y is

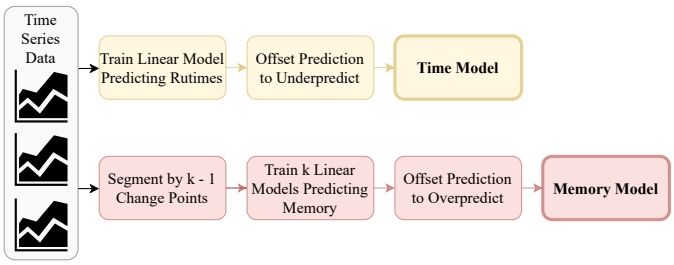


Fig. 3: The figure shows the prediction model creation steps divided into runtime prediction and memory prediction

a list of the memory usage over time of a single task execution. Each Y of size j, $j = length(Y)$, can be transformed into a segmented time series Y^* with k segments using $k-1$ change points in form of:

$$Y^* = (s_1, ..., s_k) = ((y_1, ..., y_i), (y_{i+1}, ..., y_{2i}), ...,$$

$$(y_{(k-2)i+1}, ..., y_{(k-1)i}), (y_{(k-1)i+1}, ..., y_j)), \text{ where } i = \lfloor \frac{j}{k} \rfloor.$$

Then, we have to provide a function that returns a segment's memory usage. Therefore, we use the transformed dataset $D^* = \{(x_1, Y_1^*), ..., (x_n, Y_n^*)\}$ and calculate the peak for each datapoint's segment, i.e., $Y^{**} = (max(s_1), ..., max(s_k))$. This simplification enables us to train k linear regressions, one for each segment, as the segment values are now scalar. After training these regressions, we add the largest positive prediction error from historical executions on the regressions' intercepts to avoid underpredictions.

C. Time Series Memory Prediction

After the models are trained, they can be used to predict a resource allocation function for the next task execution. First, we obtain a task's input data size from the SWMS and use it to train the time and the memory model. Then, we predict the task's expected runtime r_e with the time model and split it into k values $R = (r_1 = r_s, r_2 = 2r_s, ..., r_{k-1} = (k-1)r_s, r_k = r_e)$, where $r_s = \lfloor \frac{r_e}{k} \rfloor$. The memory model predicts k new resource allocations $V = (v_1, ..., v_k)$ where $v > 0$. If a segment's predicted memory value is smaller than the previous segment's memory prediction, i.e., $v_{k-1} > v_k$, we take the previous segment's memory prediction. For predictions where $v_1 < 0$, we set the default value. This ensures that the function increases monotonically. By combining the two sets of data, the following monotonically increasing step function can be constructed:

$$f(x) = \begin{cases} v_1, & \text{if } 0 \leq x \leq r_1 \\ v_2, & \text{if } r_1 < x \leq r_2 \\ \vdots & \\ v_k, & \text{if } r_{k-1} < x \leq r_k \end{cases} \quad (1)$$

Figure 4 shows such a step function with 4 segments applied to the real-world adapter removal workflow task.

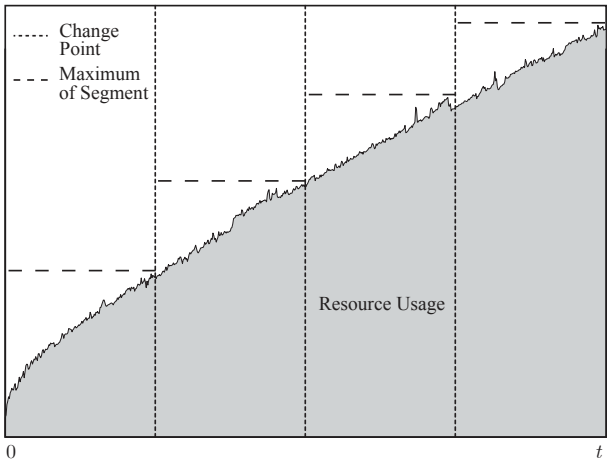


Fig. 4: Example of applying our k-Segments method to the adapter removal task with $k = 4$.

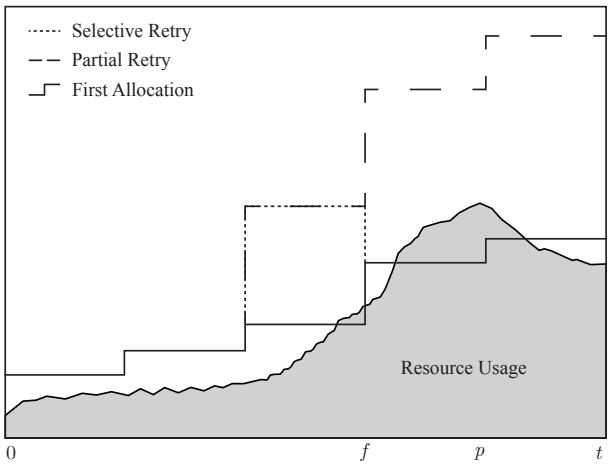


Fig. 5: The figure illustrates the selective and partial retry strategies and their effect on a retried task execution.

D. Failure Handling

Allocating too little memory will cause the task to fail, and the task must be retried. This can happen even if we use offsetting strategies. Therefore, we propose two different failure-handling strategies. First, the **Selective Retry Strategy** that adjusts only the failed segment. Second, the **Partial Retry Strategy** that adjusts every segment starting from the one that caused the failure. When adjusting the segment(s), we have to select the new memory values. Therefore, we define the retry factor l, which is multiplied by the failed allocation v to provide the new allocation v_{new}. Figure 5 shows the two different failing strategies with a retry factor of $l = 2$. Here, a selective retry strategy would again lead to a task failure as the fourth segment still underpredicts the memory.

In general, the efficacy of failure strategies depends on the task's memory consumption, and there is no overall superior strategy [9], [11]. In our time series-based method, also the number of upcoming segments and their chance of failure

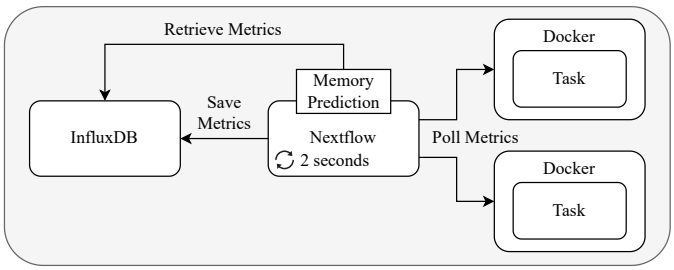


Fig. 6: The figure provides an overview of our prototype implementation and its interaction with the existing workflow management system Nextflow.

affect the efficacy of the strategy. For instance, if only one segment would cause failure, which we do not know in advance, the selective retry strategy would perform better. Lastly, we would like to mention that there is also the possibility of adjusting a task's predicted runtime instead of the predicted memory, but this type of failure strategy is not considered in our work.

IV. Evaluation

Our evaluation section is divided into five key subsections: Prototype Implementation, Experimental Setup, Baseline Methods, Results, and Discussion. These components collectively provide a thorough assessment of our method's performance and its implications in the context of memory prediction for scientific workflow tasks.

A. Prototype Implementation

Figure 6 provides a simplified overview of our implementation. We developed our method as an extension for the SWMS Nextflow [18]. By default, Nextflow collects peak and average resource usage values. We extended this with a time series monitoring component that uses InfluxDB to store periodic metrics. Our monitoring extension uses the Docker API to extract measurements. Docker uses cgroups, a Linux kernel feature that allows processes to be organized in groups that can then be limited and monitored [19], [20]. Our monitoring extension accesses the cpuacct, memory, and blkio controller from the Linux kernel via the Docker API to retrieve the monitoring information. Because our method utilizes these low-level metrics, it is also portable to non-containerized setups. The length of the monitoring interval can be adjusted and comes with a default of two seconds. We observed that two seconds does not significantly impact the performance while still being able to provide accurate measurements. Lowering the interval length involves the risk of overlooking memory peaks. In addition to monitoring the cgroups, we also monitor the files, providing information such as the number of input files or total input size. The gathered information is then retrieved from the database by our memory predictor. The memory predictor uses Python with NumPy and scikit-learn to build its models and keeps them in memory. As defaults for our method, we use $k = 4$ for the number of segments, a retry factor of $l = 2$, and 100MB as the minimum amount of memory to allocate in case the model predicts an allocation of less than zero.

Please note that Figure 6, the implementation, refers to a simplified setup on a single node. On a cluster, each node would provide a monitoring service that could store the data directly in the database, making it available to Nextflow and the memory predictor.

B. Experimental Setup

For our experiments, we run two real-world workflows from the nf-core repository [21], namely eager [22] and sarek [23], [24]. For the workflow inputs, we used data available from two studies. The eager workflow input data is from a study published in 2018 by de Barros Damgaard et al. on the population history of the Eurasian steppe [25]. The input data for the sarek workflow is from a study published in 2022 by Harrod et al. [26]. The sarek workflow ran for approximately one day and 5 hours, has 29 different tasks with an average runtime from 2 seconds to 1 hour, and an average peak memory usage from 10 MB to about 23 GB. The eager workflow ran for three days and 12 hours, contains 18 different tasks with an average runtime between 8 seconds and 4 hours, and a peak memory usage from 19MB to 14GB. The sarek workflow contains up to 1512 executions of the same task, and the eager workflow up to 136 executions of the same task. For gathering the metrics, we run the workflows with our prototype implementation on a machine consisting of an AMD EPYC 7282 16-Core Processor (32 Threads), 128GB DDR4 memory, and two 960GB SATA III SSDs.

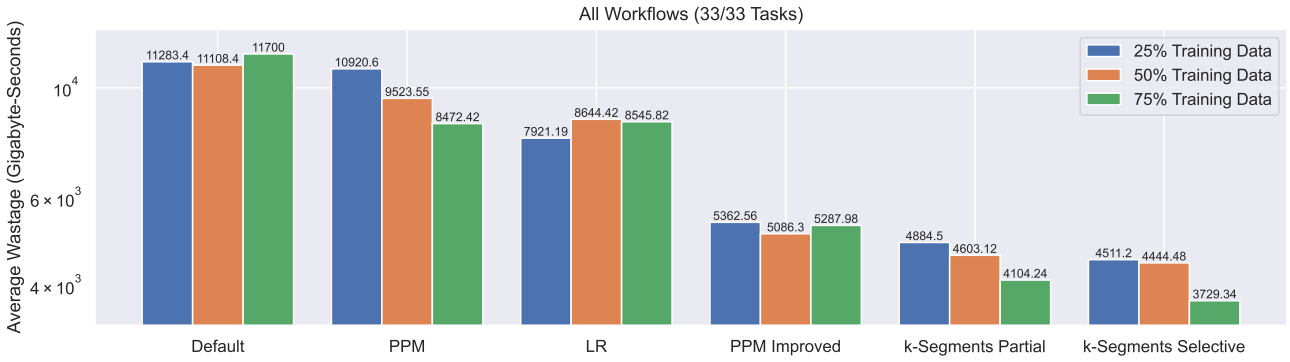
We use the gathered metrics to feed our simulation tool. Through simulation parameters, the proportion of training/test data can be defined. We simulate an online approach where finished task executions can be incorporated into the learning process respective to real-world systems. This is also an important aspect as we want to incorporate state-of-the-art baselines that can use online learning.

C. Baselines

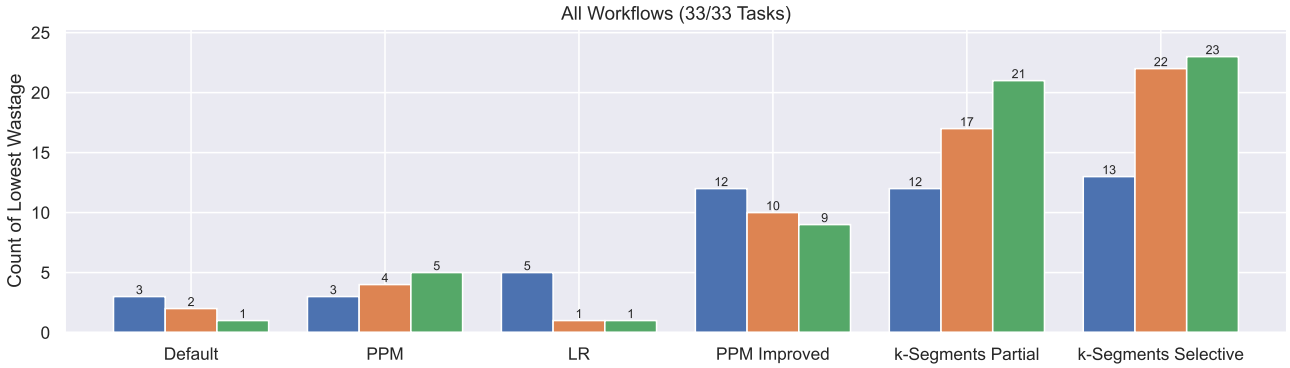
We compare our method with two state-of-the-art baselines, one state-of-the-art baseline that we have improved, and the default configurations of the workflows.

The workflows' default configurations are the ones provided by the workflow developers. They are used when running the workflows out of the box and, therefore, serve as a sanity baseline.

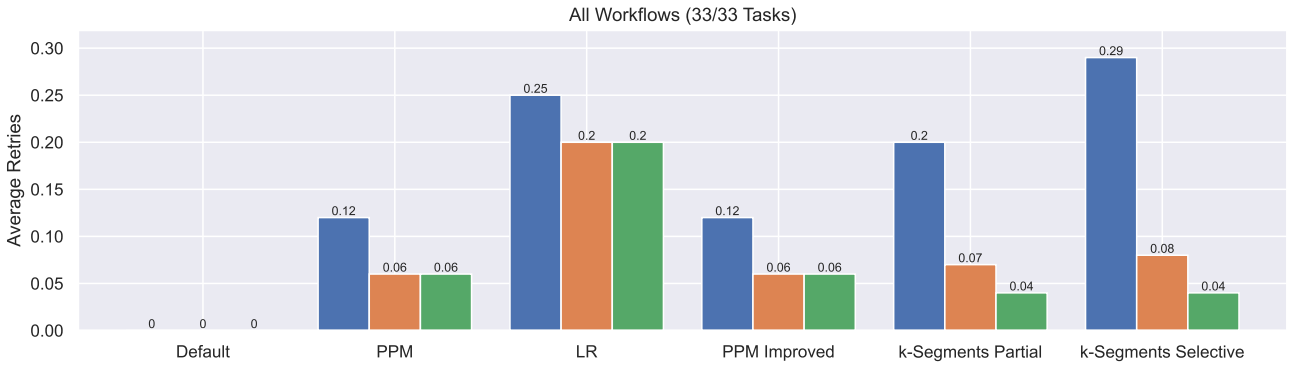
Tovar et al. [15] use probabilities of peak memory values (PPM) from historical executions to select a task's initial amount of peak memory. Their method aims to minimize the sum of probabilities of resource peaks where the resource peak is greater than the allocated resource value. If the first resource allocation leads to an underprediction and the task fails, Tovar et al. assign a node's maximum amount of memory. We use the source code provided by the authors and integrate it into our simulation. As an additional baseline, we extend the method of Tovar et al. and implement a failure strategy that doubles the



(a) The figure compares the average wastage in GBs of our k-Segments-based methods to two state-of-the-art baselines, one state-of-the-art baseline improved by us, and the default configurations of the workflows across all 33 workflow tasks.



(b) The figure compares the number of times a method achieves the lowest wastage across all 33 workflow tasks.



(c) The figure compares the average retries of our k-Segments-based methods to two state-of-the-art baselines, one state-of-the-art baseline improved by us, and the default configurations of the workflows across all 33 workflow tasks.

Fig. 7: The figures show the experimental results.

memory upon failure instead of assigning a node's maximum memory immediately. We name this baseline PPM Improved.

Witt et al. [16] use an online learning method that uses a linear regression model (LR) to estimate a task's peak memory, assuming a linear relationship between data input size and memory usage. As an offset, they add the standard deviation based on historical predictions. The authors assign double the amount of memory for failed tasks and execute them again. Due to the lack of provided links to the source code, we had to implement the method according to the descriptions in the paper.

D. Results

Figure 7 shows the results of the memory prediction with three different amounts of training data for 33 workflow tasks from the two workflows.

Figure 7a presents the wastage of all approaches in gigabyte-seconds. Among these, the default baseline exhibits the highest wastage, approximately 2.5 to 3 times higher than the best-performing method, k-Segments Selective. Comparatively, the literature baselines manage to outperform the default but still demonstrate significantly higher wastage compared to our k-Segments methods. Across all levels of training

data, k-Segments Selective consistently achieves the lowest wastage, closely followed by k-Segments Partial. The k-Segments models demonstrate a wastage reduction of 22.39% for the partial retry strategy and 29.48% for the selective retry strategy compared to the best-performing baseline, PPM Improved, using 75% of the training data. Noticeably, while our methods, k-Segments Selective and k-Segments Partial, exhibit lower wastage with increasing training data, the same isn't necessarily true for the baselines. Some baselines exhibit slight performance degradation when provided with more data.

Figure 7b shows the number of times a method scores the lowest wastage. If two methods both have the least wastage, they both get one point. Using 25% of the training data, the k-Segments Selective method has the highest count, closely followed by k-Segments Partial and PPM Improved by Tovar et al., which yield identical counts. For 25% training data, PPM and the default baseline exhibit the lowest values. For 50% and 75% training data, LR by Witt et al. and the default baseline similarly display the lowest values. Increasing the amount of training data for PPM Improved and the k-Segments methods shows an increase in counts for the k-Segments methods and a decrease for PPM Improved, again indicating that more training data positively affects our k-Segments methods.

Figure 7c shows the methods' average number of retries. The default strategy has an average of zero, indicating no task failures due to insufficient memory. Using 25% of the training data, our k-Segments Selective method has the highest number of failures, followed by the LR and the k-Segments Partial method. PPM (Improved) shows the lowest average number of failures for a prediction method. By increasing the training data to 50%, PPM (Improved) and LR can reduce the average number of failures but show no improvement when providing more than 50% of the training data. Our k-Segments models are able to significantly reduce the average number of retries, even showing the lowest number achieved by any prediction method when using 75% of the data.

(a) Qualimap task

(b) Adapter removal task

Fig. 8: The figures show the memory wastage of the tasks using our method as a function of the chosen parameter k.

E. Discussion

The evaluation results show that using our method with time series data to predict a task's memory usage leads to lower wastage. Additionally, our method benefits from a growing quantity of training data, whereas the state-of-the-art methods do not necessarily exhibit improvements. The increase in training data positively influences our method's retry count, thereby accounting for the lower wastage.

An unexpected outcome of the results is the substantial reduction in memory wastage observed using PPM Improved compared to its original implementation. On average, PPM Improved significantly outperforms Witt et al.'s LR method. We attribute this discrepancy to the improved failure-handling strategy, which was not assessed in Tovar et al.'s original paper but appears to yield superior outcomes. Due to the machines in our experimental setup being equipped with 128GB of memory, directly assigning the maximum amount of memory to a task upon failure leads to significant memory wastage.

One of our method's limitations is the selection of the parameters, most importantly k. In general, selecting a bigger k can decrease the wastage but leads to possibly more error-prone assignments due to prediction errors that eventually increase the wastage. For our experiments, we tested several values for k and chose $k = 4$ for all tasks. However, k can also be selected for each task individually, posing a local optimization problem. For instance, Figure 8 presents two tasks from the eager workflow and the average wastage in relation to the chosen k when using 50% of training data. The Qualimap task has a zigzag pattern with multiple local optima and a global optimum at $k = 9$. In contrast, the Adapter Removal task reduces its wastage up to $k = 13$, making it easier to search for a global optimum. As zigzag patterns such as in Figure 8a make it hard to deploy gradient-based search methods, reoptimizing k on each iteration during online learning appears to be an option for this [11].

A second limitation of our method is its applicability in present-day systems. Resource managers now require a memory estimate for the entire duration of a job's runtime while we provide an estimate over time. This necessitates that the allocated resources can be modified over time to derive benefits from the fine-grained predictions. Additionally, scheduling must account for dynamic changes to memory over time.

V. Conclusion

In this paper, we presented our novel k-Segments method that predicts a workflow task's memory usage over time. To this end, our method predicts a task's runtime and divides the predicted time into k equally-sized segments. Then, our method predicts each segment's memory consumption depending on the task's input data size. Using prediction offsets, the k-Segments model aims to avoid underallocations during the prediction process.

In our evaluation with two real-world scientific workflows from the publicly available nf-core repository, we show that our k-Segments method is able to outperform two state-of-the-art baselines, achieving a memory wastage reduction of 29.48% compared to the best competitor. Furthermore, our experiments illustrate that our method benefits significantly from an increasing amount of training data, with less wastage and fewer failures, compared to the state-of-the-art baselines.

Our method requires the number of segments k as an input, which can vary between tasks. In the future, we aim to explore methods of finding k by using exploration and exploitation techniques from the field of reinforcement learning. Additionally, our method's predictions are dynamic, requiring resource managers to support adjustments during runtime. As this is not the current norm, we want to extend an existing resource manager's interface to cope with dynamic adjustments of memory claims.

Acknowledgments

Funded by the Deutsche Forschungsgemeinschaft (DFG, German Research Foundation) as FONDA (Project 414984028, SFB 1404).

References

[1] J. S. Farnes, B. Mort, F. Dulwich, K. Adámek, A. Brown, J. Novotny, S. Salvini, and W. Armour, "Building the world's largest radio telescope: The square kilometre array science data processor," in *2018 IEEE 14th International Conference on e-Science (e-Science)*. IEEE, 2018.

[2] M. Barisits, T. Beermann, V. Garonne, T. Javurek, M. Lassnig, C. Serfon, A. Collaboration *et al.*, "The atlas data management system rucio: Supporting lhc run-2 and beyond," in *Journal of Physics: Conference Series*, vol. 1085, no. 3. IOP Publishing, 2018.

[3] R. F. da Silva, R. M. Badia, V. Bala, D. Bard, P.-T. Bremer, I. Buckley, S. Caino-Lores, K. Chard, C. Goble, S. Jha *et al.*, "Workflows community summit 2022: A roadmap revolution," *arXiv preprint arXiv:2304.00019*, 2023.

[4] J. Bader, J. Witzke, S. Becker, A. Lößer, F. Lehmann, L. Doehler, A. D. Vu, and O. Kao, "Towards advanced monitoring for scientific workflows," in *Big Data*. IEEE, 2022.

[5] E. Deelman, T. Peterka, I. Altintas, C. D. Carothers, K. K. van Dam, K. Moreland, M. Parashar, L. Ramakrishnan, M. Taufer, and J. Vetter, "The future of scientific workflows," *The International Journal of High Performance Computing Applications*, vol. 32, no. 1, 2018.

[6] A. B. Yoo, M. A. Jette, and M. Grondona, "Slurm: Simple linux utility for resource management," in *Workshop on job scheduling strategies for parallel processing*. Springer, 2003.

[7] B. Burns, B. Grant, D. Oppenheimer, E. Brewer, and J. Wilkes, "Borg, omega, and kubernetes," *Communications of the ACM*, 2016.

[8] 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 *CCGrid*, 2023.

[9] 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 *2021 IEEE Workshop on Workflows in Support of Large-Scale Science (WORKS)*. IEEE, 2021.

[10] 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, 2012.

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

[12] A. Lößer, J. Witzke, F. Schintke, and B. Scheuermann, "Bottlemod: Modeling data flows and tasks for fast bottleneck analysis," in *2022 IEEE International Conference on Big Data (Big Data)*. IEEE, 2022.

[13] A. M. Kintsakis, F. E. Psomopoulos, and P. A. Mitkas, "Reinforcement learning based scheduling in a workflow management system," *Engineering Applications of Artificial Intelligence*, vol. 81, 2019.

[14] 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 *2022 IEEE International Parallel and Distributed Processing Symposium (IPDPS)*. IEEE, 2022.

[15] 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.

[16] C. Witt, D. Wagner, and U. Leser, "Feedback-based resource allocation for batch scheduling of scientific workflows," in *2019 International Conference on High Performance Computing & Simulation (HPCS)*. IEEE, 2019.

[17] J. Bader, N. Zunker, S. Becker, and O. Kao, "Leveraging reinforcement learning for task resource allocation in scientific workflows," in *2022 IEEE International Conference on Big Data (Big Data)*. IEEE, 2022.

[18] P. Di Tommaso, M. Chatzou, E. W. Floden, P. P. Barja, E. Palumbo, and C. Notredame, "Nextflow enables reproducible computational workflows," *Nature biotechnology*, vol. 35, no. 4, 2017.

[19] Y. Al-Dhuraibi, F. Paraiso, N. Djarallah, and P. Merle, "Autonomic vertical elasticity of docker containers with elasticdocker," in *2017 IEEE CLOUD*. IEEE, 2017.

[20] Z. Zhuang, C. Tran, J. Weng, H. Ramachandra, and B. Sridharan, "Taming memory related performance pitfalls in linux cgroups," in *2017 ICNC*. IEEE, 2017.

[21] 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, 2020.

[22] J. A. F. Yates, T. C. Lamnidis, M. Borry, A. A. Valtueña, Z. Fagernäs, S. Clayton, M. U. Garcia, J. Neukamm, and A. Peltzer, "Reproducible, portable, and efficient ancient genome reconstruction with nf-core/eager," *PeerJ*, vol. 9, 2021.

[23] F. Hanssen, M. U. Garcia, L. Folkersen, A. S. Pedersen, F. Lescai, S. Jodoin, E. Miller, O. Wacker, N. Smith, nf-core community *et al.*, "Scalable and efficient dna sequencing analysis on different compute infrastructures aiding variant discovery," *bioRxiv*, 2023.

[24] M. Garcia, S. Juhos, M. Larsson, P. I. Olason, M. Martin, J. Eisfeldt, S. DiLorenzo, J. Sandgren, T. D. De Ståhl, P. Ewels *et al.*, "Sarek: A portable workflow for whole-genome sequencing analysis of germline and somatic variants," *F1000Research*, vol. 9, 2020.

[25] P. d. B. Damgaard, N. Marchi, S. Rasmussen, M. Peyrot, G. Renaud, T. Korneliussen, J. V. Moreno-Mayar, M. W. Pedersen, A. Goldberg, E. Usmanova *et al.*, "137 ancient human genomes from across the eurasian steppes," *Nature*, vol. 557, no. 7705, 2018.

[26] A. Harrod, C.-F. Lai, I. Goldsbrough, G. M. Simmons, N. Oppermans, D. B. Santos, B. Győrffy, R. C. Allsopp, B. J. Toghill, K. Balachandran *et al.*, "Genome engineering for estrogen receptor mutations reveals differential responses to anti-estrogens and new prognostic gene signatures for breast cancer," *Oncogene*, vol. 41, no. 44, 2022.