Feedback-Based Resource Allocation for Batch Scheduling of Scientific Workflows

Conference Paper · August 2019

CITATIONS
0

READS
61

3 authors, including:

Carl Witt
Humboldt-Universität zu Berlin
9 PUBLICATIONS **44** CITATIONS
SEE PROFILE

Ulf Leser
Humboldt-Universität zu Berlin
349 PUBLICATIONS **5,367** CITATIONS
SEE PROFILE

Feedback-Based Resource Allocation for Batch Scheduling of Scientific Workflows

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

Dennis Wagner
Humboldt-Universität zu Berlin
Berlin, Germany
wagnerde@informatik.hu-berlin.de

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

***Abstract*—A scientific workflow is a set of interdependent compute tasks orchestrating large scale data analyses or in-silico experiments. Workflows often comprise thousands of tasks with heterogeneous resource requirements that need to be executed on distributed resources. Many workflow engines solve parallelization by submitting tasks to a batch scheduling system, which requires resource usage estimates that have to be provided by users. We investigate the possibility to improve upon inaccurate user estimates by incorporating an online feedback loop between workflow scheduling, resource usage prediction, and measurement.**

Our approach can learn resource usage of arbitrary type; in this paper, we demonstrate its effectiveness by predicting peak memory usage of tasks, as it is an especially sensitive resource type that leads to task termination if underestimated and decreased throughput if overestimated.

We compare online versions of standard machine learning models for peak memory usage prediction and analyze their interactions with different workflow scheduling strategies. By means of extensive simulation experiments, we found that the proposed feedback mechanism improves resource utilization and execution times compared to typical user estimates.

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

I. Introduction

The scale of analysis workflows in science and commercial applications grows rapidly. For instance, bioinformatics, material science, astronomy, and earth science frequently require the processing of large data sets with various command-line tools arranged in a complex analysis pipeline [1], [2]. To scale these workflows to large problem sizes, it is necessary to distribute the individual steps across compute resources.

A common approach to parallelization data-intensive workloads is task-based computing, e. g., by partitioning the data for a single analysis step into chunks that are processed in parallel. For compute-intensive workloads, parameter sweeps are a common pattern that lead to large numbers of tasks. To orchestrate distributed execution of tasks with dependencies, scientific workflow management systems (SWMS) can be used. SWMS accept workflow specifications in a specific language and execute them on different compute infrastructures; examples are Pipeline Script for Nextflow [3], DAX for Pegasus [4], or Cuneiform [5] for Saasfee [6].

Due to the complexity of executing tasks on distributed compute infrastructures, many workflow systems are designed as layers on top of distributed resource management software. For example, Nextflow comes with executors that can run a workflow on various batch scheduling systems, such as LSF or HTCondor, which is also the underlying resource manager for the Pegasus SWMS.

Batch scheduling systems require users to estimate task resource usage and often enforce these estimates by killing tasks that exceed them. This incentivizes users to overestimate resource usage, which is especially problematic for peak memory estimates: Compared to overestimated runtimes, where excess time can be allocated to other tasks after a task finishes, excessively allocated memory is not usable by other tasks during the runtime of a task and is thus wasted. Depending on the variety and complexity of analysis steps, workflow task frequently have vastly heterogeneous resource requirements, and optimizing resource allocations is key to reduce workflow execution times.

Even experienced users may not be able to accurately estimate resource usage for their programs [7]. As an alternative, estimates based on past performance measurments have been proposed [8]. However, the resource usage of a task might strongly depend on the output of its ancestor tasks, which is reliably known only at runtime. In addition, data analysis workflows in a scientific context are “routinely unique” [9], meaning that past resource usage measurements on similar input data and compute infrastructure might not be available, contrary to repetitive production batch jobs in commercial scenarios [10].

Here, we investigate a system design that allows for online learning task resource consumption. Our approach is based on a feedback loop (Figure 1) where scheduling decisions affect predictor re-training and assessment, and predictions affect scheduling decisions. This allows estimates to be continuously updated and to incorporate the size of intermediate result sets for predictions, whereas offline schedulers have to ignore or guess intermediate result set sizes in advance. In summary, we make the following contributions:

- Contrary to prior work, which views prediction and scheduling as separate problems, we show that there exist significant interaction effects.
- We show that statistical methods can serve as a reliable fallback in cases where user estimates align poorly with actual resource usage.

- We provide insights on the impact of improved prediction accuracy on memory utilization and makespan.

The paper is structured as follows: Section 2 covers background and related work, Section 3 describes our online scheduling and prediction approach, Section 4 describes online prediction models, Section 5 covers the experimental setup, and Section 6 presents our results. We conclude with a discussion and future work in Section 7.

II. Background and Related Work

We briefly introduce the notion of a scientific workflow, prior work on resource usage prediction and scheduling, and discuss the potential for improving resource usage with an online learning approach.

A. Scientific Workflows

A scientific workflow specifies how different command-line tools are combined to analyze or produce large amounts of data in a parallel fashion [1]. These workflows often comprise programs with widely varying resource requirements ranging from simple data transformations to complex analysis steps. Executing the same tool with varying input data or parameters yields a set of similar tasks, which we collectively refer to as an *abstract task*. Figure 2 shows an exemplary workflow that illustrates the typical repetitive structure of scientific workflows [11], where each abstract task corresponds to one level in the graph.

Scientific workflows treat both tasks and data as black boxes, which contrasts with distributed computing approaches centered around a single language or data model, such as Spark [12]. Each analysis step is encapsulated as an invocation of a program which processes input files in an arbitrary format, yielding maximum flexibility in the choice of tools and data formats.

B. Task Sizing and Distributed Resource Management

The task sizing problem [13] refers to selecting the right amount of resources for a task. This problem arises whenever a scientific workflow management systems (SWMS) obtains compute resources by negotiating with a resource manager. For instance, the SWMS Nextflow [3] supports job schedulers like LSF or Slurm [14] to run workflows on HPC resources. A task is executed by submitting a corresponding job to the job scheduler, along with the requested resources, e. g., CPU cores, memory, or an upper bound on the execution duration of the job. The required estimates have to be included by the user in Nextflow's workflow description format. To handle suboptimal initial estimates, Nextflow allows the user to specify how to increase, e. g., the amount of memory with every task failure. In contrast, we propose to derive this information at run time from the observed memory usage of successful tasks, which reduces the number of successive trial and error allocations.

Many batch scheduling systems provide detailed resource usage information, such as CPU utilization or memory usage. In addition, scientific workflow management systems often collect resource usage measurements [11], [15], [16]. In our model, we immediately use this information to improve resource allocation for the remaining workflow.

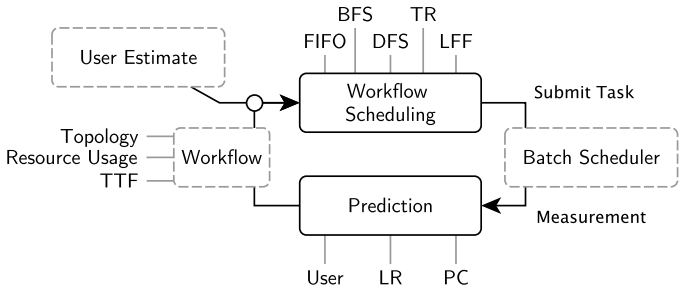

Fig. 1: The Scheduling-Prediction feedback loop. Dashed lines indicate components beyond the control of our approach.

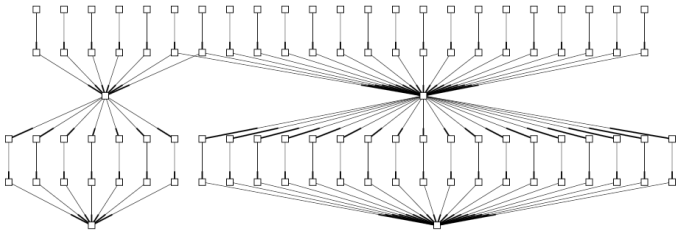

Fig. 2: Ligo is a typical embarrassingly parallel workflow. Tasks that execute the same program on different data are placed at the same vertical position.

C. Resource Consumption Prediction and Scheduling

Most workflow scheduling research assumes that resource usage estimates, usually task execution duration and communication times, are given [17]. The problem of obtaining such estimates is considered a separate issue [8]. On the other hand, research on predicting the resource usage of computational workloads usually focuses on sophisticated methods to learn from and extrapolate historical performance measurements, without considering interactions with a scheduler. Our work considers a novel scenario where resource estimates change according to scheduling decisions.

Classical workflow scheduling algorithms [18], [19] use task execution and file transfer duration estimates to compute a schedule before execution. We consider memory as an additional dimension, motivated by memory-intensive workflows [2], [20] and the requirements of batch schedulers and resource managers like YARN to specify the amount of main memory for each job/task. Dynamic scheduling algorithms decide at runtime which task to place where and often lack the ability to plan ahead and incorporate the entire DAG into their decisions. Scheduling with both multi-variate resource demands and dependencies between tasks is rarely studied, despite its importance [21]. An overview of workflow scheduling algorithms is given in [22], but none considers the online learning scenario covered here.

Since static schedulers depend on the accuracy of runtime estimates, prior studies have evaluated the scheduling quality

under fixed degrees of prediction accuracy [23], [24], [25], [26], [22]. We are interested in the novel scenario where resource usage estimates are changing at runtime, in response to scheduling decisions. Silva et al. [27] showed that applying a task runtime prediction model during workflow execution yields improvements compared to estimating file sizes and runtimes in advance. However, in their work, no learning takes place at runtime, since the prediction model is built from prior executions before the workflow starts, and only the actual intermediate result set sizes are passed to the pre-built model at runtime. Our prediction models are continuously updated and applied at runtime.

III. Feedback-Based Scheduling

In this section, we describe our approach to feedback-based scientific workflow scheduling. Figure 1 the the components and their interactions in our architecture. We control the workflow scheduling component and resource usage prediction component, as described in the following two subsectinos. In addition, the architecture comprises three components that are beyond our control: (1) the batch scheduler used for task execution, (2) the workflow specification, and (3) the initial user estimates.

The basic idea is to immediately incorporate resource usage measurements collected from successful task executions into online prediction models to optimize the resource allocation for the execution of the remaining workflow. Resource usage measurements constitute the first type of feedback. Task successes and failures constitute a second type of feedback that influences the order in which tasks are considered for execution, as described in the following.

A. Workflow Scheduling

We consider different workflow scheduling heuristics for the use within our feedback-based scheduling approach. The core responsibility of the workflow scheduler is to select tasks to be executed when resources are scarce and the workflow cannot run at its full degree of parallelism, besides enforcing the dependency constraints on task execution order. Here we present basic heuristics (see Table I) and introduce our least-finished-first (LFF) heuristic, which is tailored to the online prediction approach.

A common approach to workflow scheduling is to execute tasks in a first-in-first-out (FIFO) order. When a task becomes eligible for execution, i. e., its parent tasks have finished, the task is queued for execution. When resources are available, tasks are taken from the head of the queue for execution.

We compared this strategy to schedulers that prioritize tasks based on workflow topology. The BFS and DFS heuristics use a task's depth as priority, i. e., the longest path to a task without predecessors. The BFS heuristic executes ready tasks with a low depth first. This is similar to a breadth-first search through a graph. A possible advantage of this strategy is that it tends to produce larger sets of ready tasks, which in turn can lead to higher degrees of parallelism. In contrast, the DFS strategy prioritizes tasks with high depth, which resembles a depth-first search. This has the potential advantage of running tasks of newly available abstract task types as early as possible, which produces resource usage measurements for abstract tasks earlier.

The topology-rank (TR) heuristic is based on the rank of a task, i. e., the longest path to a task without children. This focus on the ends of the workflow incorporates a lookahead component into scheduling that tends to favors task with more downstream work. A similar concept is used in the popular HEFT [28] heuristic, which uses average task runtimes as weight. We do not assume task runtime estimates to be available when starting a task and consider only topology.

Finally, we propose a least-finished-first (LFF) heuristic, which favors tasks that belong to abstract tasks of which the least number of tasks have completed, i. e., for which the least training data is currently available. This follows a similar rationale like the DFS heuristic. But instead of looking only at the topology, LFF considers the number of successfully executed tasks and thus the amount of training data for the online prediction models. Because the success of tasks depends on prediction accuracy, the available resources, and the relative runtimes of the tasks, this yields significantly different task execution orders compared to DFS.

B. Resource Usage Feedback

To predict peak memory usage for tasks, we train one model (e. g., linear regression, see Section IV) per abstract task. Every time a task completes successfully, the model for that abstract task is retrained incrementally with the observed peak memory usage and the size of the input files. The updated model is then used to predict the peak memory usage of the remaining tasks of that type.

When a task fails because a model has underestimated its peak memory usage, we ensure that the task is eventually assigned sufficient memory by repeatedly doubling the allocated memory. For any task's first attempt, the predictor is applied if it is ready, otherwise an estimate provided by the user is used. If the task has failed before, we distinguish two cases. If the faulty peak memory estimate was based on an initial guess and the predictor has become ready in the meantime, the predictor is applied. If the peak memory estimate was based on a prediction, the memory assigned to the task is repeatedly doubled until the task succeeds or fails on the maximum possible amount of memory, in which case the workflow cannot be executed with the given resources.

C. Task Failure Feedback

We found that incorporating the number of failed attempts into scheduling has a stong impact on throughput and resource utilization.

We thus derived two variants from each of the topology-aware scheduling heuristics. The *persevere* variant prioritizes tasks by with a high number of failed attempts. The intuition is that failed executions do not contribute new peak memory observations and thus re-trying such tasks persistently helps

TABLE I: Workflow Scheduling Heuristics Overview.

Scheduler	First Priority	Second Priority
FIFO	Earliest queue insertion time	None
BFS	Shortest path to entry (shallow tasks)	Failed attempts
DFS	Longest path to entry (deep tasks)	Failed attempts
TR	Task with longest path to exit (rank)	Failed attempts
LFF	Abstract task with least finished tasks	Failed attempts

to gather the additional data needed to improve the inaccurate prediction model. The *postpone* variant executes those tasks with the least number (usually zero) of failed attempts first, in an attempt gather additional data from better predictable tasks.

After a task has failed, it becomes ready again and is re-inserted in a scheduler's priority queue. The number of failed attempts is used as a secondary criterion by first sorting according to depth or rank and then by the number of failed attempts. The only exception is the FIFO scheduler, which treats a failed task as if it became ready for the first time, which we refer to as the *time of arrival* failure handling strategy, which is similar to the *postpone* strategy. For an overview of the simulated variants, refer to II.

IV. Memory Usage Prediction

We experimented with two approaches for predicting peak memory usage: (1) based on the distribution of memory usages seen so far, and (2) by correlating it to input size.

A. Basic Prediction with Percentiles and Regression

Our first method, the percentile predictor predicts peak memory usage of a task as a percentile of the peak memory usages of all successful task executions so far. We use the 95th percentile (PC 95) as a conservative estimator that predicts memory usage close to the largest observed memory usage. We also consider the median memory usage (PC 50) as a more optimistic predictor. Although this leads to an expected half of the tasks failing on their first attempt, it may still work well if tasks fail quickly.

Our second method is based on the observation that memory usage correlates to input size. We evaluate a linear regression predictor (LR) that relates the sum of a task's input files to its peak memory consumption. This predictor is less conservative than the percentile predictor and aims at tighter memory allocations rather than reducing the number of failed task attempts. We update the regression coefficients incrementally with every new observation.

We start using the PC and LR predictors as soon as 1 or 2 successful observations have been collected, respectively.

B. Offsetting for Linear Regression

Since we expect the consequences of predicting slightly too little memory to be more severe than predicting slightly too much memory, we also developed more conservative variants of the regression approach by adding various offset terms.

The offset terms are adaptive in the sense that they depend on the prediction errors made by the current fitted model f. Assume a set of (input size, peak memory usage) observations $D = \{(x_1, y_1), \ldots, (x_n, y_n)\} \subset X \times Y$ and a prediction function $f : X \to Y$. We derive more conservative estimates from f by adding an offset $o(f, D)$ that depends on the function and the training data.

The *LR mean* $\pm$ approach adds as offset $o(f, D)$ the sample standard deviation, i. e., the expected difference between the predicted and the true peak memory usage.

$$o(f, D) = \sqrt{\frac{1}{n-1} \sum_{(x,y) \in D} (f(x) - y)^2} \tag{1}$$

The *LR mean* $-$ approach is similar to the previous approach, but considers only underpredictions, i. e., negative errors. To this end, we divide by $m = |\{(x, y) \in D | f(x) < y\}|$.

$$o(f, D) = \sqrt{\frac{1}{m-1} \sum_{(x,y) \in D:\ f(x) < y} (f(x) - y)^2} \tag{2}$$

The *LR max* $-$ approach is the most conservative approach and adds the largest seen underprediction to the next prediction.

$$o(f, D) = \max_{(x,y) \in D} y - f(x) \tag{3}$$

After a new task completes, the regression coefficients are updated incrementally and the offset is recomputed.

V. Experimental Setup

This section covers the Memory Allocation Quality metric, the synthetic workflows used for evaluation, and general simulation parameters used in the experiments.

A. Memory Allocation Quality

To characterize the quality of a scheduler's memory allocation decisions, we introduce the Memory Allocation Quality (MAQ) metric. The metric indicates the amount of wasted memory that results from underestimating or overestimating task memory usage. In contrast to average memory utilization, MAQ does not depend on the compute infrastructure or task precedence constraints, which may enforce inevitable periods of low utilization.

We measure memory usage and wastage in megabyte-seconds, i. e., the product of the amount of allocated memory and the duration of the allocation. Since allocating too little memory leads to task failure, we consider all of the memory allocated to this attempt as wasted. When allocating too much memory, we consider the difference between allocated memory and the task's peak memory usage as wasted. We define the amount of wasted memory W as the sum of wasted megabyte-seconds. Memory allocation quality is defined as the used megabyte-seconds U divided by total allocated megabyte-seconds. Accordingly, 100% corresponds to no wastage.

$$\text{MAQ} = \frac{U}{U + W} \tag{4}$$

B. Synthetic Workflows

We generate workflows using the Pegasus Workflow Generator [31], which synthesizes workflows from five different domains. We extended the workflow generator to generate a hypothetical memory model for each abstract task from which peak memory consumption values are sampled. The design of our memory models is based on the observation that memory consumption may vary strongly within and across abstract tasks [11], [32] and tasks frequently consume memory that is linearly related to the sum of input file sizes [20].

Each memory model describes the relationship between input size and memory consumption for a particular abstract task. With a chance of 50%, an abstract task exposes a linear relationship to the sum of input file sizes, otherwise input size and memory consumption are sampled independently. The error term $\mathcal{N}(0, v)$ adds variation in memory consumption that is not explained by input size. We first generate a random mean μ, standard deviation σ, error v, and slope m_1 for the model according to Table II. We then sample input sizes and memory consumption such that they follow Equation 5.

$$y(t) = m_1 x(t) + m_0 + \mathcal{N}(0, v) \tag{5}$$

We consider different relative times to failure (TTF) for tasks in case of insufficient memory allocation. In reality, the time to failure depends on the task's memory usage profile, i. e., at which point in time the memory usage first exceeds the assigned amount of memory. In an optimistic scenario tasks fail early, reducing the resource wastage resulting from underprediction. In a pessimistic scenario, tasks execute almost to completion before they run out of memory. We used TTFs of 0.1, 0.5, and 0.9 in our experiments.

We generated a total of 500 workflows, 100 each from the workflow types Sipht, Montage, Genome, Ligo, and Cybershake [11]. Each workflow comprises 1000 tasks. After generating peak memory consumption values for each task, the task runtimes are scaled such that each workflow consumes exactly one terabyte-week of main memory. This is convenient for evaluating experiments because the total amount of work, i. e., runtimes and peak memory consumption, across workflows may differ by orders of magnitude.

C. Generating User Estimates

To provide reference values to compare our prediction models to, we use two approaches to generate hypothetical user estimates for our synthetic workflows.

The first approach is based on rounding: Given the true maximum memory usage for an abstract task, the *Power 2* approach computes the user estimate as the nearest power of two, in Gigabytes. For instance, if the actual peak memory usage across the tasks of an abstract task is 1.2 GB, the estimate for this abstract task is assumed to be 1 GB. This reflects the fact that user estimates are typically coarse-grained and round numbers [29]. As a variant, we use the same method but round to the nearest power of ten Gigabytes (Power 10).

As a second method, we generate hypothetical user estimates from the true peak memory usage of each abstract task by applying a constant factor (*Max* approach). For instance, Max $+20\%$ estimates the memory usage of a task as the maximum memory usage of the abstract task multiplied by 1.2. This approach reflects the fact that users tend to provide conservative estimates in practice [30].

TABLE II: Simulation Parameters

Group	Parameter	Value
Workflows	Type	Cybershake, Genome, Ligo, Montage, Sipht
	Number	500 (100 per type)
	Tasks	1000 per workflow
	Time to failure	0.1, 0.5, 0.9
Memory Models	Average	$\mu = \mathcal{U}(1\text{GB}, 1\text{TB})$
	Std. Deviation	$\sigma = \mathcal{U}(0.03, 0.1)\mu$
	Error	$v = \mathcal{U}(0.1, 0.75)\sigma^2$
	Slope	$\mathcal{U}(1, 10)$
Resources	Cores	128
	RAM	1952 GB
Execution	Scheduling	FIFO, BFS, DFS, TR, LFF
	Predictor	PC, LR, User Estimates (Power, Max)

D. Simulation Setup

Our simulations are conducted in DynamicCloudSim [33], a simulation environment based on the CloudSim [34] software. DynamicCloudSim adds workflow execution functionality to CloudSim, such as data structures for workflows and algorithms to schedule them. We added the proposed feedback-based scheduling mechanism and the online prediction models from Section IV to the framework.

In our experiments, a simulation scenario corresponds to a combination of a specific workflow (including its memory usage profile), a scheduling algorithm, a predictor, and a TTF value. Our simulations assume a shared resource pool of 128 CPU cores and 1952 GB of main memory. Since all workflows allocate a total of one terabyte-week of main memory, a lower bound on workflow execution time is 7 terabyte-days$/1.952$ Gigabyte ≈ 3.5 days. The simulation parameters are summarized in Table II.

VI. Results

In the following, we report simulation results on the relationship between resource efficiency and speed, the best performing scheduling policies, and demonstrate interaction effects between scheduling policy and prediction model.

A. Relationship between MAQ and Makespan

Figure 3 shows how MAQ relates to makespan for a sample of 50,000 simulated scenarios. It is apparent that MAQ limits the execution time from below, which reflects the memory bottleneck induced by the compute infrastructure. Although higher MAQ is correlated to lower makespans, makespans expose considerable variation across scenarios with identical MAQ. This is due to different workflow scheduling policies, the different graph structures of the different workflow types, and different TTF values. These factors will be analyzed in the next sections. In the following, we will use MAQ and makespan as separate evaluation metrics.

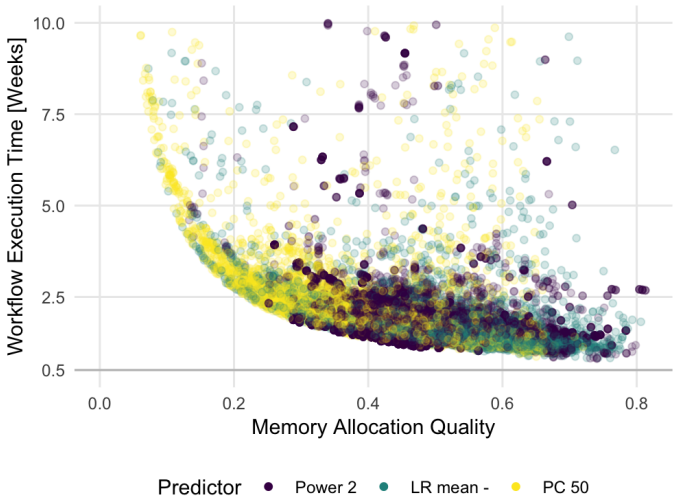

Fig. 3: The relationship between MAQ and execution time on a sample of 50,000 simulation scenarios for three peak memory prediction methods. Power 2 corresponds to a type of user estimates, LR corresponds to a linear regression model, and PC 50 corresponds to predicting memory usage as the median of previously seen memory consumptions. High MAQ is necessary but not sufficient for low makespan.

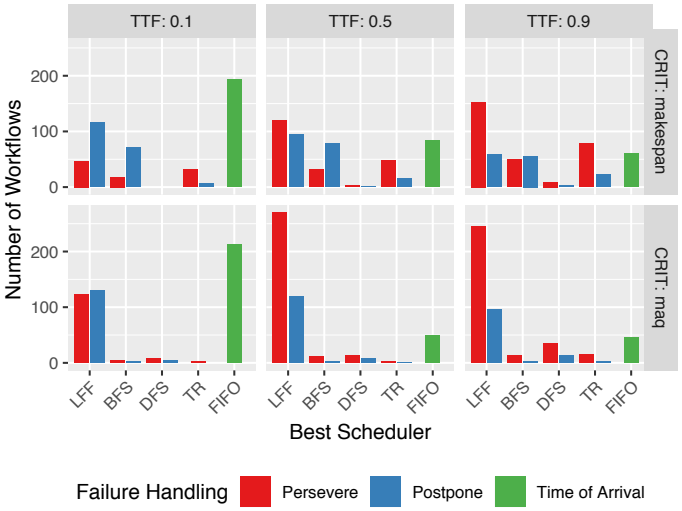

Fig. 4: Which workflow scheduler performs best depends whether MAQ or makespan should be optimized, the time to failure, and the failure handling strategy.

Across the selected predictors, few simulation scenarios exceed a MAQ of 80%. The maximum observed MAQ across all simulation scenarios is 87.1%. The PC 50 predictor is responsible for most of the worst simulation scenarios, because it leads to many failed task attempts, that have severe impact on makespan when the relative time to task failure is high. User estimates, i. e., the Power 2 predictor performs better, but frequently show MAQ values below 50%. The simulation scenarios featuring the LR mean− approach concentrate favorably in a region of high MAQ and low makespan, but include some extreme scenarios with very low MAQ and high makespans.

B. Best Scheduling Policies

Next, we analyze which workflow scheduling heuristics perform best. For each workflow under each TTF, we determine the combination of predictor and scheduling heuristic that achieves the lowest makespan or highest MAQ, respectively[1]. Figure 4 shows the number of workflows (out of the 500 workflows) for which each scheduling heuristic performs best. If tasks fail quickly, the commonly used FIFO strategy performs best, both for optimizing makespan and MAQ. However, the picture changes dramatically for larger time to failure values. In these scenarios, LFF performs best, both for makespan and MAQ. Overall, the persevere failure handling strategy performs better than the postpone strategy, likely because failures indicate that user estimates or predictions are inaccurate, and a certain degree of persistence is necessary to bring tasks to completion, which allows to retrain prediction models. Interestingly, neither of the topology-based scheduling heuristics DFS, BFS, and TR are competitive. In the following, we focus our analysis on the best scheduling heuristics, FIFO and LFF.

[1]In 98% of the cases, there is a single best performing combination. If ties between combinations exist, we count all winning schedulers.

C. Interaction Between Scheduling and Prediction

In our feedback-based scheduling model, prediction and scheduling strongly influence each other. Scheduling decisions control which training data becomes available first, and prediction errors modify task priorities. Figure 5 shows the best combinations of scheduler and makespan, as the number of workflows where that combination achieves the lowest makespan or best MAQ, respectively. This shows that FIFO's advantage in the low TTF scenarios strongly depends on the choice of the PC 50 predictor. For LFF, moderately conservative prediction models such as LR mean− or PC 95 work best. Figure 6 shows how the memory allocation quality differs for each workflow when using a regression approach compared to fixed user estimates, both with the LFF Perservere scheduler.

Figure 7 shows the worst performing combinations. These feature mainly the optimistic predictors PC 50 in combination with a postpone failure handling strategy, which leads to large numbers of failed tasks attempts, without learning about resource usage and improving prediction models.

VII. Conclusion and Future Work

We demonstrated the potential of workflow execution engines that feature joint workflow scheduling and resource usage prediction. Compared to requesting user estimates prior to workflow execution, the proposed feedback-based scheduling offers the advantage of adapting to the true resource usage of tasks during workflow execution. We also demonstrated that

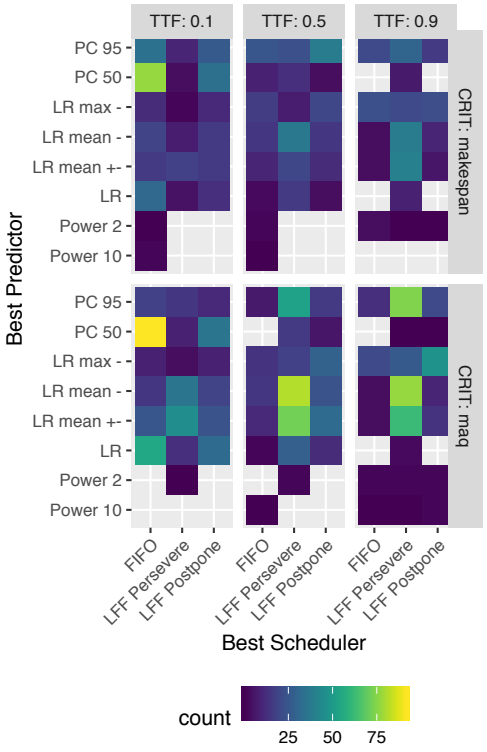

Fig. 5: Number of workflows for which a given scheduler-predictor combination performs best with respect to workflow execution time (top) and memory allocation quality (bottom).

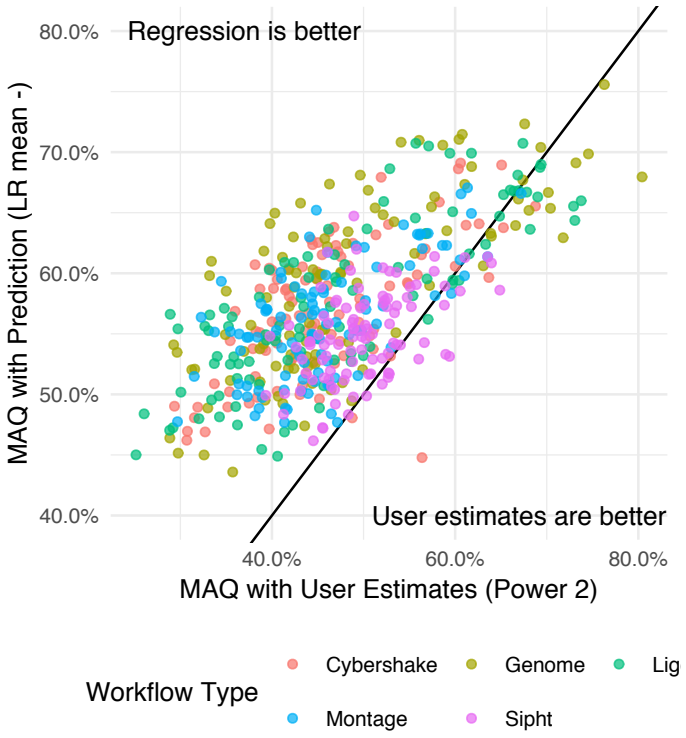

Fig. 6: Comparison of the memory allocation quality achieved by user estimates and a linear regression approach. Each point represents one workflow, the scheduler was fixed to LFF Persevere.

in a feedback-based design, scheduling and prediction, which are traditionally thought of as separate independent problems, expose significant interaction effects.

Regarding online prediction, we showed how to modify a standard linear regression model to construct appropriately conservative predictors that allow to take into account input file sizes as available only during workflow execution.

We plan to extend the architecture such that the scheduler learns models of runtime and also time to failure for each abstract task. It also seems promising to incorporate information gathered from failed task attempts as censored data since they already provide a lower bound on the true peak memory usage and run time of the task.

In some scenarios, resource demands may not be hard constraints, but allow trading-off between resources. Some abstract tasks can be assigned a varying number of CPU cores, leading to different runtimes. Similarly, some tasks may be able to complete with lesser memory than requested, taking a runtime penalty for using swap. Taking into account malleable resource demands adds further complexity to the scheduling problem but also provides additional degrees of freedom for the scheduler to reduce makespan.

A related question is whether additional information and predictions could be used for more sophisticated scheduling. In most situations, the scheduler can sacrifice memory allocation quality to reduce the chance of task failure. Which one is favorable depends on the specific loss of time associated with a task attempt, its importance for making progress in the workflow, the current pressure on resources, etc. For instance, it seems reasonable to assign more memory to tasks that have many children or need to be completed before workflow execution can progress (tasks that are synchronization barriers). This could be complemented by online learned estimates of task runtimes, both in case of failure and success, to estimate the risk associated to underestimating the memory usage of specific tasks.

Acknowledgements

Carl Witt received funding by Deutsche Forschungsgemeinschaft through the SOAMED graduate school (GRK 1651).

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] A. Rheinländer, M. Lehmann, A. Kunkel, J. Meier, and U. Leser, "Potential and Pitfalls of Domain-Specific Information Extraction at Web Scale," in *SIGMOD Conference*. Humboldt-Universitat zu Berlin, Berlin, Germany, 2016, pp. 759–771.

[3] 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.

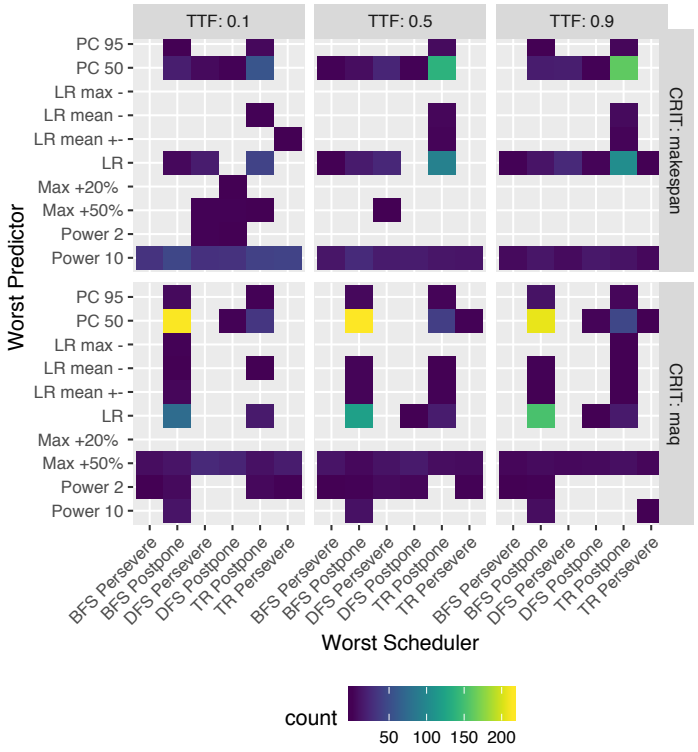

Fig. 7: Number of workflows for which a given scheduler-predictor combination performs worst with respect to workflow execution time (top) and memory allocation quality (bottom).

[4] 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.

[5] J. Brandt, W. Reisig, and U. Leser, "Computation semantics of the functional scientific workflow language Cuneiform," *Journal of Functional Programming*, vol. 27, p. 373, Oct. 2017.

[6] 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.

[7] A. W. Mu'alem and D. G. Feitelson, "Utilization, Predictability, Workloads, and User Runtime Estimates in Scheduling the IBM SP2 with Backfilling," *IEEE Transactions on Parallel and Distributed Systems*, vol. 12, no. 6, pp. 529–543, Jun. 2001.

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

[9] J. Chang, "Core services: Reward bioinformaticians," *Nature*, vol. 520, no. 7546, pp. 151–152, Apr. 2015.

[10] O. Alipourfard, H. H. Liu, J. Chen, S. Venkataraman, M. Yu, and M. Zhang, "CherryPick - Adaptively Unearthing the Best Cloud Configurations for Big Data Analytics," in *USENIX Symposium on Networked Systems Design and Implementation*. 13th USENIX Symposium on Networked Systems Design and Implementation, 2017.

[11] G. Juve, A. L. Chervenak, E. Deelman, S. Bharathi, G. Mehta, and K. Vahi, "Characterizing and profiling scientific workflows." *Future Generation Computer Systems*, vol. 29, no. 3, pp. 682–692, 2013.

[12] M. Zaharia, M. Chowdhury, T. Das, A. Dave, J. Ma, M. McCauly, M. J. Franklin, S. Shenker, and I. Stoica, "Resilient Distributed Datasets - A Fault-Tolerant Abstraction for In-Memory Cluster Computing." *NSDI*, 2012.

[13] 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.

[14] A. B. Yoo, M. A. Jette, and M. Grondona, "SLURM: Simple Linux Utility for Resource Management," in *Computer Performance Engineering*. Berlin, Heidelberg: Springer Berlin Heidelberg, 2003, pp. 44–60.

[15] T. Ohta, T. Tanjo, and O. Ogasawara, "Accumulating computational resource usage of genomic data analysis workflow to optimize cloud computing instance selection," *bioRxiv*, p. 456756, Oct. 2018.

[16] Tracing & visualisation — Nextflow 18.10.1 documentation. [Online]. Available: https://www.nextflow.io/docs/latest/tracing.html

[17] B. Jennings and R. Stadler, "Resource Management in Clouds - Survey and Research Challenges." *J. Network Syst. Manage.*, vol. 23, no. 3, pp. 567–619, 2014.

[18] Y.-K. Kwok and I. Ahmad, "Static scheduling algorithms for allocating directed task graphs to multiprocessors," *ACM Computing Surveys*, vol. 31, no. 4, pp. 406–471, Dec. 1999.

[19] M. A. Rodriguez and R. Buyya, "A taxonomy and survey on scheduling algorithms for scientific workflows in IaaS cloud computing environments." *Concurrency and Computation: Practice and Experience*, vol. 29, no. 8, 2017.

[20] M. Bux, J. Brandt, C. Witt, J. Dowling, and U. Leser, "Hi-WAY: Execution of Scientific Workflows on Hadoop YARN," in *International Conference on Extending Database Technology*, 2017.

[21] R. Grandl, S. Kandula, S. Rao, A. Akella, and J. Kulkarni, "GRAPHENE - Packing and Dependency-Aware Scheduling for Data-Parallel Clusters." *OSDI*, 2016.

[22] M. Malawski, G. Juve, E. Deelman, and J. Nabrzyski, "Algorithms for cost- and deadline-constrained provisioning for scientific workflow ensembles in IaaS clouds," *Future Generation Computer Systems*, vol. 48, pp. 1–18, Jul. 2015.

[23] R. Armstrong, D. Hensgen, and T. Kidd, "The relative performance of various mapping algorithms is independent of sizable variances in run-time predictions," in *Seventh Heterogeneous Computing Workshop (HCW'98)*. IEEE Comput. Soc, 1989, pp. 79–87.

[24] R. Gibbons, "A Historical Application Profiler for Use by Parallel Schedulers." *Job Scheduling Strategies for Parallel Processing*, vol. 1291, no. Chapter 3, pp. 58–77, 1997.

[25] H. Casanova, A. Legrand, D. Zagorodnov, and F. Berman, "Heuristics for scheduling parameter sweep applications in grid environments," *9th Heterogeneous Computing Workshop (HCW 2000)*, pp. 349–363, 2000.

[26] 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.

[27] R. Ferreira da Silva, G. Juve, M. Rynge, E. Deelman, and M. Livny, "Online Task Resource Consumption Prediction for Scientific Workflows," *Parallel Processing Letters*, vol. 25, no. 03, p. 1541003, 2015.

[28] H. Topcuoglu, S. Hariri, and M.-Y. Wu, "Performance-effective and low-complexity task scheduling for heterogeneous computing," *IEEE Transactions on Parallel and Distributed Systems*, vol. 13, no. 3, pp. 260–274, Mar. 2002.

[29] 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.

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

[31] R. Ferreira da Silva, W. Chen, G. Juve, and K. Vahi, "Community resources for enabling research in distributed scientific workflows," in *e-Science and Grid Computing, IEEE International Conference on*, 2014.

[32] A. Singh, A. Rao, S. Purawat, and I. Altintas, "A machine learning approach for modular workflow performance prediction," in *Workflows in Support of Large Scale Science*. New York, New York, USA: ACM Press, 2017, pp. 1–11.

[33] M. Bux and U. Leser, "DynamicCloudSim: Simulating heterogeneity in computational clouds," *Future Generation Computer Systems*, 2015.

[34] R. N. Calheiros, R. Ranjan, A. Beloglazov, C. A. F. De Rose, and R. Buyya, "CloudSim: a toolkit for modeling and simulation of cloud computing environments and evaluation of resource provisioning algorithms," *Software: Practice and Experience*, vol. 41, no. 1, pp. 23–50, Aug. 2010.