KS+: Predicting Workflow Task Memory Usage Over Time

Jonathan Bader*‡, Ansgar Lößer*†, Lauritz Thamsen§, Björn Scheuermann†, and Odej Kao‡
‡ {firstname.lastname}@tu-berlin.de, Technische Universität Berlin, Germany
§ {firstname.lastname}@kom.tu-darmstadt.de, Technische Universität Darmstadt, Germany
§ lauritz.thamsen@glasgow.ac.uk, University of Glasgow, United Kingdom

***Abstract*—Scientific workflow management systems enable the reproducible execution of data analysis pipelines on cluster infrastructures managed by resource managers such as Kubernetes, Slurm, or HTCondor. These resource managers require resource estimates for each workflow task to be executed on one of the cluster nodes. However, task resource consumption varies significantly between different tasks and for the same task with different inputs. Furthermore, resource consumption also fluctuates during a task's execution. As a result, manually configuring static memory allocations is error-prone, often leading users to overestimate memory usage to avoid costly failures from under-provisioning, which results in significant memory wastage.**

We propose KS+, a method that predicts a task's memory consumption over time depending on its inputs. For this, KS+ dynamically segments the task execution and predicts the memory required for each segment. Our experimental evaluation shows an average reduction in memory wastage of 38% compared to the best-performing state-of-the-art baseline for two real-world workflows from the popular nf-core repository.

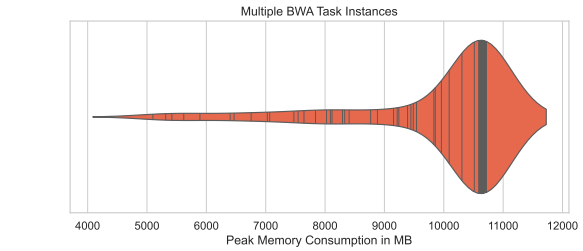
***Index Terms*—Resource Management, Scientific Workflows, Machine Learning, Memory Prediction, Cluster Computing**

I. Introduction

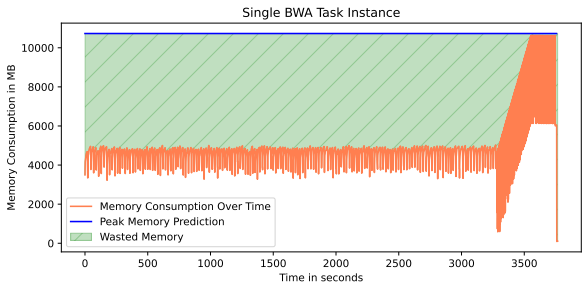
As the amount of scientific data available increases, it is becoming impractical for scientists to analyze their datasets manually using sequences of custom scripts. Scientists, therefore, increasingly rely on scientific workflow management systems (SWMS) to help them define, execute, orchestrate, and monitor their data analysis workflows [1]–[3]. Such data analysis workflows typically consist of a set of interdependent analysis steps, called tasks, whose execution order is defined by data dependencies. Thousands of instances of the same task can be executed with different data inputs when executing a workflow on large datasets [4]–[6]. It is impractical to run such a large number of tasks on a scientist's personal computer. Thus, SWMS enable interoperability with various resource managers such as Slurm [7], HTCondor [8], or Kubernetes [9] to execute the data analysis workflow on a cluster infrastructure.

Resource managers require memory usage limits for each task instance. Exceeding these limits will result in costly bottlenecks or, more often, task termination [10]–[12]. Thus, scientists are effectively incentivized to overestimate resource consumption to avoid timely re-executions of tasks or even complete workflows. However, requesting more memory than needed wastes precious cluster resources and limits the throughput on both a workflow and a cluster level.

*equal contribution



(a) Distribution of peak memory consumption of multiple BWA task executions. Lines show observations.



(b) A BWA task's memory consumption over time.

Fig. 1: Variability of memory consumption a) for the Burrows-Wheeler Aligner (BWA) task with different inputs and b) for a single BWA task over time during execution.

The problem of setting accurate resource limits for tasks is further aggravated by the fact that the same task consumes different amounts of memory for different inputs, often depending on the size of the input data. Figure 1a shows the distribution of peak memory consumption for the Burrows-Wheeler Aligner (BWA) task — a popular tool for aligning short DNA sequences — when running the eager workflow [13]. It can be observed that the median memory consumption is around 10600 MB. However allocating the median memory usage for all BWA tasks would result in considerable memory wastage for half of the tasks, while the other half would fail. In addition, most tasks do not maintain a constant memory usage throughout their execution. For example, a task may wrap multiple existing programs, resulting in varying memory consumption patterns as each

of the programs is executed. Even when a task wraps only one program, memory usage often fluctuates. A common example is a task that first loads input data into memory before processing it. An instance of such non-static memory consumption can be seen in Figure 1b. Here, the BWA task uses about 5.1 GB of memory for about 80% of its runtime, before the memory usage more than doubles to finally around 10.7 GB. Simply allocating a fixed value of 10.7 GB would work, but would waste substantial memory as highlighted in green in the figure.

Currently, state-of-the-art workflow memory prediction methods only estimate a task's peak memory usage. To do this, many methods use machine learning techniques, such as regression models [14], [15], reinforcement learning [16], or ensemble methods that combine multiple machine learning models [17], [18]. Recently, we proposed the k-Segments method [19], which predicts workflow task memory consumption over time by leveraging time series monitoring data. Our method estimates a task's runtime, divides it into k equally sized segments, and predicts the memory consumption for each segment based on the data input size. However, as can be seen in Figure 1b, equally sized segments are not necessarily able to model a task's memory consumption over time accurately.

In this paper, we present KS+, an extension of our previously proposed method, which works as follows: First, KS+ dynamically sizes the segments and predicts peak memory values for each segment. KS+ ensures that the prediction function is monotonically increasing, preventing task failures caused by reducing memory too early. Subsequently, KS+ offsets the predictions to improve reliability. In addition, if a task fails, KS+ adaptively adjusts the segments' position and the predicted memory before re-execution.

The contributions of this paper are:

- We introduce KS+, which dynamically predicts the memory consumption of workflow tasks by dynamically estimating the position, length, and memory consumption for individual task segments. The source code is publicly available[1].
- We propose a novel task failure strategy for dynamic task memory prediction that re-adjusts not only the memory but also the segments for task re-execution.
- We evaluate our KS+ method on traces from two real-world workflows, using different amounts of training data, and show for example an average memory wastage reduction of 38% compared to the original k-segments method and 51% compared to the best-performing peak memory prediction baseline.

II. Approach

To predict the memory consumption of tasks with variable-size segments over time, we first need a segmentation strategy for a single historical task execution. The segment parameters, segment start, segment end, and peak memory value per segment, of multiple executions can then be used to build a model that can predict memory usage over time. Figure 2 shows the advantage of using two dynamically sized segments (green line) compared to fixed-sized segments (blue line). Variable-size segments are able to better model a task's memory consumption and allow for sophisticated retry strategies that involve re-setting the segments for re-execution.

[1] github.com/dos-group/KSPlus

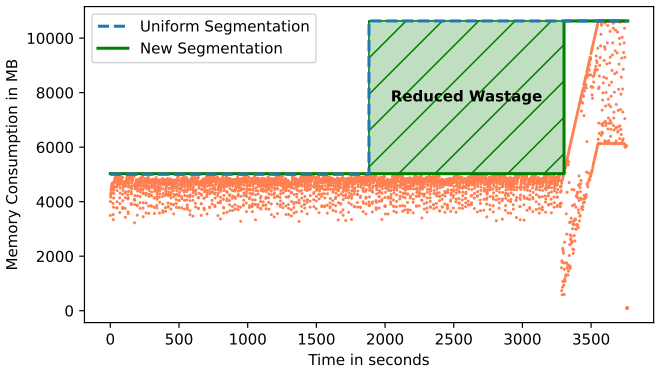


Fig. 2: Memory consumption over time for a single execution of the BWA task. The dashed blue line represents the prediction with two uniform segments. The green line uses two segments with variable size, generated with our KS+ method.

Algorithm 1 Segmentation Algorithm

Require:
$M \leftarrow$ array of memory values
$k \leftarrow$ number of output segments
function GetSegments($M[\,], k$)
  $S \leftarrow [1]$
  $P \leftarrow [M_0]$
  for $i \leftarrow 1$ to $Length(M)$ **do**
    if $M_i \geq P_{-1}$ **then**
      $S_{-1} \leftarrow S_{-1} + 1$
    else
      $Append(P, M_i)$
      $Append(S, 1)$
    end if
  end for
  while $Length(P) > k$ **do**
    $e \leftarrow [(P_{i+1} - P_i) \cdot S_i]$
    $idx \leftarrow MinElement(e)$
    $S_{idx+1} \leftarrow S_{idx} + S_{idx+1}$
    $Remove(S, idx)$
    $Remove(P, idx)$
  end while
  return (S, P)
end function

A. Dynamic Segment Placement

We model the memory consumption of a single historical task execution through a step function with k segments, where the i-th segment $1 \leq i \leq k$ is defined by the peak memory consumption P_i in the respective runtime interval of duration

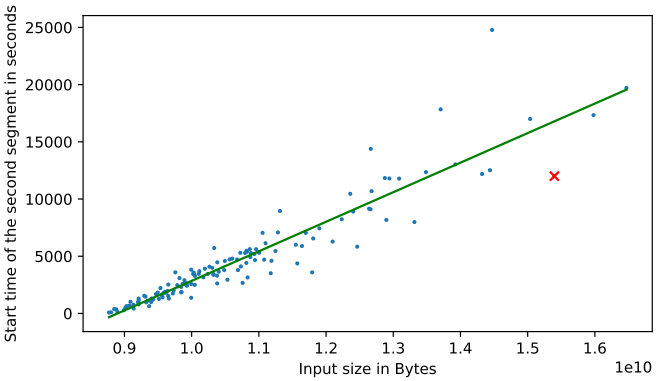


Fig. 3: Start time for the second segment for all BWA task executions. The blue points represent individual task executions, the green line is the estimation, minimizing the mean squared error. The deviation gets larger for bigger input sizes. The red cross marks an execution that ran much faster than expected.

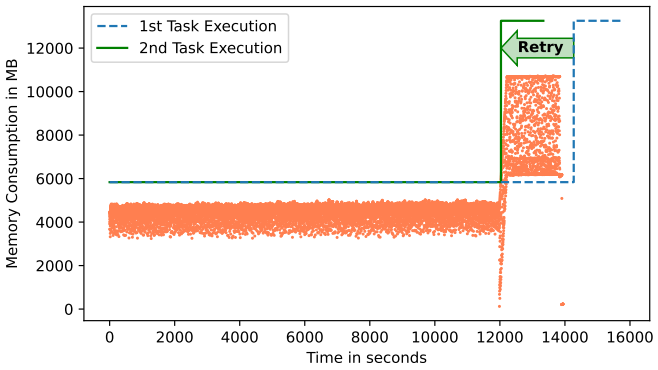


Fig. 4: Memory consumption over time of the BWA task execution marked in Figure 3. The dashed blue line represents the predicted KS+ allocation strategy, that fails due to the execution reaching the second segment faster than expected. The green line shows the allocation for the second try.

S_i. We model the memory consumption as monotonically increasing, i.e. for each segment it must hold that $P_i \leq P_{i+1}$, to avoid task failures caused by reducing memory too early. The resulting function should define the segments in such a way that the wastage is minimized, i.e., the individual measurement points must be as close as possible to the modeled peak of the corresponding segment, but not higher, as this would result in a task failure.

We developed a greedy segmentation algorithm, which is shown in Algorithm 1. The algorithm proceeds in two steps. First, every data point is considered to be a segment by itself. The peak memory consumption corresponds to the value of the data point and the size of the segment is one. We first merge every segment with its predecessor, if the peak value of the segment is smaller than the peak value of the preceding segment. This is done front to back until the constraint of being monotonically increasing is fulfilled. In the second step, we further reduce the number of segments. To decide which segments to merge, we look at the increase of the error e_i, which is introduced by merging the i-th segment with its successor:

$$e_i = (P_{i+1} - P_i) \cdot S_i \tag{1}$$

We always merge the segment S_i with the smallest merging error e_i, until only k segments are left. This results in a fast heuristic to find the segments and the corresponding peak values. An example of the resulting segmentation is shown in Figure 2.

B. Segment Prediction

In order to be able to predict the segments and their memory consumption for task execution, Algorithm 1 is used to gather the k segment parameter pairs (S_i, P_i) for multiple executions of the same task. As shown in [4], [14], [15], [20], [21], the accumulated input file size of all input files is a good indicator for the overall execution time and the peak memory consumption of a task.

We use linear regression based on the input file size I_t for the task execution t to estimate the peak memory consumption $\hat{P}_{t,i}(I_t)$ as well as the start positions for each segment $\hat{S}_{t,i}(I_t)$, resulting in two individual linear regression models for each segment. These models can then be used to estimate a memory allocation strategy for new executions of the same task with a different input size. The segment times can scale differently with the input sizes, therefore it is advantageous to use distinct predictions for each segment. For instance, the execution time of the first process of a task might scale linearly with the input size, while the second process might always take a constant amount of time, resulting in very different time ratios between these two processes for different input sizes.

It is difficult to accurately predict the starting times of the different segments, especially for long-running tasks. This is due to uncertainty in resource allocation. For example, a lack of CPU resources caused by other tasks running on the same machine may change the task behavior over time, resulting in slower task execution. As shown in Figure 3, this makes long-running tasks more susceptible to a larger misprediction. Since underpredicting the memory results in the restart of a task, it is generally less expensive to overpredict than to underpredict the memory. Therefore, we added a safety margin by overpredicting the memory peaks by 10% and underpredicting the segment start times by 15%. Since our memory allocation strategy is monotonically increasing, underpredicting the segment start times is always the safer option.

C. Retry Strategy

The introduction of dynamic segment sizes enables more flexible retry strategies in case a task execution runs out of memory. Due to the uncertainty in predicting the segment sizes, it is much more likely that the peak memory consumption was predicted correctly, but the execution reached a more demanding segment earlier than expected. Instead of doubling

the memory allocation upon failure, like most state-of-the-art approaches, we focus on changing the allocation timing.

As soon as a task execution runs out of memory, which is detected, for instance, by the Linux OOM Killer, we check the current runtime of this execution. This point in time is compared to the expected start time of the next segment. The start times of all succeeding segments are reduced by a factor so that the next segment would have started when the execution ran out of memory. This approach is visualized in Figure 4. Only if the point of failure is already in the last segment, the peak of the segment is increased by 20%. If the task fails again, this procedure is repeated until the execution finishes successfully. This strategy utilizes the knowledge about when the execution ran out of memory to drastically reduce the uncertainty in the segment start time prediction for the current execution.

III. Evaluation

Our evaluation section contains the experimental setup, briefly explains the baselines, and shows the experimental results.

A. Experimental Setup

For our experiments, we use the data published alongside the original k-Segments method [19]. The data consists of two nf-core [22] workflows, eager [13] and sarek [23]. Eager uses input data from a 2018 study that examines the population history in the area of the Eurasian steppe [24]. The sarek workflow was run with input from a 2022 study, which investigated estrogen receptor mutations in breast cancer [25]. Both workflows were executed on a machine equipped with an AMD EPYC 7282 processor and 128GB DDR4 memory.

Figure 5 provides an overview of the two workflows and their peak memory consumption. It can be observed that the sarek workflow has more task instances and an average peak memory usage of 1.67 GB. In contrast, the eager workflow contains fewer task instances but has an average peak memory usage of 2.31 GB.

We run the experiments ten times, each time with a different seed that is used to split the train/test data. The values given are averages over all runs. For wastage, the metric used is gigabyte seconds (GBs), which accounts for memory wastage over time. The wastage for a single task execution can be defined as the difference between requested and used memory over time plus the sum of allocated memory over time from its failed task executions and thus also accounts for task failures.

B. Baselines

We compare our KS+ method with four state-of-the-art methods from the literature and the workflow developers' default as a sanity baseline.

Tovar et al. [26] predict a task's peak memory using historical peak memory probabilities (*Tovar-PPM*). If a task fails due to an initial under-allocation, the maximum available memory of the machine is allocated for re-execution. We use their publicly available source code as the basis for implementation in our simulation environment.

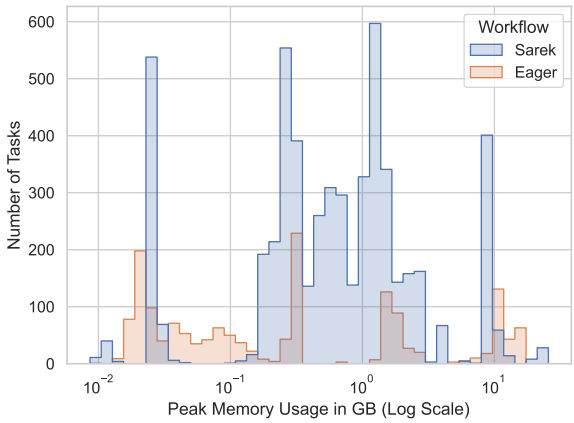


Fig. 5: Memory consumption of the experimental workflow tasks.

We provide an improved version of Tovar-PPM, *PPM-Improved*, which does not allocate the maximum available memory upon task failure, but instead doubles it, resulting in potentially less wastage for cluster setups with nodes equipped with lots of memory.

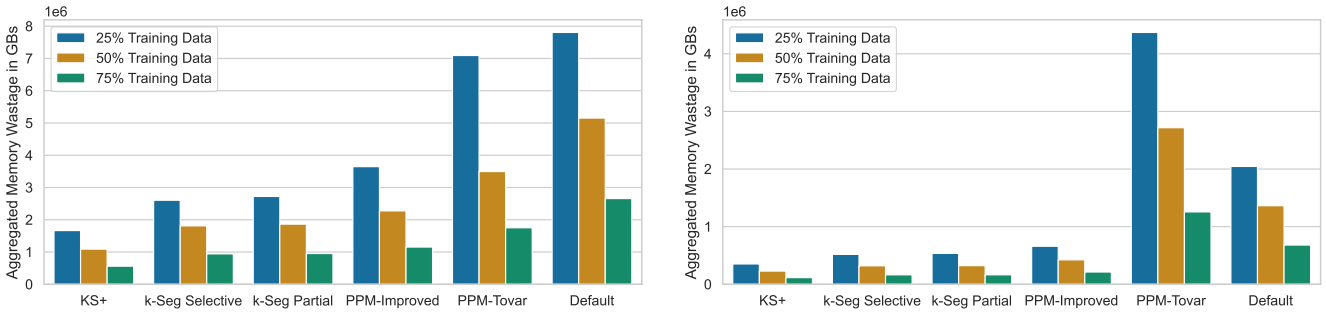
k-Segments Selective and *k-Segments Partial* [19] also serve as baselines and dynamically predict memory wastage over time using equally-sized segments and a failure strategy that either offsets everything after the failed segment or selectively offsets only the failed segment.

The default baseline uses the workflow developers' task memory limits.

C. Experimental Results

In this section, we first compare the aggregated memory wastage for all methods, followed by a task-based analysis. Subsequently, we analyze the impact of the number of segments on the prediction results of our KS+ method.

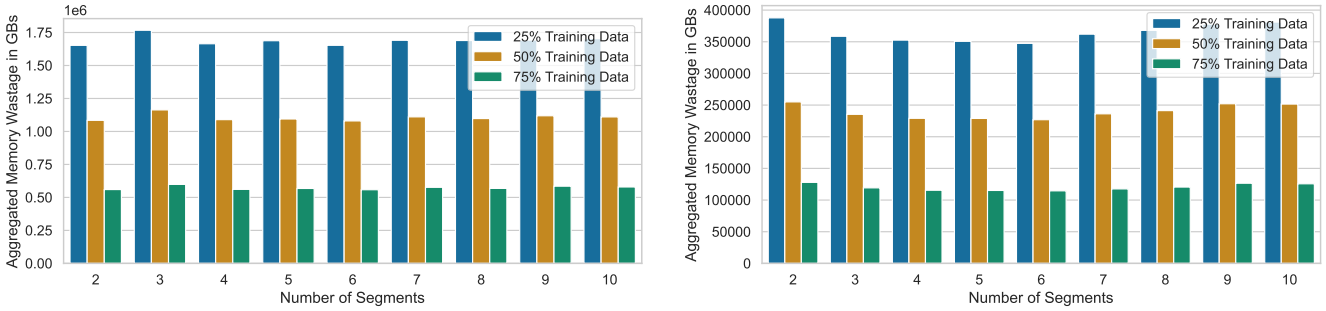
Figure 6 shows the aggregated memory wastage for each method. For the eager workflow, Figure 6a, KS+ achieves the lowest memory wastage among all methods. We can observe a reduction in memory wastage of 36%, 39%, and 40% for 25%, 50%, and 75% training data compared to the best-performing baseline and a reduction of 54%, 52%, and 51% compared to the best-performing peak memory prediction baseline, PPM-Improved. Also, for the sarek workflow, Figure 6b, KS+ significantly outperforms all baselines. Compared to the best-performing baseline, k-Segments Selective, it can be observed that KS+ reduces the wastage by 31%, 28%, and 29%. Compared to PPM-Improved, there is an average reduction of 45% for all training sets. Notably, PPM-Improved significantly outperforms PPM-Tovar, with the retry strategy adjustment being the only changed feature. We suspect that this is due to the large amount of memory on our experimental nodes, which results in high memory wastage for retried task executions.



(a) Eager workflow

(b) Sarek workflow

Fig. 6: Aggregated memory wastage in GBs for each method.



(a) Eager workflow

(b) Sarek workflow

Fig. 7: Effect of the number of segments on the memory wastage of our KS+ method.

This could also cause the default baseline to achieve lower memory wastage than PPM-Tovar for the Sarek workflow.

Figure 8 shows the memory wastage for each task in the eager workflow. It can be observed that the bwa task contributes the most to the total memory wastage. For this task, we observe a memory wastage reduction of 42%, 41%, and 37% compared to the best-performing baseline. For the mtnucratio task, we observe the largest relative reduction in memory wastage among all eager tasks. The AdapterRemoval task and the Samtools task are the only two tasks that show a slight increase in memory wastage compared to the k-Segments Selective method. Over both workflows, we achieve an unweighted average memory wastage reduction of 20%, 18%, and 21% compared to k-Segments Selective and a reduction of 37%, 38%, and 40% compared to PPM-Improved.

Finally, we analyze how the number of segments affects KS+'s memory wastage. Figure 7 shows the aggregated memory wastage as a function of the number of segments. For both workflows, there are no significant outliers and KS+ achieves low memory wastage. Furthermore, we cannot observe that fewer or more segments have a positive or negative impact on the total memory wastage. Both plots show minimal wastage when six segments are selected. However, not all tasks follow this trend, and different workflows and tasks may benefit from a different selection.

IV. Related Work

This section first describes research on workflow task memory prediction and then compares it to our KS+ method.

Tovar et al. [26] introduced a task memory prediction strategy for high-throughput scientific workflows. They propose a model that predicts the peak resource usage for a specific task using an analytical approach. The model can be adjusted to achieve two different objectives: maximizing throughput or minimizing resource wastage. Both objectives optimize resource usage based on a slow-peaks model, which assumes a scenario where tasks fail at the end of their execution.

Witt et al. [14] presented an online feedback loop-based resource allocation method to reduce resource wastage. Their predictor uses a task's aggregated input file sizes and employs various linear regression models to predict peak memory usage, adjusting the linear regression to overpredict and thus prevent task failures due to underprovisioning. They propose several offset strategies: the LR mean ± strategy considers the standard deviation as an offset, the LR mean - strategy considers only negative prediction errors, and the LR max strategy adds the largest observed underprediction as an offset.

Witt et al. [15] propose a second method to address the memory allocation problem by focusing on minimizing resource wastage rather than prediction error. Again, they assume a relationship between the size of the input data and a task's peak memory usage, training a linear model

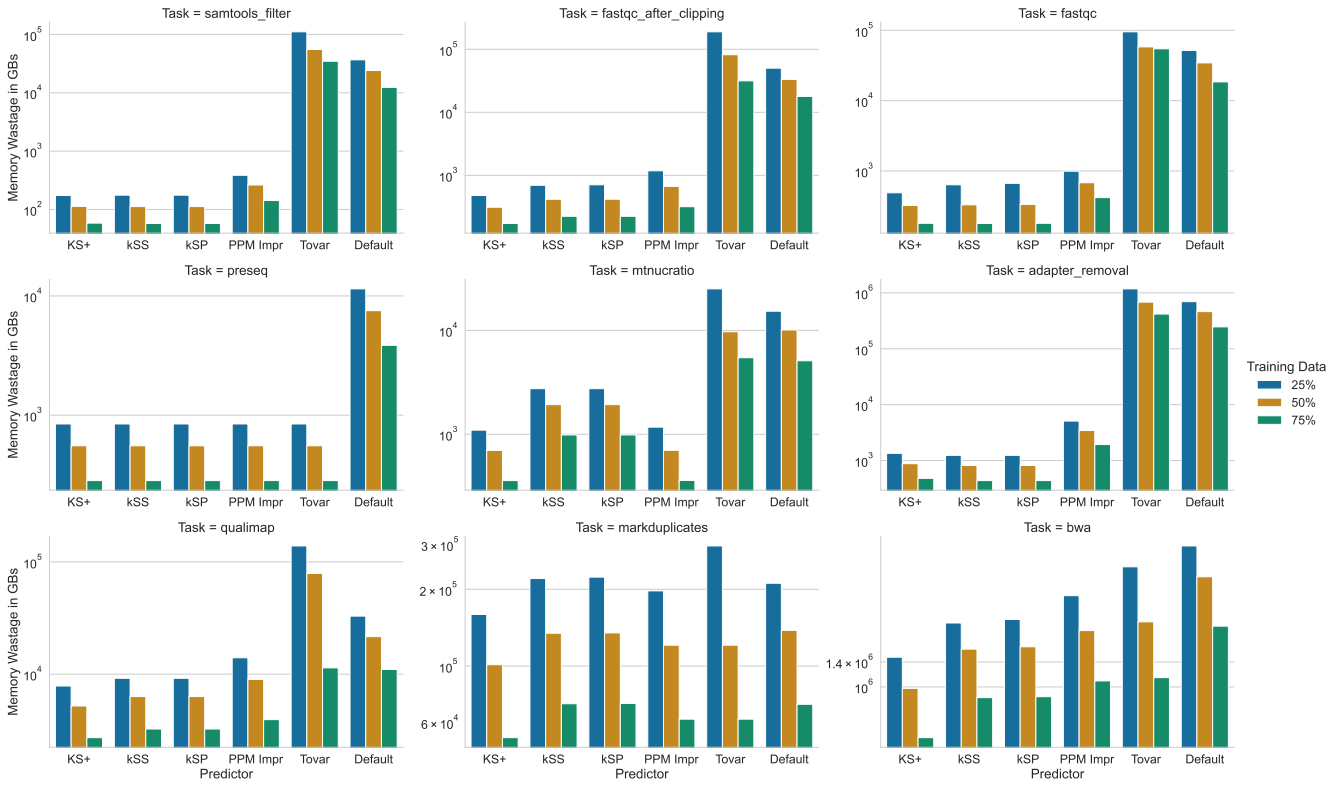


Fig. 8: Memory wastage in GBs for each of the nine tasks to be predicted in the eager workflow using different amounts of training data.

based on this assumption. They evaluate different strategies for handling task failures and show that the right failure-handling strategy has a significant impact on resource wastage because underpredictions are hard to avoid.

In our own previous work [16], we introduced two reinforcement learning approaches using gradient bandits and Q-learning to minimize resource wastage. These reinforcement learning bandits did not implement an offset technique as the reinforcement learning agents are inherently incentivized to avoid underpredictions. One drawback of our proposed methods is that they do not account for the dependency between task input size and resource usage.

We also presented a method to predict the memory consumption of workflow tasks over time [19]. We utilized time series monitoring data to predict the expected task runtime, which we then divided into equally sized segments. For each segment, we trained a linear regression model that predicts peak memory, resulting in a step function that models a task's memory consumption over time. Both the internal runtime prediction model and the segment-by-segment peak memory prediction models incorporate an offset strategy.

Compared to the static peak memory prediction methods from the literature, our KS+ method dynamically predicts memory over time, allowing for changing memory allocation during workflow execution and potentially reducing memory waste. Since such fine-grained memory prediction opens the space for underpredictions and subsequent task failures, we extend our previously presented k-Segments model [19] with a) dynamic segment size selection, b) an improved memory prediction model for the segments, and c) an improved failure handling that takes advantage of our dynamically sized segment lengths.

V. Conclusion

This paper presented KS+, a method for dynamically predicting workflow task memory consumption over time. To this end, KS+ predicts segments that define the timing and length of memory allocations. It then predicts the memory usage for each of these segments. Our method further employs a dynamic re-assignment strategy that not only adjusts the memory upon task failure but also re-assigns the segments.

Our experimental evaluation with four state-of-the-art baselines and two workflows from the popular nf-core repository has shown an average memory wastage reduction of 38% compared to the best state-of-the-art baseline. Furthermore, we have shown that our model is robust to different segment configurations.

In the future, we plan to dynamically determine the optimal number of segments for each task.

Acknowledgments

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

References

[1] E. Deelman, K. Vahi, G. Juve, M. Rynge, S. Callaghan, P. J. Maechling, R. Mayani, W. Chen, R. F. Da Silva, M. Livny *et al.*, "Pegasus, a workflow management system for science automation," *Future Generation Computer Systems*, vol. 46, pp. 17–35, 2015.

[2] R. F. Da Silva, R. Filgueira, I. Pietri, M. Jiang, R. Sakellariou, and E. Deelman, "A characterization of workflow management systems for extreme-scale applications," *Future Generation Computer Systems*, vol. 75, pp. 228–238, 2017.

[3] 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 *2022 IEEE International Conference on Big Data (Big Data)*, 2022, pp. 2709–2715.

[4] R. F. 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 *Proceedings of the 8th workshop on workflows in support of large-scale science*, 2013, pp. 58–67.

[5] S. Callaghan, P. J. Maechling, F. Silva, M.-H. Su, K. R. Milner, R. W. Graves, K. B. Olsen, Y. Cui, K. Vahi, A. Kottke *et al.*, "Using open-science workflow tools to produce scec cybershake physics-based probabilistic seismic hazard models," *Frontiers in High Performance Computing*, vol. 2, p. 1360720, 2024.

[6] 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, pp. 346–356.

[7] 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, pp. 44–60.

[8] D. Thain, T. Tannenbaum, and M. Livny, "Distributed computing in practice: the condor experience," *Concurrency and computation: practice and experience*, vol. 17, no. 2-4, pp. 323–356, 2005.

[9] B. Burns, B. Grant, D. Oppenheimer, E. Brewer, and J. Wilkes, "Borg, omega, and kubernetes," *Communications of the ACM*, vol. 59, no. 5, pp. 50–57, 2016.

[10] H. Albahar, S. Dongare, Y. Du, N. Zhao, A. K. Paul, and A. R. Butt, "Schedtune: A heterogeneity-aware gpu scheduler for deep learning," in *2022 22nd IEEE International Symposium on Cluster, Cloud and Internet Computing (CCGrid)*. IEEE, 2022, pp. 695–705.

[11] 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, pp. 94–106, 2019.

[12] W. Chen, A. Pi, S. Wang, and X. Zhou, "Os-augmented oversubscription of opportunistic memory with a user-assisted oom killer," in *Proceedings of the 20th International Middleware Conference*, 2019, pp. 28–40.

[13] 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.

[14] 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.

[15] 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.

[16] 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.

[17] E. R. Rodrigues, R. L. Cunha, M. A. Netto, and M. Spriggs, "Helping hpc users specify job memory requirements via machine learning," in *2016 Third International Workshop on HPC User Support Tools (HUST)*. IEEE, 2016, pp. 6–13.

[18] J. Bader, F. Skalski, F. Lehmann, D. Scheinert, J. Will, L. Thamsen, and O. Kao, "Sizey: Memory-efficient execution of scientific workflow tasks," in *2024 IEEE International Conference on Cluster Computing (CLUSTER)*, 2024.

[19] J. Bader, N. Diedrich, L. Thamsen, and O. Kao, "Predicting dynamic memory requirements for scientific workflow tasks," in *2023 IEEE International Conference on Big Data (BigData)*, 2023, pp. 182–189.

[20] R. F. 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] 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, pp. 171–185, 2024.

[22] 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.

[23] F. Hanssen, M. U. Garcia, L. Folkersen, A. S. Pedersen, F. Lescai, S. Jodoin, E. Miller, M. Seybold, O. Wacker, N. Smith *et al.*, "Scalable and efficient dna sequencing analysis on different compute infrastructures aiding variant discovery," *NAR Genomics and Bioinformatics*, vol. 6, no. 2, p. lqae031, 2024.

[24] 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.

[25] 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.

[26] 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.