Reshi: Recommending Resources for Scientific Workflow Tasks on Heterogeneous Infrastructures

Jonathan Bader*, Fabian Lehmann‡, Alexander Groth*, Lauritz Thamsen§,
Dominik Scheinert*, Jonathan Will*, Ulf Leser‡, Odej Kao*
* {jonathan.bader, a.groth, dominik.scheinert, will, odej.kao}@tu-berlin.de, Technische Universität Berlin, Germany
‡ {fabian.lehmann, leser}@informatik.hu-berlin.de, Humboldt-Universität zu Berlin, Germany
§ lauritz.thamsen@glasgow.ac.uk, University of Glasgow, United Kingdom

***Abstract*—Scientific workflows typically comprise a multitude of different processing steps which often are executed in parallel on different partitions of the input data. These executions, in turn, must be scheduled on the compute nodes of the computational infrastructure at hand. This assignment is complicated by the facts that (a) tasks typically have highly heterogeneous resource requirements and (b) in many infrastructures, compute nodes offer highly heterogeneous resources. In consequence, predictions of the runtime of a given task on a given node, as required by many scheduling algorithms, are often rather imprecise, which can lead to sub-optimal scheduling decisions.**

We propose Reshi, a method for recommending task-node assignments during workflow execution that can cope with heterogeneous tasks and heterogeneous nodes. Reshi approaches the problem as a regression task, where task-node pairs are modeled as feature vectors over the results of dedicated micro benchmarks and past task executions. Based on these features, Reshi trains a regression tree model to rank and recommend nodes for each ready-to-run task, which can be used as input to a scheduler. For our evaluation, we benchmarked 27 AWS machine types using three representative workflows. We compare Reshi's recommendations with three state-of-the-art schedulers. Our evaluation shows that Reshi outperforms HEFT by a mean makespan reduction of 7.18% and 18.01% assuming a mean task runtime prediction error of 15%.

***Index Terms*—Resource Management, Scientific Workflow, Profiling, Heterogeneous Cluster Resources, Scheduling**

I. Introduction

The popularity of scientific workflows increases and has become essential in many scientific domains like bioinformatics, geoinformatics, or physics [1]–[8]. For example, as new genome sequencing machines, satellites, and microscopes produce more and fine-granular data, workflows become more complex and have to deal with larger datasets. Accordingly, the execution of a single workflow can take multiple days or even weeks on big clusters [2], [6], [8].

Scientific workflows consist of a set of black box tasks and a set of directed edges which describe the dependencies between the tasks. Accordingly, a predecessor task has to finish before the successor task can start. The tasks communicate via files and the output of a predecessor task is the input of its successor. Workflows following this definition can be represented as a Directed Acyclic Graph (DAG).

As workflows can comprise thousands of black box task instances, scientific workflow management systems (SWMS) like Pegasus [4] or Nextflow [3] are used to reduce a scientist's manual configuration effort by, for example, execution monitoring, automatic parallelization, or improved reusability. With the help of resource managers like Kubernetes [9] or Slurm [10], the SWMS schedule the tasks to the available cluster nodes.

Some clusters support multiple purposes and, therefore, consist of heterogeneous nodes. Other clusters get upgraded over time or get partially replaced with newer components once hardware failures occur, leading to a heterogeneous infrastructure [6], [8], [11]–[13]. However, even nodes with the same amount of CPU cores, memory, or disk space can yield highly different runtimes, for example when memory latencies, clock speeds, or read/write rates differ. Furthermore, different tasks often have different resource demands. For instance, some tasks are very CPU-heavy or memory-intensive, while others mostly read and write to the disk [2], [6], [14]. Hence, cluster resource heterogeneity can be exploited by schedulers for optimized task placements.

Despite the extensive research on scheduling algorithms that incorporate heterogeneity aspects, there is a lack of use in existing real-world systems. The missing uptake can be explained with a lack of accurate task runtime estimates since even state-of-the-art estimators yield a median prediction error between 10% and 20% [8], [14]–[17]. In practice, popular resource managers handle tasks as black boxes and, therefore, often resort to simpler scheduling approaches like Round-robin or fair scheduling [6], [8], [18], [19]. For example, Kubernetes uses a Round-robin strategy to assign tasks to nodes [18], while YARN does it in a fair fashion [19].

In this paper, we propose a recommender system for heterogeneous infrastructures (Reshi) to rank machines for scientific workflow tasks without being affected by error-prone runtime estimates. Our approach consists of four steps. During the first step, we profile the existing cluster hardware with a set of benchmarks to gather detailed performance insights. Then, Reshi analyzes existing task performance metrics from historical workflow executions. In case of no existing executions a quick workflow profiling with highly reduced inputs can be conducted to gather task execution metrics [8], [20]. Based on the infrastructure details and the task-performance data, we train a regression tree model for our recommender system

to dynamically rank the nodes for each task. Through these ranks, we avoid relying on accurate runtime estimates and data-dependent resource usage predictions. The ranks can then be used to schedule workflow tasks without assuming accurate runtime knowledge.

We provide a practical implementation of our approach as a prototype, comprising a cluster benchmarking tool for heterogeneous clusters and the recommender system for ranking the tasks to the infrastructure. For our evaluation, we benchmarked and profiled 27 different AWS EC2 machines to evaluate our prototype using three real-world scientific workflows from the popular nf-core framework [3]. Further, we provide a WorkflowSim extension[1] that enables the lookup of actually measured runtimes and the inclusion of prediction errors in the simulation. Based on this simulation extension, our comparison with the state-of-the-art schedulers HEFT, MinMin, and Round-robin shows that Reshi helps the SWMS to achieve low workflow makespans while being independent from task prediction errors.

II. Related Work

In this section, we first cover the scheduling of scientific workflows on heterogeneous clusters. Then, we focus on runtime prediction approaches since their task runtime estimates frequently serve as the input for related scheduling approaches [8].

A. Scheduling Scientific Workflows on Heterogeneous Clusters

Existing approaches either consider workflow scheduling in a statically or dynamically manner [21]. Static scheduling heuristics like HEFT [22] or HCPPEFT [23] assign a set of tasks to compute resources before workflow execution. Dynamic approaches like P-HEFT [24] map tasks to infrastructure components at runtime or are able to adjust their scheduling plan dynamically. Therefore, these techniques are more flexible to changes in the actual execution, e.g., node failures. However, the presented state-of-the-art scheduling approaches have in common that they need extensive knowledge about the physical DAG, execution times on all machines, and communication times between dependent tasks. Accordingly, these approaches are often not feasible in real-world systems due to the comprehensive knowledge that is required.

In our approach, the employed and trained recommender system ranks machines for task instances on demand. These ranks can then be used by a more sophisticated scheduler to reduce the dependence from task runtimes, e.g., a scheduling approach that works with ranks and avoids the usage of error-prone task runtime estimates.

B. Task Runtime Prediction

Some approaches predict the runtime of tasks in a workflow [8], [14]–[17]. The papers employ several prediction models to find a reasonable runtime estimate. Therefore, they use regression, decision trees, neural networks, or other statistical approaches to create prediction models. As we use

[1] github.com/CRC-FONDA/WorkSim-PredError

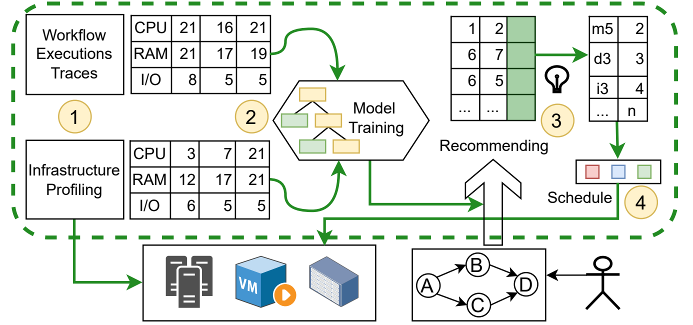

Fig. 1: Our approach in green with step ① to ④.

the runtime estimation approaches as a reference for state-of-the-art task runtime prediction errors, we will describe them in more detail in Section IV-B.

Our approach explicitly avoids predicting runtimes or relying on runtime predictions. Instead, our aim is to rank machines for a task, which can work as a substitute for the exact runtime. Therefore, different input parameters, like larger datasets, would influence the runtime but not necessarily the rank.

III. Reshi Approach

This section gives an overview of our proposed system. The numbering matches the circled numbers in Fig. 1.

In the ① **Cluster Profiling** step, we profile the available nodes in the cluster with a test suite that contains several benchmarks. Additionally, we add and align available data from historical workflow executions and the contained tasks to the data repository. If there are no existing workflow executions, an additional short workflow execution with highly reduced data inputs can be conducted to gather initial traces [8]. In the ② **Model Creation** step, we create a regression tree model based on the previously gathered infrastructure attributes like CPU speed, memory speed, or I/O. We enrich the model with task monitoring data from the historical workflow executions. This model is intended to rank and recommend nodes for a certain task in a heterogeneous cluster. The ranking is done in the ③ **Ranking and Recommendation** step. Thereby the model uses one submitted task as the input and then creates a ranking of fitting nodes. This ranking is used in the ④ **Scheduling** step, serving as a recommendation for the scheduler to place a task in the cluster.

① *Cluster Profiling*

We assume the compute cluster consists of several nodes with different kinds of hardware. Initially, we gather performance characteristics. For this, we use various benchmarks to measure CPU, memory, and disk I/O characteristics. Benchmarking the heterogeneous cluster can be done in parallel and has to be conducted only once for each node. Once failures or changes are detected in the cluster, the cluster resource manager, e.g. Kubernetes or Slurm, can rerun the profiling on the changed nodes. Instead of working with the absolute

measurement values, we rank the nodes for each benchmarked feature.

The task performance data originates from historical workflow executions or workflow profiling that uses the monitoring part of the SWMS. The data contains CPU, memory, disk I/O characteristics.

② *Model Creation*

The recommender system uses a subset C_{exc} of all possible task-node combinations C of task executions on the possibly heterogeneous infrastructure, i.e., $C_{exc} \subseteq C$. This input enables Reshi to learn an effective mapping and allow for recommendations of promising combinations. Based on user-defined conditions, e.g., a change in the cluster architecture, the recommender system can be scheduled for retraining.

In a next step, the data obtained from profiling the i-th cluster node and executing the j-th task is consolidated into an input vector $\vec{c}_{ij} \in \mathbb{R}^{v+w}$, where v denotes the number of task-related metrics and w the number of resource-related metrics respectively. We hereby obtain a vector for each executed combination entry $c_{ij} \in C_{exc}$. The gathered information is then transformed into a matrix $X^{n \times l}$, with $n = |C_{exc}|$ and $l = v + w$. We use this matrix as a training input for our regression tree model.

③ *Ranking and Recommendation*

Each node m_i in the set of nodes M has $q \in Q$ available resources, e.g. CPU, memory, I/O, GPU, denoted as $rm_i^{(q)}$. At the same time, each task t_j in the set of tasks T formulates requirements with respect to the different resources, denoted as $rt_j^{(q)}$. Therefore, our goal is to establish a ranking of nodes and to select the best fitting one for each given task.

First, we filter the set of nodes M to select all nodes that fulfill the task's resource requirements. We define the set of allocatable nodes as

$$M_{alloc} = \{m_i \in M | \forall q \in Q : rm_i^{(q)} \geq rt_j^{(q)}\},$$

i.e., in order to be considered as allocatable, the nodes need to have enough available resources to fulfill the resource request of the respective task t_j. Then, we pass the physical task t_j together with the set of allocatable nodes M_{alloc} to our regression tree model. The model then ranks the task t_j for each node $m_i \in M_{alloc}$ and creates a node priority list P in ascending order according to the ranking. The node with the lowest list index is the most recommended node. However, the priority list P can be used by the scheduling unit to, for example, optimize over a list of available tasks inside the queue and their ranks.

④ *Usage in Scheduling*

In the last step, a scheduler has to assign the tasks to the best fitting node. We propose two simple prioritizing approaches that use Reshi's recommendations.

The first task prioritizing strategy, Reshi-C, compares the number of children tasks and prefers tasks with many children. The second strategy, Reshi-M, orders all tasks by the average runtime from historical executions descending. Both strategies then allocate nodes to the tasks using the recommendations and the ordered queue.

These prioritizing strategies serve as simple examples as we intend to show how the ranking could be used. More sophisticated schedulers could substitute their dependence from runtimes per task-node pair through our provided ranking.

IV. Experiment Setup

In this section, we describe profiling benchmarks for heterogeneous clusters, our WorkflowSim [25] extension that can incorporate prediction errors, the evaluation workflows, and the baselines.

A. Prototype Implementation

This subsection shortly explains the infrastructure profiler and the recommender implementation.

a) Infrastructure Profiler: For profiling and benchmarking, we build on the Phoronix Test Suite[2]. The Phoronix Test Suite supports the installation, execution, and monitoring of a large variety of benchmarks.

To determine the CPU's maximum throughput of hashes per second, we use *John the Ripper* (JtR)[3]. Additionally, as a second metric of CPU performance, we use the time required to build the Linux kernel (BLK)[4]. While JtR is fully CPU-bound, BLK is largely CPU bound but can be impacted by the I/O for extremely low build times.

For memory performance measurements, we use *RAMspeed*[5] and *Stream*[6]. Both tools run four different operations, namely COPY, SCALE, ADD, and TRIAD. We combine both tools for higher accuracy in measuring the performance of the RAM.

Lastly, *fio* measures the data transfer rate and IOPS of the storage medium for sequential and random access. We chose block sizes of 4KB and 2MB for the random and sequential tests, respectively.

b) Recommender Implementation: For our recommender implementation, we use a regression tree model. The profiling values from the nodes, CPU, memory, and I/O metrics, serve as the first part of the input vector. The second part of the input vector contains the task traces, for example, the CPU usage, read/written bytes, peak memory, and average memory usage.

B. Workflow Simulation

Established simulation tools, like WorkflowSim [25] or WRENCH [26] assume accurate task runtime knowledge. However, task runtime predictions inherently yield a certain prediction error. We want to incorporate such a systematic prediction error into our simulations.

Further, they define a certain number of MIPS (millions of instructions per second) to a node in the cluster and

[2]phoronix-test-suite.com, Accessed: June 2022
[3]github.com/openwall/john, Accessed: June 2022
[4]kernel.org, Accessed: June 2022
[5]alasir.com/software/ramspeed, Accessed: June 2022
[6]cs.virginia.edu/stream, Accessed: June 2022

use this value to calculate the runtime depending to the node. However, this oversimplification neglects that tasks show different resource access patterns, e.g., particular applications run faster or slower on different CPUs architectures, while other tasks are solely I/O bound.

To overcome these limitations, we extend the popular WorkflowSim simulation environment in our simulation setup.

First, to incorporate the systematic task runtime prediction error, we evaluated the papers from Section II in order to derive realistic prediction errors. Nadeem et al. [15] report a normalized average absolute prediction error of 10%, 11%, and 15% for three different workflows. For the tasks in two workflows, the error is normally distributed, while the third workflow yields that the majority of tasks show higher error rates. Hilman [17] report the prediction errors for all tasks in a single workflow. They show that their technique is able to provide a task runtime estimation error below 5% for two tasks but also errors above 30% for three other workflow tasks. They are able to outperform the Two-stages baseline [16], where the authors report a slightly higher relative absolute error. In our own previous work, Lotaru [8], we achieved a median prediction error between 14% and 22% for heterogeneous cluster infrastructures, while the prediction error for the best performing baseline yielded a median error of 31%. Further, our results showed that the prediction errors of tasks over a workflow are frequently distributed according to a long-tailed exponential distribution.

Accordingly to these observations, we introduce a prediction error noise to the scheduler input in our WorkflowSim fork. Therefore, the predicted runtime r_p is defined as $r_p = r * (1 \pm \mathcal{N}(1, 0.5) * err)$ for a normal distribution and as r_p as $r_p = r * (1 \pm Exp(1) * err)$ for an exponential distribution where r is the true runtime and err the prediction error.

The respective prediction error is sampled from either a normal or an exponential distribution. However, the actual task runtimes remain unchanged and are not influenced by the error.

Second, instead of relying on the runtime extrapolation by simply using the MIPS, our WorkflowSim fork looks up the real runtimes of a given task on a certain machine. For our setup, we used 27 real instance types from AWS EC2, using the sizes *large*, *xlarge*, and *2xlarge*, where applicable. We run all evaluation workflows on these machines to gather detailed task runtimes.

Out of these 27 machines, we created 200 random heterogeneous clusters comprising of 40 nodes each. All approaches run once on each of the 200 clusters.

C. Evaluation Workflows

We selected three publicly available real-world Nextflow [3] workflows from the popular nf-core [1] repository: *Viralrecon*[7] – variant calling for viral samples, *Eager* – ancient DNA analysis [27], and *Chipseq*[8] – peak-calling. The workflows have different resource usage patterns, and different Directed-Acyclic-Graph (DAG) structures.

[7]github.com/nf-core/viralrecon, Accessed: June 2022

[8]github.com/nf-core/chipseq, Accessed: June 2022

TABLE I: Workflow runtimes with different schedulers assuming a normally distributed runtime prediction error of 15%.

		Mean %	**90p %**	**95p %**	**Max %**
Chipseq	HEFT	5.58	59.43	70.79	129.62
	Reshi-C	**0.00**	**7.53**	**7.58**	**27.97**
	Reshi-M	14.32	7.85	35.39	45.35
	MinMin	8.71	22.44	28.70	42.76
	RR	69.41	116.35	119.21	122.61
Eager	HEFT	10.47	35.47	39.53	54.07
	Reshi-C	14.53	27.33	29.07	30.81
	Reshi-M	**0.00**	**6.40**	**6.40**	**16.28**
	MinMin	12.21	29.07	33.72	56.40
	RR	52.33	99.42	119.19	119.77
viralrecon	HEFT	25.25	82.47	115.77	104.02
	Reshi-C	10.72	23.24	37.60	38.48
	Reshi-M	**0.00**	**15.30**	**18.55**	**29.08**
	MinMin	18.88	37.73	41.54	45.84
	RR	44.71	63.73	65.24	80.90

To feed our WorkflowSim extension with actual task-machine runtimes, we collect real trace information from real executions, which then can be extrapolated. We run each workflow five times on the 27 instance types. Further, we extended Nextflow in a way that it stores the traces in a WorkflowSim readable file. The runtime is then extrapolated to simulate long-running workflows.

D. Baselines

We compare Reshi with three baselines, namely, Round-Robin, MinMin, and HEFT (Heterogeneous Earliest Finish Time).

The first baseline, Round-Robin, is a frequently used scheduling technique by resource managers, e.g., Kubernetes uses a round-robin like approach [18]. MinMin is a popular dynamic job scheduling algorithm that orders the queue to be scheduled ascending by the task runtime and then selects the fastest machine [28]. HEFT [22] is a static list scheduling algorithm that incorporates different task-machine runtimes, communication times, and the structure of the directed acyclic graph (DAG). Except for Round-Robin, all baselines require a-priori knowledge about task-machine runtimes.

Since Reshi itself recommends machines and does not aim to schedule the tasks, e.g., prioritize the tasks by runtime or number of descendants, we show Reshi in combination with two simplified task prioritizing techniques. Reshi-C compares the number of children tasks and prefers tasks with many children. The second strategy, Reshi-M, orders all tasks by the average runtime from historical executions descending.

V. Evaluation Results

We evaluate two different use-cases in our experiments using the same 200 clusters and three workflow setups each. For both use-cases, we assume that the task runtime is predicted from historical execution traces [14]–[17]. In the first use-case, we assume a prediction error that is normally distributed. The second use-case incorporates an exponentially distributed prediction error. We test both scenarios with different mean

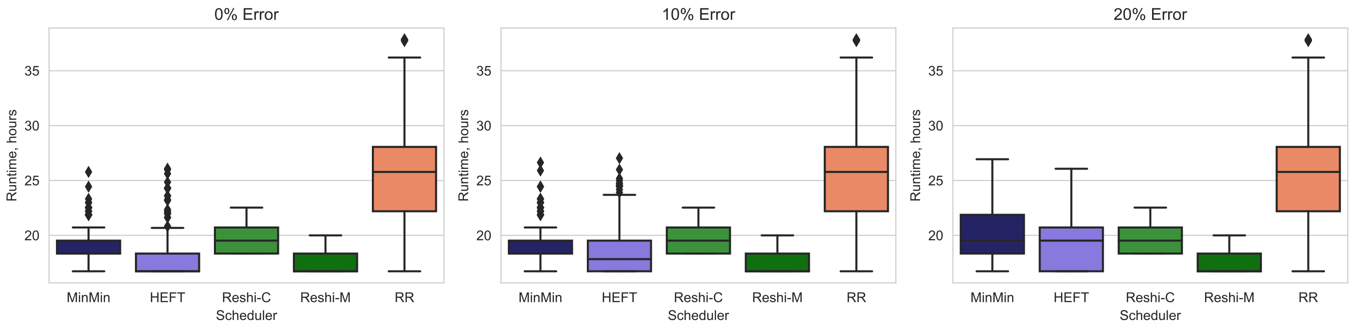

Fig. 2: Makespans of the Eager workflow for different schedulers with normally distributed runtime prediction errors.

TABLE II: Workflow runtimes with different schedulers assuming an exponentially distributed runtime prediction error of 15%.

		Mean %	90p %	95p %	Max %
Chipseq	HEFT	21.86	76.56	92.49	229.37
	Reshi-C	**0.00**	**4.98**	**5.03**	**24.94**
	Reshi-M	11.61	31.63	32.18	41.90
	MinMin	15.41	41.40	51.86	115.54
	RR	65.39	111.22	114.01	117.33
Eager	HEFT	23.84	54.65	87.79	119.77
	Reshi-C	14.53	27.33	29.07	30.81
	Reshi-M	**0.00**	**6.40**	**6.40**	**16.28**
	MinMin	29.07	83.72	99.42	119.77
	RR	52.33	99.42	119.19	119.77
viralrecon	HEFT	35.83	73.37	86.69	198.87
	Reshi-C	10.53	23.24	37.60	38.48
	Reshi-M	**0.00**	**15.30**	**18.55**	**29.08**
	MinMin	28.17	57.91	62.18	77.78
	RR	44.71	63.73	65.24	80.90

error rates that are based on reported research results, as elaborated in Section IV-B.

Due to the high number of evaluation setups, the following sections report detailed results for the Eager workflow with different error assumptions and results for all workflows with a systematic over- or under-prediction of 15%.

A. Normally Distributed Error

In our first scenario, we compare Reshi's resource allocation with Round-Robin and two state-of-the-art schedulers assuming a normally distributed prediction error. Figure 2 shows the makespan of the Eager workflow with different normally distributed runtime prediction errors. Reshi-M and HEFT achieve the same median makespans assuming a totally accurate task runtime, i.e., no prediction error. However, once the error increases, all schedulers except for Reshi-C, Reshi-M, and Round-Robin lead to higher makespans. With an error of 15%, HEFT yields a 75th percentile that has a 9.81% higher makespan compared to the 75th percentile of Reshi-M. Reshi-M's percentile is always below the respective 75th percentile of the competitors. For all the baseline approaches, except for Round-Robin, an increased error leads to longer workflow runtimes and more cases where outliers, i.e., workflow executions with significantly longer makespans, can be detected.

Table I summarizes all workflow makespans assuming a normally distributed task prediction error of 15%. For each workflow, we set the lowest mean value to 0 and depict the relative change according to that value. One can see that two out of three times, Reshi-M achieves the lowest workflow runtimes and for the other workflow Reshi-C. For Eager and viralrecon, Reshi-M achieves the best results. This is, for example, due to the Eager workflow structure, where the maximum depth is two, and, therefore, strategies that prefer long graph dependencies, e.g., Reshi-C, yield to a higher makespan. Reshi's 95th percentile is below the competitors' 90th percentile in all workflows. Further, HEFT's maximum values are frequently more than two times higher compared to Reshi's maximum reported value. For the Chipseq workflow, the maximum value is more than 4.5 times higher

B. Exponentially Distributed Error

In our second experiment we assume an exponentially distributed task runtime prediction error. Again, Figure 3 shows the Eager workflow makespan for the different schedulers assuming various prediction error levels. The maximum reported makespan of Reshi-M and Reshi-C is below the 75th percentile of HEFT. Table II shows that Reshi's maximum value is always below the 90th percentile of the baseline approaches. Further, the mean and the percentiles differences are much more considerable now. Compared to the experiment with a normally distributed error from the section before, the baseline schedulers react stronger to an increase in the error rate, assuming an exponential error distribution. Here, a higher error rate necessarily leads to higher makespans for the baselines, except for Round-Robin. Since Reshi-C and Reshi-M do not rely on predicted runtimes or estimated resource usages, they constantly achieve the same results.

VI. Conclusion

This paper presented an approach to dynamically map scientific workflow tasks onto heterogeneous infrastructures using our recommender systems. Through the task-node ranks, Reshi does not rely on error-prone runtime estimates and has not to cope with data-dependent resource predictions.

Our experimental evaluation with three real-world Nextflow workflows from the popular nf-core framework shows that

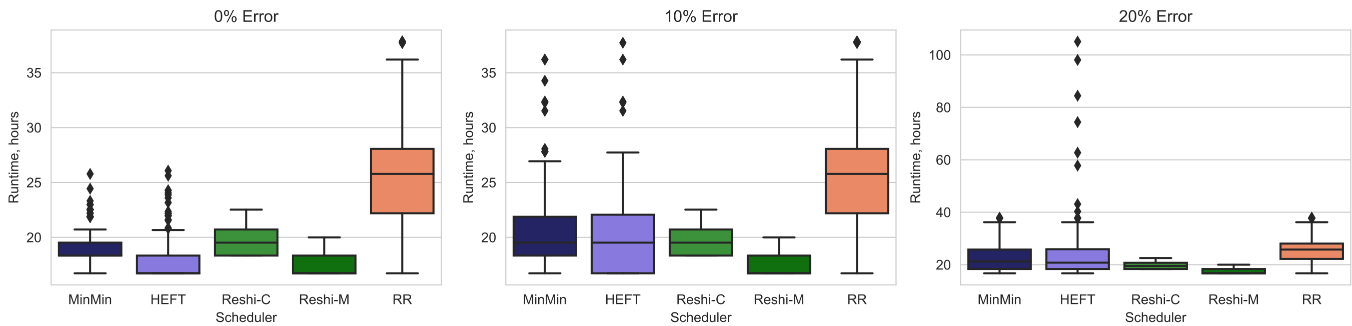

Fig. 3: Makespans of the Eager workflow for different schedulers with exponentially distributed runtime prediction errors.

Reshi provides efficient task-machine allocations without requiring accurate task runtime estimates, while we show that in comparison the performance of state-of-the-art schedulers is highly dependent on accurate task-runtime predictions.

Pairing Reshi's recommendations with a simple scheduler, Reshi is able to outperform HEFT by a mean makespan reduction of 7.18% for a normally distributed error of 15% and a mean makespan reduction of 18.01% for an exponentially distributed error of 15%. Further, the baseline schedulers yield significantly higher 90th percentile quarterlies and a significantly higher variance.

Acknowledgments

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

References

[1] 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.

[2] R. F. da Silva, H. Casanova, A.-C. Orgerie, R. Tanaka, E. Deelman, and F. Suter, "Characterizing, modeling, and accurately simulating power and energy consumption of i/o-intensive scientific workflows," *Journal of computational science*, vol. 44, p. 101157, 2020.

[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, 2017.

[4] E. Deelman, K. Vahi, M. Rynge, R. Mayani, R. F. da Silva, G. Papadimitriou, and M. Livny, "The evolution of the pegasus workflow management software," *Computing in Science & Engineering*, vol. 21, no. 4, pp. 22–36, 2019.

[5] S. Mohammadi, L. PourKarimi, and H. Pedram, "Integer linear programming-based multi-objective scheduling for scientific workflows in multi-cloud environments," *The Journal of Supercomputing*, 2019.

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

[7] F. Lehmann, D. Frantz, S. Becker, U. Leser, and P. Hostert, "Force on nextflow: Scalable analysis of earth observation data on commodity clusters," in *CIKM Workshop*, 2021.

[8] J. Bader, F. Lehmann, L. Thamsen, J. Will, U. Leser, and O. Kao, "Lotaru: Locally estimating runtimes of scientific workflow tasks in heterogeneous clusters," in *SSDBM 2022*. ACM, 2022.

[9] B. Burns, B. Grant, D. Oppenheimer, E. Brewer, and J. Wilkes, "Borg, omega, and kubernetes," *Queue*, vol. 14, no. 1, 2016.

[10] 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.

[11] K. Hutson, D. Andresen, A. Tygart, and D. Turner, "Managing a heterogeneous cluster," in *PEARC*, 2019.

[12] J. Will, L. Thamsen, D. Scheinert, J. Bader, and O. Kao, "C3O: Collaborative Cluster Configuration Optimization for Distributed Data Processing in Public Clouds," in *IEEE IC2E*. IEEE, 2021, p. to appear.

[13] S. Kulagina, H. Meyerhenke, and A. Benoit, "Mapping Tree-shaped Workflows on Memory-heterogeneous Architectures," in *HeteroPar 2022*. Springer, 2022.

[14] 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, p. 1541003, 2015.

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

[16] T.-P. Pham, J. J. Durillo, and T. Fahringer, "Predicting workflow task execution time in the cloud using a two-stage machine learning approach," *IEEE Transactions on Cloud Computing*, 2017.

[17] M. H. Hilman, M. A. Rodriguez, and R. Buyya, "Task runtime prediction in scientific workflows using an online incremental learning approach," in *2018 IEEE/ACM 11th International Conference on Utility and Cloud Computing (UCC)*. IEEE, 2018, pp. 93–102.

[18] C. Carrión, "Kubernetes scheduling: Taxonomy, ongoing issues and challenges," *ACM Computing Surveys (CSUR)*, 2022.

[19] S. Tang, B.-S. Lee, and B. He, "Fair resource allocation for data-intensive computing in the cloud," *IEEE Transactions on Services Computing*, vol. 11, no. 1, pp. 20–33, 2016.

[20] C. Delimitrou and C. Kozyrakis, "Quasar: resource-efficient and qos-aware cluster management," *ACM SIGPLAN Notices*, 2014.

[21] T. L. Casavant and J. G. Kuhl, "A taxonomy of scheduling in general-purpose distributed computing systems," *IEEE Transactions on Software Engineering*, vol. 14, no. 2, 1988.

[22] H. Topcuoglu, S. Hariri, and M.-y. Wu, "Performance-effective and low-complexity task scheduling for heterogeneous computing," *IEEE transactions on parallel and distributed systems*, vol. 13, no. 3, 2002.

[23] Y. Dai and X. Zhang, "A synthesized heuristic task scheduling algorithm," *The Scientific World Journal*, vol. 2014, 2014.

[24] J. G. Barbosa and B. Moreira, "Dynamic scheduling of a batch of parallel task jobs on heterogeneous clusters," *Parallel computing*, 2011.

[25] W. Chen and E. Deelman, "Workflowsim: A toolkit for simulating scientific workflows in distributed environments," in *2012 IEEE 8th international conference on E-science*. IEEE, 2012, pp. 1–8.

[26] H. Casanova, S. Pandey, J. Oeth, R. Tanaka, F. Suter, and R. F. Da Silva, "Wrench: A framework for simulating workflow management systems," in *WORKS*. IEEE, 2018.

[27] A. Peltzer, G. Jäger, A. Herbig, A. Seitz, C. Kniep, J. Krause, and K. Nieselt, "Eager: efficient ancient genome reconstruction," *Genome biology*, vol. 17, no. 1, pp. 1–14, 2016.

[28] N. Sharma, S. Tyagi, and S. Atri, "A comparative analysis of min-min and max-min algorithms based on the makespan parameter." *International Journal of Advanced Research in Computer Science*, 2017.