Learning Low-Wastage Memory Allocations for Scientific Workflows at IceCube

Carl Witt
Humboldt-Universität zu Berlin
Berlin, Germany
wittcarl@informatik.hu-berlin.de

Jakob van Santen
Deutsches Elektronen-Synchrotron
Zeuthen, Germany
jakob.van.santen@desy.de

Ulf Leser
Humboldt-Universität zu Berlin
Berlin, Germany
leser@informatik.hu-berlin.de

***Abstract*—In scientific computing, scheduling tasks with heterogeneous resource requirements still requires user estimates, which tend to be inaccurate in spite of laborious manual processes used to derive them. In this paper, we show that machine learning outperforms user estimates and models trained at runtime can be used to improve the resource allocation for workflows. We focus on allocating main memory in batch systems, which enforce user estimates by terminating jobs.**

The key idea is to train prediction models that minimize the costs resulting from prediction errors rather than minimizing prediction errors. In addition, we detect and exploit opportunities to predict resource usage of individual tasks based on their input size.

We evaluated our approach on a 10 month production log from the IceCube South Pole Neutrino Observatory experiment. We compare our method to the performance of the current production system and a state-of-the-art method. We show that memory usage can be increased from 50% to 70%, while at the same time allowing users to provide only rough estimates of resource usage.

***Index Terms*—Machine Learning, Resource Management, Scheduling, Distributed Computing, Scientific Workflows**

I. Introduction

Scientific workflows [1] are a standard paradigm to orchestrate large-scale computational experiments or data analyses. Workflow management systems, e. g., Nextflow [2], Pegasus [3], Makeflow [4], or Saasfee [5] commonly implement large-scale parallelization by mapping workflow tasks to jobs in batch schedulers like HTCondor, Slurm, PBS, and LSF, or resource managers like Hadoop Yarn.

A commonality of distributed resource management systems is the necessity for manually providing resource usage estimates as a basis for job scheduling within the resource manager. These estimates, even though they are known to be notoriously inaccurate [6], [7] are typically interpreted as strict limits, meaning that resource managers kill jobs that violate their requested resources. In this case, a workflow management system typically makes another attempt to run the task with an increased resource allocation.

To run workflows on batch systems, users must derive estimates of peak resource usage (memory, storage, etc.) by carefully benchmarking new workflow configurations. The problem is to choose estimates that are large enough to avoid frequent failures due to insufficient resources, but not so large as to unduly restrict the achievable degree of parallelism. Striking this balance requires both time and expert knowledge of the workflow and the current state of the compute pool that only a few people have.

This problem is also faced by the IceCube Neutrino Observatory [8], a cosmic-ray research initiative. The project runs large amounts of simulations that are implemented as scientific workflows and executed via the IceProd [9] workflow management system. IceCube has dedicated and opportunistic compute resources at its disposal, which are located at different compute sites and are managed through HTCondor.

The problem of allocating the right amount of resources is non-trivial even when historical data is available. For instance, always requesting the maximum resource usage seen so far is not a good choice when job resource usage varies strongly. Although requesting less than the maximum resource usage necessarily introduces failures due to insufficient resources, it might reduce excess allocations for a large number of smaller jobs. The optimal choice depends on the distribution of resource usage, run times, and the costs of failures.

In this paper, we show how to train machine learning models for peak memory usage prediction that account for these factors, which lowers the burden on users to accurately estimate the memory usage of different steps and configurations before running a workflow. Instead, users can provide rough estimates which are used only during a short period of time in which training data on the actual resource usage of tasks is collected. Subsequently, user estimates are replaced by more accurate predictions, improving resource usage efficiency.

In addition, automating resource usage prediction allows to make predictions at the individual task level, whereas in IceProd and other workflow management systems, users provide peak resource usage estimates only for groups of similar tasks, which further limits the accuracy of estimates. We evaluate our approach on real-world data from the IceCube project and compare it to a state-of-the-art resource allocation strategy.

The paper is structured as follows: Section II formalizes the problem and summarizes prior work. Section IV introduces our machine learning approach. Section V provides background information on scientific workflows at IceCube and validates the potential for our approach based on the production logs. Section VI presents the results. We conclude with a discussion and future work in Section VII.

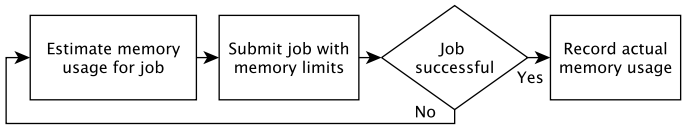

Fig. 1: The process of executing jobs on batch resources. The memory estimation module is key to the resource-efficient execution of jobs.

II. The Memory Allocation Problem

To execute a workflow using a batch scheduler, a memory usage estimate has to be provided for every attempt to execute a job. If the estimate is larger than the actual peak usage of the job, the attempt succeeds; otherwise, it fails and the job needs to be submitted again, requesting increased resources. The process is shown in Figure 1. The goal is to allocate memory such that resource wastage as defined below is minimized.

In this work, we focus on peak memory usage, although a job's memory usage usually varies over time. The reason is that in practice batch scheduler allocations are rigid, i. e., do not change over time. This means that job failures depend only on whether peak memory usage exceeds the allocation.

A. Problem Definition

Let $k_i \geq 1$ denote the number of attempts needed to execute the i-th job. We denote the memory allocated to the j-th attempt of the i-th job as a_{ij}.

The resource usage of the i-th job corresponds to the product of its peak memory usage r_i and its run time τ_i. The resource usage of the attempt equals the job resource usage if it succeeds, and 0 if the attempt fails. This notation is similar to that in [10].

$$\text{usage}(r_i, \tau_i, a_{ij}) = \begin{cases} r_i \tau_i & \text{if } a_{ij} \geq r_i \\ 0 & \text{otherwise} \end{cases} \tag{1}$$

$$U = \sum_i \sum_{j=1}^{k_i} \text{usage}(r_i, \tau_i, a_{ij}) \tag{2}$$

Resource wastage corresponds to the product of excess allocation and the attempt's run time. On successful attempts, we consider the difference between allocated memory a_{ij} and actual peak usage r_i as excess allocation. On failed attempts, we consider all allocated memory a_{ij} as excess allocation. We refer to the wastage resulting from the former and latter as over- and under-sizing wastage, respectively. In case of insufficient memory, the run time of a job usually differs from its run time with sufficient resources, and is denoted as τ_i^*.

$$\text{wastage}(r_i, \tau_i, a_{ij}) = \begin{cases} (r_i - a_{ij})\tau_i & \text{if } a_{ij} \geq r_i \\ a_{ij}\tau_i^* & \text{otherwise} \end{cases} \tag{3}$$

$$W = \sum_i \sum_{j=1}^{k_i} \text{wastage}(r_i, \tau_i, a_{ij}) \tag{4}$$

This definition of wastage reflects the asymmetric costs between allocating too much and too little memory. Usually, a slight over-prediction is not a problem, but a slight under-prediction potentially wastes a lot of resources. However, the amount of under-sizing wastage heavily depends on the time to failure τ_i^*.

We refer to the ratio between used and allocated resources as memory allocation quality (MAQ). The metric relates resource usage to both over- and under-sizing wastage. Low values correspond to high relative wastage and 100% corresponds to perfect memory allocation with no wastage at all. The memory allocation problem consists in minimizing resource wastage, or equivalently, maximizing memory allocation quality.

$$\text{MAQ} = \frac{U}{U + W} \tag{5}$$

MAQ elegantly separates the quality of resource allocation decisions from the workflow scheduling problem, since different length schedules can have the same MAQ, depending on how the attempt sequences are packed by the scheduler.

III. Related Work

Tovar et al. [10] have proposed a strategy to solve the memory allocation problem. The approach takes an empirical distribution of peak memory usage as input and computes the amount of resources a_1 to assign to the first attempt of each job. The initial allocation is chosen such that the sum of over- and under-sizing wastage is minimized, under the assumption that a job is restarted using a maximum allocation a_m, should the first allocation be insufficient. To simplify the computation, independence of run times and peak memory consumption is assumed. As a result, Tovar's a_1 minimizes the amount of excess allocation as defined in Section II-A, rather than the wastage (which corresponds to excess allocation weighted by attempt run time).

Witt et al. [11] have reviewed machine learning based prediction methods for memory usage, run time, and queue times in the context of batch scheduled workloads. Very few approaches focus on predicting memory usage, and Tovar et al. propose the first method with an asymmetric loss function.

Gaussier et al. [12] have proposed a machine learning approach with asymmetric loss function for predicting the run times of batch jobs. This is useful for batch schedulers with backfilling, i. e., that allow short jobs to skip the queue to fill currently idle resources that are insufficient for the job at the head of the queue [13]. Similar to the memory allocation problem, over- and under-prediction have asymmetric effects on backfilling, because jobs with under-predicted run times may turn out not to fit in a gap between other jobs.

Zhang et al. [14] have proposed a clustering-based approach that determines groups of jobs with similar resource usage within a workflow. A disadvantage of this approach is that there is no means to predict the cluster of a job at runtime other than from a previous execution. In contrast, we use job parameters such as input size to predict peak memory usage prior to the first execution of a job.

IV. Low-Wastage Memory Allocation

Our approach to the memory allocation problem is based on two ideas. First, we use input sizes, i. e., the amount of data processed by a job to predict its peak memory usage. Second, we train the prediction model in a way that minimizes resource wastage, rather than prediction error. This accounts for the asymmetric costs of over- and under-sizing.

In the following, we put the memory allocation quality from Section II in concrete terms for different failure handling strategies. We then show how to optimize the parameters of a linear model so as to maximize memory allocation quality and how the resulting prediction models differ from standard regression models.

A. Wastage for Exponential Strategies

Since memory usage is usually not reliably predictable, a strategy to handle under-prediction is needed independently of how first allocations are chosen. An intuitive approach is to double the allocated memory upon each failed attempt. We refer to this strategy as exponential strategy with basis $b = 2$.

$$a_{ij} = b \cdot a_{i(j-1)} \tag{6}$$

For an exponential allocation strategy with base b, the number of failed attempts for the i-th job is

$$k_i = \max\left(0, \left\lceil \log_b \frac{r_i}{a_{i1}} \right\rceil\right) \tag{7}$$

The under-sizing wastage U equals the resources allocated to the k_i failed attempts and the oversizing wastage O equals the excess allocation in the successful final attempt.

$$U = \sum_i \sum_{j=1}^{k_i} a_{i1} b^{j-1} \tau_i^* = a_{i1} \frac{b^{k_i} - 1}{b - 1} \tau_i^* \tag{8}$$

$$O = \sum_i (a_{i1} b^{k_i} - r_i) \tau_i \tag{9}$$

B. Wastage for Maximum-Strategies

Tovar et al. [10] use a three-step allocation strategy where the second attempt of a job is allocated the largest seen memory usage a_m in the training data. If a new job fails with the historical maximum allocation size a_m, Tovar et al. recommend using the amount of resources offered by the largest available compute node a_v for the third attempt. The assumption is that if the third attempt fails, the workflow is not executable.

$$a_{i2} = a_m \tag{10}$$

$$a_{i3} = a_v \tag{11}$$

Determining a_v can be problematic in a pool of opportunistic compute nodes, however, a three-step strategy has the potential advantage of inducing lesser failures when initial allocations are far from the true peak memory usage. The under- and oversizing wastages follow from Equation 3 and have no special closed form as for exponential strategies.

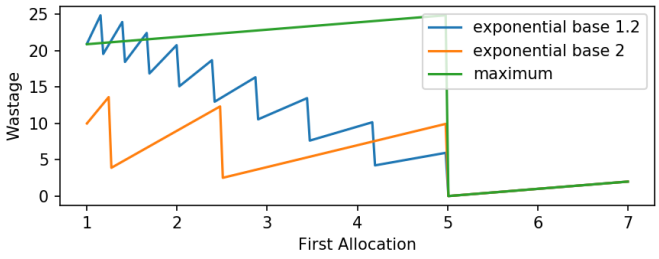

Fig. 2: Exemplary wastage for different failure handling strategies as a function of first allocation. This example is constructed for a single task with a peak memory usage of 5 units. The maximum strategy uses $a_m = 4.9$ and $a_v = 20$.

Figure 2 shows an example of the combined wastage (over-sizing + under-sizing) resulting from different first allocations under three failure-handling strategies. A smaller basis for exponential strategies leads to more failures for a strong under-prediction, but produces less over-sizing wastage for a small under-prediction. For the maximum strategy, the costs of under-prediction heavily depend on how tight the second allocation a_m and final allocation a_v is chosen.

C. Low-Wastage Regression

The wastage criteria from Sections IV-A and IV-B both depend on the amount a_{i1} of memory allocated to the first attempt of a job. We derive this amount from a job's input size s_i, replacing negative values with a minimum allocation size a_l.

$$a_{i1} = \max(\theta_1 s_i + \theta_0, a_l) \tag{12}$$

Substituting this into the wastage expressions (e.g., Equations 8 and 9) for a failure handling strategy gives a loss function that states the resource wastage when choosing first allocations as a linear function of input size with slope θ_1 and intercept θ_0. Note that Tovar's allocation method is a special case of this approach with $\theta_1 = 0$.

The slope and intercept parameters can be optimized using a training set of (input size, peak memory consumption) tuples by using standard approaches such as grid search, evolutionary algorithms, or gradient descent. However, due to the zigzag shape of the loss function for a single task (Figure 2), the loss function for a set of tasks also has many local optima, which makes it difficult for gradient-based methods to find a good solution.

The globally optimal parameters always describe a line that passes through two of the training data points, because a line in between points just increases the under-sizing wastage for the points above and the over-sizing wastage for the points below the line. However, testing all $\mathcal{O}(n^2)$ lines is rather costly, since evaluating each line takes $\mathcal{O}(n)$ time to compute the wastage. Where historical data comprises thousands of tasks, this is inconveniently slow.

We found that testing different quantile regression lines provides an accurate and fast alternative. In addition, the

optimal solution to the optimization problem usually does not generalize well, because it passes exactly through two data points, often leading to impractically tight predictions. We thus test quantile regression lines with quantiles ranging from the median up to the 99.99% percentile. One could also include smaller quantiles, but having more than half of the jobs fail on their first attempt is not advisable in practice. We do not select evenly spaced quantiles but use a quadratic approach that generates more candidate quantiles towards the end of the range. Across all candidate quantiles, we select the one that yields the least wastage when using it for predicting memory usage. In the experiments, we tested $k = 25$ quantile candidates. The approach is summarized in Algorithm 1.

Result: Slope θ_1 and intercept θ_0 for computing the first allocation according to Equation 12.

$w_{\text{min}} \leftarrow \infty$;
$\theta^* \leftarrow (\infty, \infty)$;
$l \leftarrow$ interpolate k values linearly between 0.01 and 0.7;
for *quantile* $q \in \{1 - a^2 \mid a \in l\}$ **do**
  $\theta_1, \theta_0 \leftarrow$ fit a quantile regression line for q;
  $a_{i1} = \theta_1 s_i + \theta_0$;
  if *MAQ*$(a_{i1}) < w_{min}$ **then**
    $\theta^* \leftarrow (\theta_1, \theta_0)$;
  end
end

Algorithm 1: Low-Wastage Regression: Selecting model parameters for peak memory prediction by testing different quantile regression lines on the amount of resulting resource wastage.

V. Case Study: IceCube Workflows

We evaluated our method on production logs from scientific workflows at the IceCube project. This section provides background information on the IceCube project, its scientific computing workloads, the IceProd workflow management system, and an analysis of the memory usage efficiency during a ten month period in the production system.

A. IceCube Neutrino Observatory

The IceCube Neutrino Observatory is an array of 5160 Digital Optical Modules (DOMs) buried 1.5–2.5 km below the surface of the glacial ice at the geographic South Pole. Each DOM records the arrivals of individual Cherenkov photons as they pass through the 1 km^3 instrumented volume. Interpretation of the experimental data requires a model of the detector's response to the stimuli under consideration. Because the particles IceCube observes can have energies far beyond those that can be produced artificially, it is impossible to measure the model directly. Instead, it must be computed by Monte Carlo simulation of individual particles. These simulations must be produced in large quantity to generate a detector response model that is sufficiently precise to analyze the experimental data.

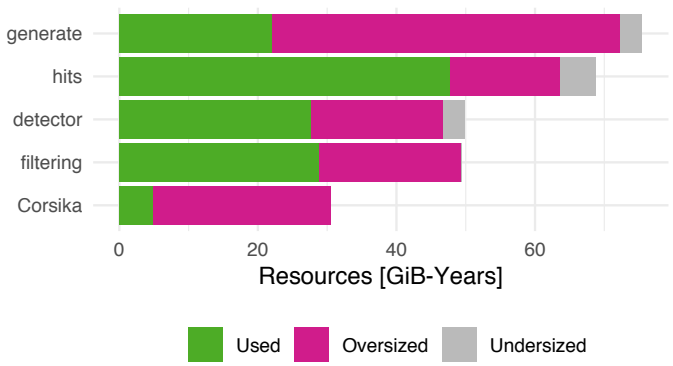

Fig. 3: Total resource allocation per task type. Green, pink, and gray areas indicate the fraction of used and unused resources due to oversized allocation and undersized allocation, respectively.

B. Simulation Production Workflow for IceCube

IceCube's simulation production workload is divided into *datasets*; each dataset consists of a set of *abstract tasks* and a number of *jobs*. The abstract tasks can be arranged in an arbitrary DAG, but tend to be linear in practice. Each job is a concrete instance of an abstract task graph with concrete parameters, e.g. the pseudo-random number sequence used to drive the simulation of particles. Jobs are independent of each other, compute- and memory- rather than I/O-bound, and tend to not be especially time-critical.

Figure 3 shows the most resource consuming abstract tasks in the workload, as captured by production logs on resource usage (see Section V-D). Gray indicates memory allocated to failed tasks, pink indicates excessively allocated memory in successful tasks, and green indicates the amount of memory that was actually used. The combined height equals the total allocated memory.

In the *generate* task, high-energy particles are propagated through the detector medium. Particle interactions trigger cascades in which single high-energy particles produce many lower-energy particles. The stochastic nature of the process also causes variance in peak memory usage. This task is typically followed by *hits*, in which the induced Cherenkov photons are tracked from their sources to the DOMs. *Detector* simulates the response of the detection electronics and produces the same kind of data as the real detector. Finally, *filter* applies the same event selection and reconstruction as for experimental data. *Corsika* is a variant of *generate* that uses the CORSIKA air-shower simulation program [15], which uses static memory allocation.

C. IceProd Workflow Management System

Workflows are executed by IceProd [9], a custom middleware running on top of HTCondor [16], [17]. An IceProd *pilot* runs in an HTCondor slot and claims tasks from a central manager that fit within the slot's allocation of CPUs, GPUs, main memory, storage space, and wall-clock time. Because

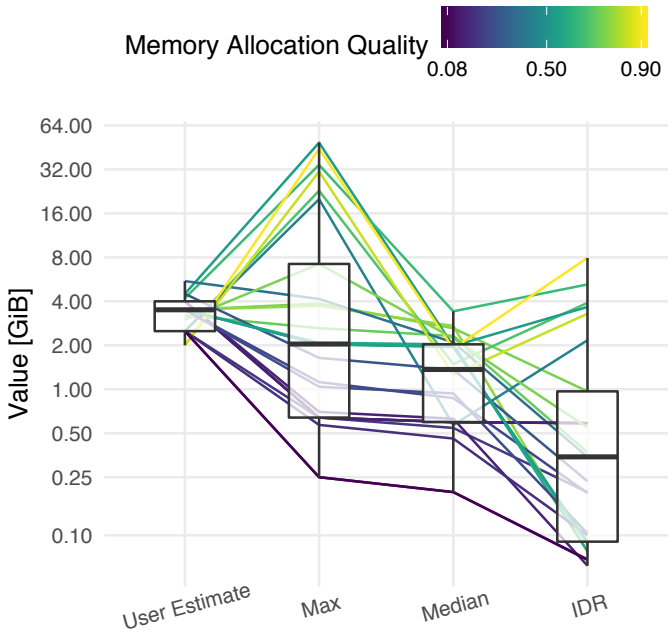

Fig. 4: Memory usage as estimated by IceProd users compared to actual peak, median, and interdecile memory usage of the 25 abstract tasks with the highest accumulated run time.

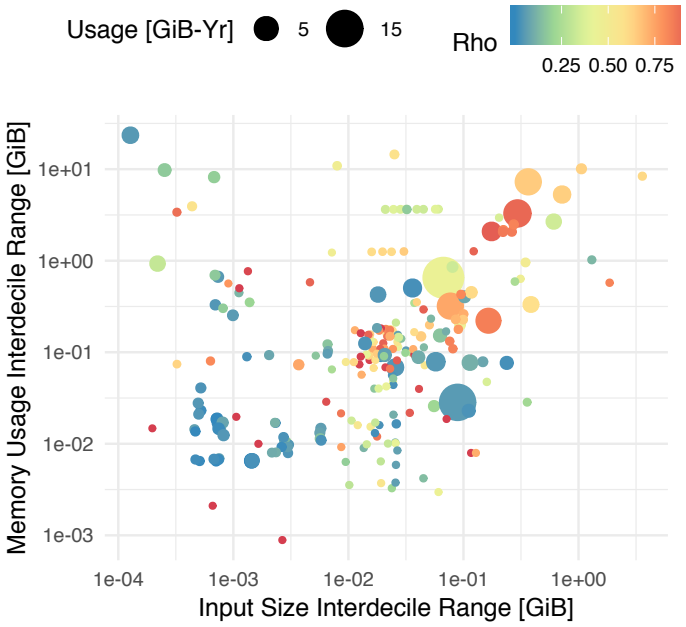

Fig. 5: Variability of input size and memory usage per abstract task. Regression based memory allocation has the highest potential where input sizes and memory usage vary strongly (top-right corner) and are highly correlated (red).

the slots are distributed over both dedicated and opportunistic resources at multiple sites, the allocations can vary greatly, from e.g. a single CPU core with 4 GB of main memory to 24-core, whole-machine slots with 12 GB per core.

Each task's initial resource requirements are based on a user estimate for the abstract task. IceProd monitors the execution of each task, and aborts a task if its resource usage exceeds its request. The task is then restarted with twice the requested resource whose constraint was violated. For main memory constraints, the time to failure tends to be uniformly distributed fraction of the time it would take the task to run to completion.

D. Insights from Production Logs

In this section, we analyze the resource usage and allocation during a 10 months period in the production system. We first focus on the quality of the user estimates and then analyze the predictive potential of input sizes and predecessor task metrics.

The log covers 727,220 tasks with a total memory usage of 1.06 PiB and a total CPU usage of 76 Core-Years[1]. The total area, i. e., product of CPU time and memory usage, amounts to 133 GiB-Core-Years. The median task resource usage is 32 Core-Minutes and 1.4 GiB.

1) User Estimate Accuracy: Figure 4 shows the relationship between user estimates and actual memory usage for the 25 most time-consuming abstract tasks. These abstract tasks account for 73% of the total GiB-Year usage.

[1]997,947 tasks with a total CPU usage of 136 Core-Years were excluded due to missing memory usage information

It is striking that all user estimates are located between 2 and 6 GiB, whereas true peak memory usage ranges from 0.25 to 49 GiB. In addition, the variability of memory usage varies strongly. The interdecile range, i. e., the difference between the 90% percentile and 10% percentile can be as small as 0.06 GiB and as large as 7.93 GiB.

The memory allocation quality also varies strongly across abstract tasks. Among the top 25 abstract tasks the lowest MAQ is 8% and the highest MAQ is 91%. Interestingly, the tasks with the best MAQ are not necessarily the ones with least variability. The rank correlation (Kendall) between MAQ and interdecile range is only 0.29.

The total oversizing wastage amounts to 136 GiB-Years, whereas only 12 GiB-Years are allocated to tasks that failed due to insufficient memory. This shows that user estimates are conservative, and confirms the known tendency of users to overestimate resource usage [6]. Dividing the total 133 used GiB-Years by the 133+136+12 allocated GiB-Years, we find that the overall MAQ is 47%.

2) Predictive Potential of Input Size: A feature of scientific workflows is the availability of intermediate result set sizes for tasks, as given by the sum of a job's input files. In the workflows considered here, tasks consume at most one input file and produce one output file. In this section, we analyze the potential of input size to predict peak memory usage. Compared to the baseline method of Tovar et al., which computes a single allocation per abstract task, using input sizes allows to predict memory usage per individual task. This makes sense for abstract tasks with heterogeneous memory

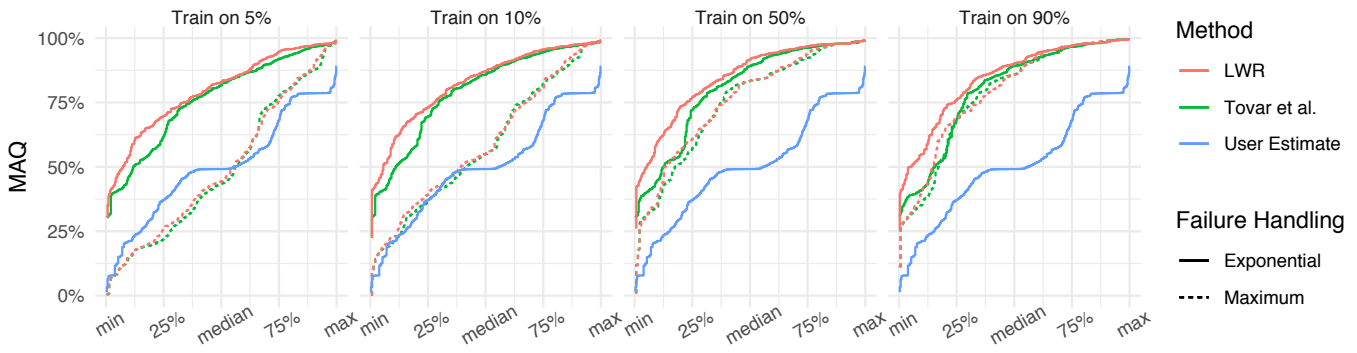

Fig. 6: Comparison of the memory allocation quality (MAQ) cumulative distributions (higher is better) achieved by different combinations of allocation method and failure handling strategy. MAQ is computed per abstract task, where the first k% (with respect to a job's finish time in the logs) of the data are used for training and the rest for and evaluation. The green line shows the performance of the estimates per abstract task as provided by IceProd users.

consumption, for which a single allocation size does not fit well. If memory usage is related to some other variable, such as input size, predictions can be improved.

Figure 5 shows the heterogeneity in memory usage (y-axis) and input size (x-axis) as measured by interdecile range. For a regression-based approach, the abstract tasks in the top-right corner are most interesting, since there is variance in memory usage that can potentially be explained by input size. To measure the correlation between input size and memory consumption, we computed the Pearson correlation coefficient between both variables for each abstract task, as displayed by the color of the points. There is potential for a regression based-approach, since several resource-intensive abstract tasks expose relatively high correlation (> 0.7).

We also checked the predictive potential of other features, such as the peak memory usage of a task's predecessor task. Although in some cases, a high correlation (> 0.8) exists, only a small fraction of resources are allocated to these tasks.

VI. Evaluation

We evaluate our method using the resource usage logs described in Section V-D. The focus of our analysis is the memory allocation quality achieved by different combinations of allocation methods and failure handling strategies. For training the allocation methods, we use the first k% of the data per abstract task, ordering jobs by the time they finished in the production system. We considered only the 142 abstract tasks that comprise at least 100 jobs, to make sure the training split contains at least five jobs. This retains 99.7% of the overall log file. We refer to user estimates as they have been provided by IceProd users as the baseline method. We refer to Tovar's state-of-the-art method [10] as *Tovar et al.*, where not otherwise stated, this assumes the maximum failure handling strategy. We refer to Tovar's method with an exponential failure handling strategy as *Modified Tovar et al.*; our method is referred to as Low-Wastage Regression (*LWR*).

First, we show the memory allocation quality that is achieved on an abstract task level. Abstract tasks are a natural unit of analysis since we train one prediction model per abstract task. However, since abstract tasks differ in the amount of resources allocated to them, we subsequently conduct an aggregate analysis that weights the memory allocation qualities by an abstract task's share of the overall resource allocation.

In our experiments, we assume a time to failure $\tau^* = 0.5$. In the log files, job time to failure follows an approximately uniform distribution with support $[0, 1]$.

The parameters of Tovar's maximum-strategy are chosen such that a_m equals the largest observed memory usage within the training set and a_v equals the largest observed memory usage across all jobs in the log. As described earlier, the IceProd workflow management system uses variable size *job pilots* which can be even larger than a_v. However, further increasing a_v only increases the over-sizing wastage generated by this strategy. We thus select a value that is as large as necessary and as small as possible, although in practice a_v needs to be choosen according to the resource pool. As minimum allocation size, we set a_l to 100 Megabytes.

A. Comparing MAQ Distributions

Figure 6 shows the cumulative distribution of memory allocation quality across abstract tasks. User estimates score a median MAQ of 49%. The state-of-the-art method as discussed in [10] uses the maximum failure handling strategy and achieves only a median MAQ of 43% but improves to 55% and 83% when trained on 10% and 50% of the data, respectively. Our method achieves 83% median MAQ using only 5% of the data for training. However, it turns out that the state-of-the-art method improves to 81.7% median MAQ when applying the exponential failure handling strategy rather than the maximum-strategy proposed by Tovar et al. Overall, the maximum-strategy is not recommendable, as it becomes competitive only when using at least 90% of the log data for training.

Our advantage in MAQ stems mainly from avoiding failures, as shown in Figure 7. Comparing the cumulative distributions for over- and under-sizing wastage, we have a larger advantage

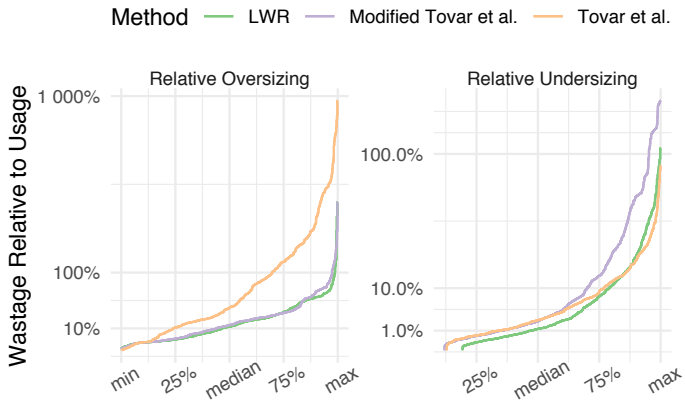

Fig. 7: Cumulative distribution of over-sizing and under-sizing wastage (lower is better), relative to the amount of used resources. This shows that the improvements of our method stem mainly from reducing under-sizing wastage.

on under-sizing wastage than on over-sizing wastage, although scoring slightly better in both aspects. The potential to reduce under-sizing wastage is most apparent in scenarios where input size correlates to peak memory usage, as shown in Figure 8. Here, a large constant allocation size produces large amounts of over-sizing wastage. The only way to reduce over-sizing wastage is to introduce job failures by allocating less than the largest observed memory usage. Here, our linear model can save significant amounts of resources by assigning less memory to jobs with smaller inputs.

Regarding robustness, i. e., the worst observed memory allocation qualities, the state-of-the-art method produces the lowest overall MAQ of only 0.3%. The worst observed MAQ when applying user estimates is 3.3%. In contrast, our lowest produced MAQ is 26%.

B. Effective Memory Allocation Quality

In this section, we evaluate the effective memory allocation quality obtained when relying on user estimates during the collection of training data and subsequently replacing user estimates with the predictions from the trained model. We split the data again according to job finish times, using the first k% for training and the rest for evaluation. This time however, we charge a cost for the training period by computing the resource wastage resulting from allocating memory according to user estimates during the training period.

Since the goal is to relieve users from the burden of having to determine the memory requirements of abstract tasks, we do not directly use the estimates provided by IceProd users. These estimates include user knowledge about the resource usage of an abstract task. To simulate a scenario without detailed user estimates, we compute coarse grained user estimates by taking the median user estimate per task type (e. g., generate, hits, detector, etc.). This reduces the 321 user estimates (for each abstract task) to one user estimate for each of 16 task types.

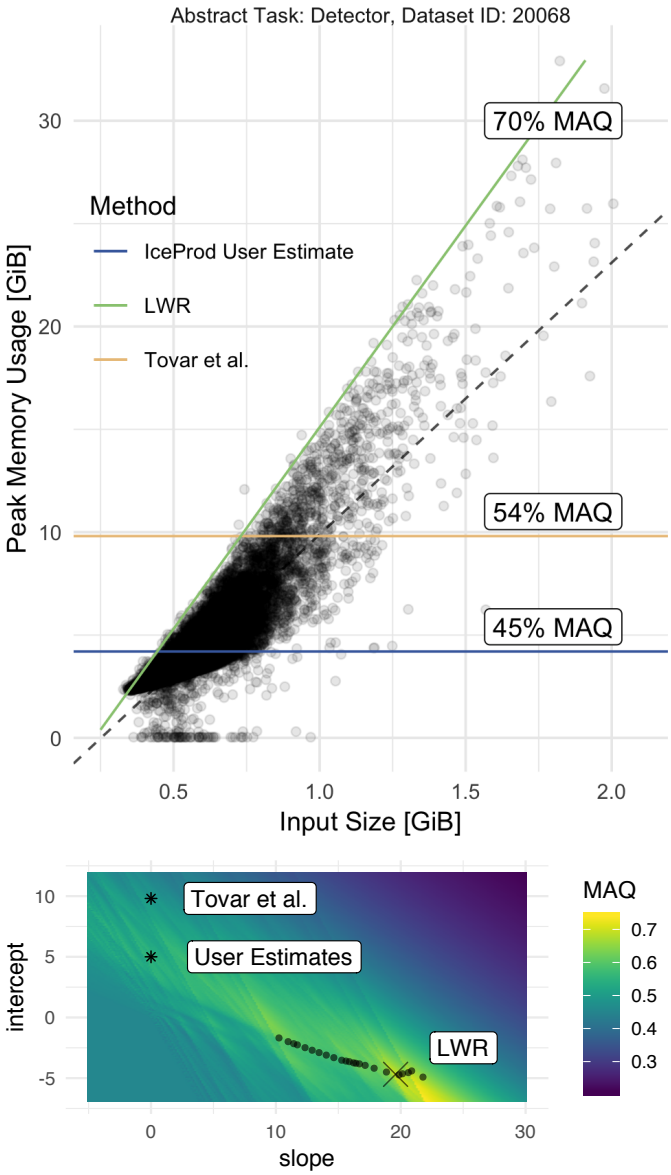

Fig. 8: Top: Relationship between input sizes and peak memory consumption on an exemplary abstract task. Tovar's methods assign each job a fixed amount of memory (orange line). We assign memory proportional to input size (purple line). Compared to a linear regression (dashed line) or quantile regression, our allocations minimize the resource wastage resulting from prediction errors, rather than prediction errors. Bottom: The exponential wastage resulting from different slope and intercept values for first allocation. The dots indicate the tested positions corresponding to different quantiles for quantile regression. The cross marks the best quantile regression line, corresponding to the 96.7 percentile. The global optimum on the surface (which does not necessarily generalize as well) scores a MAQ of 73.3%.

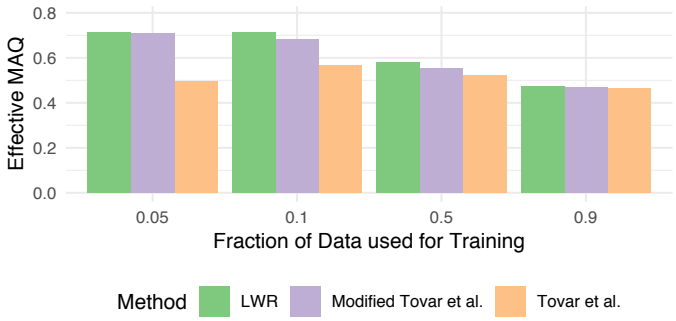

Fig. 9: Effective MAQ when using coarse grained user estimates during training and trained models afterwards. Modified Tovar refers to the state-of-the-art approach with our exponential failure handling strategy, Tovar refers to the unchanged state-of-the-art, and Witt refers to our method.

Figure 9 shows the MAQ that can be achieved when training models during workflow execution. When using 90% of the data for training, the overall achievable MAQ is roughly the same as relying completely on user estimates, since the learned models are applied only to 10% of the jobs. By waiting for 5% of the jobs of each abstract task to finish before replacing user estimates with predictions from the trained models, overall MAQ can be improved to 70%. The relatively small advantage of our method over the modified state-of-the-art method (replacing the proposed maximum-strategy with our exponential failure handling strategy) is explained by our log analysis in Section V-D. Although some tasks, such as the one in Figure 8, strongly benefit from per-job allocations, the overall share of resources allocated to such predictable tasks is not as large as expected for the IceCube workflows.

VII. Conclusion

We have proposed a machine-learning based resource allocation strategy that minimizes resource wastage. Evaluation on real-world data has shown that failure handling strategies have a strong impact on overall system performance and that doubling allocated resources after each failed attempt performs much better than the falling back to maximum observed values so far, as proposed by the state-of-the-art method.

By replacing user estimates after measuring input sizes and memory usage for each abstract task for a small amount of time, overall memory allocation quality can be improved from 50% to 70%.

As future work, similar prediction models could be used to predict the usage of other resource types, such as wall-clock time or disk storage. Peak memory prediction could also be extended to a multivariate model by taking into account additional job parameters. Additional job information at the IceCube project includes software versions, different implementations, and generator settings that control, e. g., particle energies.

In conclusion, we have shown how to learn memory allocations for jobs that significantly reduce resource wastage compared to user estimates and a state-of-the-art method.

Acknowledgments

Carl Witt received funding by Deutsche Forschungsgemeinschaft through the SOAMED graduate school (GRK 1651). We thank David Schultz and the IceCube Collaboration for providing access to IceProd monitoring data.

References

[1] C. S. Liew, M. P. Atkinson, M. Galea, T. F. Ang, P. Martin, and J. I. V. Hemert, "Scientific Workflows: Moving Across Paradigms," *ACM Computing Surveys (CSUR)*, vol. 49, no. 4, pp. 66–39, Feb. 2017.

[2] 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, pp. 316–319, Apr. 2017.

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

[4] M. Albrecht, P. Donnelly, P. Bui, and D. Thain, "Makeflow: A Portable Abstraction for Data Intensive Computing on Clusters, Clouds, and Grids," in *the 1st ACM SIGMOD Workshop*. New York, New York, USA: ACM Press, 2012, pp. 1–13.

[5] M. Bux, J. Brandt, C. Lipka, K. Hakimzadeh, J. Dowling, and U. Leser, "SAASFEE: Scalable Scientific Workflow Execution Engine." *PVLDB*, vol. 8, no. 12, pp. 1892–1903, 2015.

[6] C. Delimitrou and C. Kozyrakis, "Quasar: resource-efficient and QoS-aware cluster management." *ASPLOS*, pp. 127–144, 2014.

[7] C. Reiss, A. Tumanov, G. R. Ganger, R. H. Katz, and M. A. Kozuch, "Heterogeneity and dynamicity of clouds at scale," in *ACM Symposium on Cloud Computing*. New York, New York, USA: ACM Press, 2012, pp. 1–13.

[8] M. G. Aartsen *et al.*, "The IceCube Neutrino Observatory: Instrumentation and Online Systems," *JINST*, vol. 12, no. 03, p. P03012, 2017.

[9] D. Schultz, "IceProd 2: A Next Generation Data Analysis Framework for the IceCube Neutrino Observatory," *J. Phys. Conf. Ser.*, vol. 664, no. 6, p. 062056, 2015.

[10] B. Tovar, R. Ferreira 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, pp. 240–253, 2018.

[11] 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*, Jan. 2019.

[12] É. Gaussier, D. Glesser, V. Reis, and D. Trystram, "Improving backfilling by using machine learning to predict running times." *SC*, pp. 64–10, 2015.

[13] D. Tsafrir, Y. Etsion, and D. G. Feitelson, "Backfilling Using System-Generated Predictions Rather than User Runtime Estimates." *IEEE Transactions on Parallel and Distributed Systems*, vol. 18, no. 6, pp. 789–803, 2007.

[14] Q. Zhang, N. Kremer-Herman, B. Tovar, and D. Thain, "Reduction of Workflow Resource Consumption Using a Density-based Clustering Model," in *2018 IEEE/ACM Workflows in Support of Large-Scale Science (WORKS)*. IEEE, 2018, pp. 1–9.

[15] D. Heck, J. Knapp, J. N. Capdevielle, G. Schatz, and T. Thouw, *CORSIKA: a Monte Carlo code to simulate extensive air showers.*, Feb. 1998.

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

[17] D. Schultz, B. Riedel, and G. Merino, "Pyglidein – A Simple HTCondor Glidein Service," *J. Phys. Conf. Ser.*, vol. 898, no. 9, p. 092018, 2017.