Not All Tasks Are Created Equal: Adaptive Resource Allocation for Heterogeneous Tasks in Dynamic Workflows

Thanh Son Phung[1], Logan Ward[2], Kyle Chard[3], and Douglas Thain[1]

[1]Department of Computer Science and Engineering, University of Notre Dame
[2]Data Science and Learning Division, Argonne National Laboratory
[3]Department of Computer Science, University of Chicago

***Abstract*—Users running dynamic workflows in distributed systems usually have inadequate expertise to correctly size the allocation of resources (cores, memory, disk) to each task due to the difficulty in uncovering the obscure yet important correlation between tasks and their resource consumption. Thus, users typically pay little attention to this problem of allocation sizing and either simply apply an error-prone upper bound of resource allocation to all tasks, or delegate this responsibility to underlying distributed systems, resulting in substantial waste from allocated yet unused resources. In this paper, we will first show that tasks performing different work may have significantly different resource consumption. We will then show that exploiting the heterogeneity of tasks is a desirable way to reveal and predict the relationship between tasks and their resource consumption, reduce waste from resource misallocation, increase tasks' consumption efficiency, and incentivize users' cooperation. We have developed two info-aware allocation strategies capitalizing on this characteristic and will show their effectiveness through simulations on two modern applications with dynamic workflows and five synthetic datasets of resource consumption. Our results show that info-aware strategies can cut down up to 98.7% of the total waste incurred by a best-effort strategy, and increase the efficiency in resource consumption of each task on average anywhere up to 93.9%.**

I. Introduction

Complex workflows consisting of a large number of parallel tasks run on clusters are central to computational science today. Workflow management systems are used widely in fields as diverse as bioinformatics, high energy physics, molecular dynamics, machine learning, and more [1] [5] [8]. An emerging class of systems is dynamic workflows, in which a high-level application generates task definitions at runtime and passes them to an underlying execution system. A dynamic workflow, by definition, does not state in advance the number and types of tasks to be encountered; these only become apparent at runtime as the application evolves.

A significant problem in the design of dynamic workflows and workflow systems is resource allocation: for each task to be executed, what quantity of resources (e.g. cores, memory, disk) should be allocated to that task? The sizes of these allocations are important for the end user: allocate too little, and a task fails; allocate too much, and resources are wasted. The allocation sizes are also important to the underlying scheduling systems: a small allocation is easy to place into a busy cluster, while a large allocation may require preempting other jobs, or simply waiting until a large space becomes free.

Further, whose job it is to select appropriate resource allocations? It is easier to design a system in which the user *tells* the system what resources to allocate, but the end user may not have the time or expertise to make accurate and confident measurements or predictions [6] [10]. The underlying workflow system or batch system is better positioned to measure tasks as they run, but has little insight into the high-level goals and needs of the user.

We believe that the key to efficient resource allocation lies in a smooth *cooperation* between users and distributed systems by incentivizing users to disclose sufficient information about workflows for the distributed systems to make good automated decisions. Consequently, an intuitive question we pose and attempt to answer is: what type of information would both maximize the performance of resource allocation to tasks and minimize users' potential costs or efforts in acquiring it?

In this paper, we will formalize the problem of resource allocation to tasks in workflows, and show that utilizing information about workflows can help guide the process of resource allocation and reduce its potential waste. While different types of information have different sets of assumptions, degree of practicality, and effectiveness in decreasing waste of resources, we will show that exploiting information about the heterogeneity of tasks in workflows is desirable due to the following reasons:

1) **Observation**: Tasks are not necessarily of the same type and might perform different work. This may create a considerable gap in resource consumption between different types of tasks and present an opportunity to optimize the process of resource allocation.
2) **Opportunity**: Users are likely to have some knowledge of the heterogeneous nature of tasks, such as the number of task types or even the specific type of each task in a workflow. Consequently, users can have the freedom and opportunity to cooperate and provide different amounts of information about the heterogeneity of tasks in workflows to resource allocation schemes

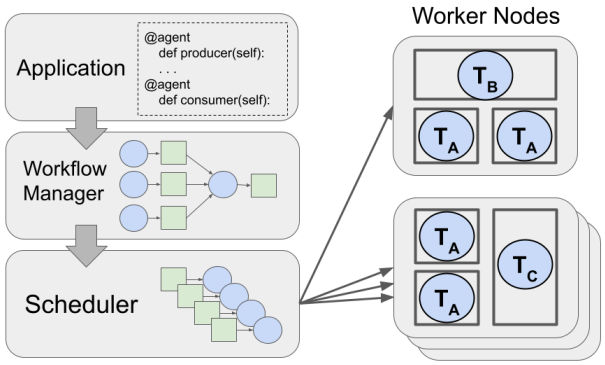
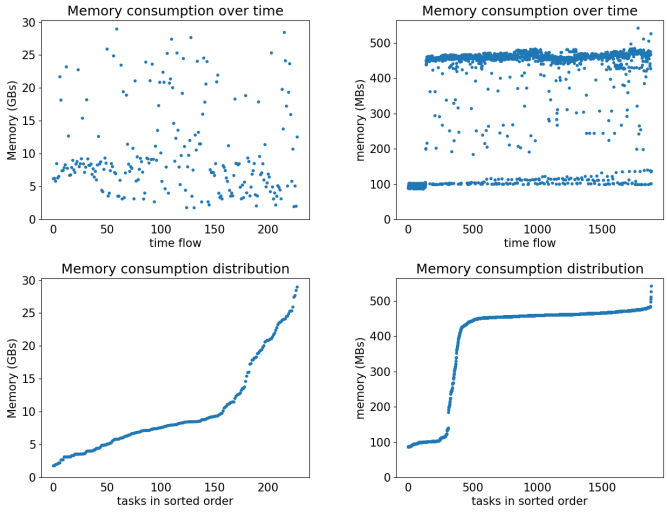


Fig. 1: Architecture of Dynamic Workflows.
A top level application (e.g. Colmena) generates tasks for a workflow manager (e.g. Parsl), which sends ready tasks to a scheduler (e.g. Work Queue), which dispatches tasks to workers, running in individual allocations.



(a) Colmena (b) TopEFT

Fig. 2: Visualization of memory consumption of tasks in Colmena-XTB and TopEFT.
Each point is a task's peak consumption. (Top) Chronological memory consumption ordered by tasks' finish time. (Bottom) Distributions of memory consumption of tasks.

depending on users' willingness and knowledge without much expertise and effort in advance.

3) **Solution**: We have developed info-aware allocation strategies that are not only able to predict tasks' resource consumption at run time given users' knowledge of the heterogeneity of workflows, but also independent from yet integrable to both applications and distributed systems' resource allocation schemes.
4) **Performance**: Our strategies show considerable savings on resources and boost in tasks' efficiency in consuming allocated resources as to be described below.

We evaluate the performance of info-aware allocation strategies through simulations on two real datasets and five synthetic datasets of resource consumption. The two real datasets are collected from resource consumption reports from actual executions of two applications with dynamic workflows: Colmena-XTB [3], with molecular simulation and inference tasks, and TopEFT [7], with physics event processing and analysis tasks. As the distributions of resource consumption of these two datasets might not reflect a wide range of possible distributions, we additionally generate five synthetic datasets following five different distributions: uniform, normal, exponential, bimodal, and trimodal. We further quantify the performances of our strategies when provided with different levels of information about the heterogeneity of workflows, from the scenario where no information is given, to the scenario where users specify the associated type of task for every task in a workflow.

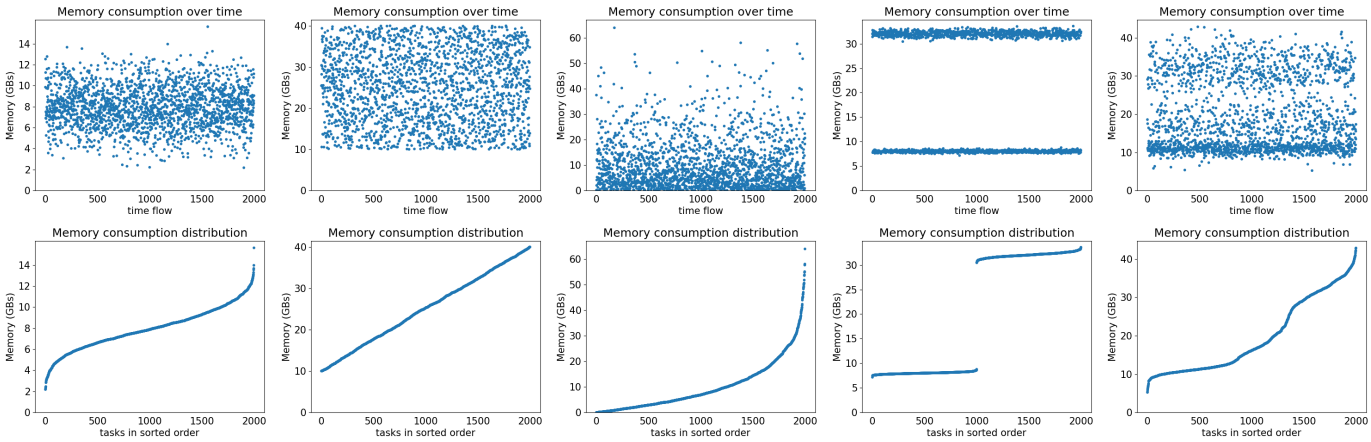
Our results show that, depending on the amount of information provided, info-aware allocation strategies can reduce waste anywhere from 2.4% to 98.7% compared to the total waste incurred by a best-effort strategy. Moreover, while the oracle-like allocation strategy can perfectly predict every task's consumption and achieve 100% average task efficiency, our allocation strategies can reach anywhere from 16.1% to 93.9%.

II. Applications and Datasets

A. Applications

Figure 1 shows the general architecture of applications considered in this paper. At the top level, an application-specific layer written in Python generates tasks in the form of calls to conventional Python functions that perform simulation, modeling, data analysis, etc. The function calls and their arguments are passed to the workflow manager at the middle layer that arranges tasks in a graph, and determines when each task is ready to run based on the availability of its inputs. Ready tasks are passed to the bottom layer, which schedules independent tasks to available worker nodes. Worker nodes then statically allocate their available resources to tasks, enforce the allocations, and record tasks' resource consumption.

Colmena-XTB is one example of an application designed in this style. This application combines machine learning techniques with traditional molecular dynamics in order to perform efficient steering of molecular design. It is built using Colmena [3], a Python library allowing users to build AI-driven applications that steer large ensembles of simulation tasks. Colmena's design revolves around a *Thinker* application, which includes two main agents: producers and consumers. In this setting, producers create both simulation tasks to perform and inference tasks to choose which simulations to perform next, while consumers retrieve and record results from completed tasks. Colmena hands off individual tasks to Parsl [5], a workflow system that enables scalable parallel executions of Python functions and programs. Parsl translates function invocations into a dynamic graph of tasks, relying



(a) Normal
μ: 8, σ: 2
types of tasks: 1

(b) Uniform
a: 10, b: 40
types of tasks: 1

(c) Exponential
λ: 10
types of tasks: 1

(d) Bimodal with 2 normal distributions
μ_1: 32, σ_1: 0.5
μ_2: 8, σ_2: 0.2
types of tasks: 2

(e) Trimodal with 3 normal distributions
μ_1: 32, σ_1: 4
μ_2: 16, σ_2: 4
μ_3: 11, σ_3: 1
types of tasks: 3

Fig. 3: Memory consumption of tasks in five synthetic datasets.
Each point is a task's peak consumption. (Top) Chronological memory consumption ordered by tasks' finish time. (Bottom) Distributions of memory consumption of tasks.

on futures to represent pending data items, and sends ready tasks to Work Queue [8] on distributed worker nodes. Work Queue is a master-worker distributed programming framework that implements dynamic and fine-grained task generation and deployment, results collection, and resource management of workflows over a variety of distributed systems.

TopEFT is another example of an application written in this style. TopEFT is designed to search for new physics that affects top-quark production by making use of the effective field theory (EFT). It operates by consuming a large number of events produced by the Large Hadron Collider (LHC) containing signatures that arise from the production of top quarks in association with W, Z, or Higgs particles. The top level of the software defines two functions: one for processing individual events, and another for accumulating results from multiple events into a complex multi-level histogram. These functions are given to Coffea [2], a general-purpose data processing framework for high energy physics. Coffea logically partitions the input datasets and generates tasks, each one invoking a function on a portion of the dataset or a partially accumulated result. Ready tasks are given to Work Queue for scheduling on worker nodes, as above.

B. Datasets

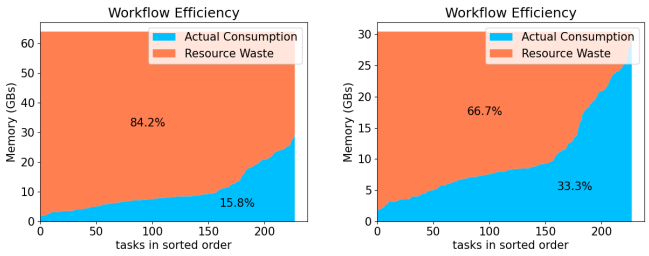
We collected resource consumption data from tasks in Colmena-XTB and TopEFT workflows using the Resource Monitor [4] tool from the CCTools [16] package, which monitors tasks' resource consumption by tracking a hierarchy of processes spawned by individual tasks and observing snapshots of such processes' resource consumption taken from the proc filesystem. Colmena-XTB generates 228 tasks of two types (simulation and inference), and TopEFT generates 1883 tasks of three types (preprocessing, processing, and accumulating).

Both Colmena-XTB and TopEFT exhibit substantial variations in task memory consumption, as shown in Figure 2 (we skip discussions on cores and disk consumption for convenience.) Colmena-XTB shows variations between two fundamentally different task types and also within each type, as the simulation and model parameters have a direct effect on the size of data structures. TopEFT also shows substantial variations between event processing and results accumulating tasks (The first 136 tasks partition data.) We further see the variations in event processing tasks where some data partitions contain fewer events of interest than others. Hence, the memory consumption of each task also varies with the heterogeneity of the input data.

To simulate how other heterogeneous applications might behave, we generated five additional datasets of resource consumption of tasks with different levels of heterogeneity and distributions by random sampling from respective distributions, each with 2,000 data points, as shown in Figure 3. We only vary memory consumption of tasks in these synthetic datasets, for simplicity.

III. Information-Performance Tradeoffs

Figure 4a shows how serious memory waste can be in a possible scenario where users running the Colmena-XTB application delegate the responsibility of allocating resources to tasks to underlying distributed systems. In this scenario, we assume that the underlying systems simply allocate a machine with 64 GBs of memory to each task (we will skip the



(a) Whole machine allocation (b) Over-allocation with 5% error

Fig. 4: Memory Consumption Efficiency of Tasks in Colmena-XTB

(Left) Users allocate each task with a whole machine of 64 GBs of memory. (Right) Users give each task an allocation of 5% larger than the maximum consumption of any task.

discussion on cores and disk and assume that tasks have equal execution time for convenience.) With this default strategy, tasks in the Colmena-XTB workflow only consume 15.8% of the total allocated memory in their execution process.

If we relax the above scenario and assume that users have some information about workflows such that they can derive a near perfect upper bound for all tasks with only 5% misprediction error, we still see that tasks only consume about one-third of their total allocated memory due to the exponential-like distribution of memory consumption, as shown in Figure 4b. Moreover, despite gaining a more than two-fold improvement in memory consumption efficiency, users are very unlikely to precisely and confidently derive such prediction by guesswork alone, and thus must pay extra costs or efforts in acquiring information about the workflow. For example, users can approximate the resource consumption of tasks with high confidence and accuracy by using information on the resource consumption of a workflow similar to Colmena-XTB. However, this too implies the undesirable cost of executing such similar workflow.

We believe that the key to alleviating waste from resource misallocation to tasks lies in the users' knowledge about the nature of workflows and willingness to cooperate with distributed systems by providing such information. That is, the more workflow-specific information users are able to provide to allocation schemes, the better the performance these allocation schemes can achieve. However, as stated above, acquiring information about workflows may be costly as such information might not be readily available.

Exploiting the heterogeneity of tasks in workflows is a candidate solution to this problem. This is because users are likely to have some knowledge about tasks in workflows in advance through either actual development of workflows or expert understanding of how tasks in workflows operate, and thus don't have to spend extra efforts acquiring such information. As we will show for the rest of this paper, exploiting the heterogeneity of tasks in workflows is even more desirable as it substantially decreases the waste due to misallocation, increases tasks' efficiency in resource consumption,

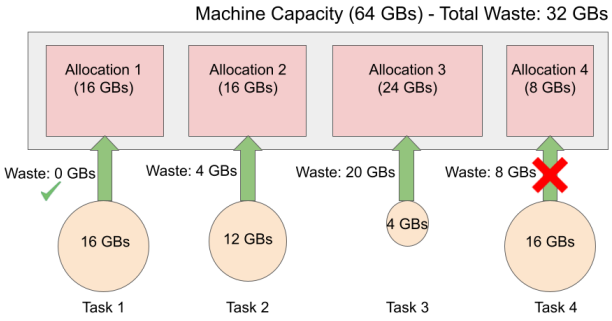


Fig. 5: The Resource Allocation Problem

A machine's memory can be divided into different allocations, each of which limits a task's memory consumption. Under-allocating resources results in waste of unusable resources due to terminated tasks, while over-allocating results in waste of unused resources.

and seamlessly integrates such information into our developed allocation strategies as described in Section V.

IV. Resource Allocation Problem

Now we provide a formal statement of the problem of resource allocation. We will hold the following assumptions throughout this section:

1) Resource allocations to tasks are static: they are determined in advance and don't change over the course of tasks' executions.
2) A task's execution is terminated immediately if it exceeds its allocation, and is either scheduled to rerun with a larger allocation, or returned to users as a failure.

The problem of allocating resources of a cluster to tasks is then comprised of two main entities: the tasks' peak resource consumption, and their allocations on machines. We will define a task to be an executable program that, when executed, consumes at most c cores, m MBs of memory, and d MBs of disk in t consecutive seconds. Thus, a task is characterized by a 4-tuple (c, m, d, t). A task T is allocated on some machine M if T is ready to execute on M and M is ready to commit c_a cores, m_a MBs of memory, and d_a MBs of disk in t_a seconds to serving T. Then, a successful allocation of T on M happens when all of the following conditions are true:

1) T runs on M in t seconds, and $t \leq t_a$.
2) There's no interruption, eviction, etc. in T's execution.
3) T's peak resource consumption does not exceed its allocation in M, and its allocation in M does not exceed the maximum capacity of M.

The last condition can be represented by these inequalities:

1) $c_M \geq c_a \geq c$
2) $m_M \geq m_a \geq m$
3) $d_M \geq d_a \geq d$

where c_M, m_M, d_M are the maximum capacity of machine M in cores, memory, and disk, respectively. Figure 5 visualizes this allocation problem with memory as an example.

In the optimal case, we will have that, for each task, $c = c_a$, $m = m_a$, $d = d_a$, and $t = t_a$. That is, a task's peak consumption is exactly equal to its allocation. If we are to achieve these conditions, then no further actions can be taken to increase the performance of a workflow. This is because, in this state, the waste due to mis-allocation is minimized, and tasks' efficiency in resource consumption is maximized. Thus, the goal of this problem naturally is to minimize, for each task, $c_a - c$, $m_a - m$, $d_a - d$, and $t_a - t$. That is, we want to allocate the amount of resources as close as possible to a given task's peak resource consumption. While we can easily minimize $t_a - t$ by terminating a task's allocation as soon as it finishes its execution, the problem of choosing c_a, m_a, and d_a such that $c_a - c$, $m_a - m$, $d_a - d$ are minimized is nontrivial and important in saving unnecessary waste of resources and boosting tasks' consumption efficiency, as demonstrated in Figure 4.

V. Strategies

In this section, we will present five resource allocation strategies of two types: info-oblivious and info-aware. Info-oblivious strategies will allocate resources to tasks without requiring and using any information prior to or during the execution of workflows. On the other hand, info-aware strategies will allocate resources to tasks based on users' guesswork or provided information on the heterogeneity of tasks in workflows and resource consumption reports of completed tasks. All strategies will hold the following assumptions:

- All machines in a given distributed system have the same configuration in the amount of cores, memory, and disk.
- No task will consume more resources than a machine's capacity.
- There are no interruptions during a task's execution.

A. Info-Oblivious Strategies

1) Whole machine: In this strategy, each task is allocated with a whole machine during its execution. This strategy is straightforward to implement as we will allocate a free machine to each task, and tasks will execute successfully in the first try by the assumptions above. However, tasks are likely to consume much less than the capacity of a machine, as demonstrated in Figure 4. Thus, a large portion of resources of a machine will be unused for the entire duration of a task's execution. Instead of allocating a machine per task, there are good chances that several tasks can be packed into a single machine to avoid unnecessary waste of resources and speed up workflows.

2) Double allocation: This strategy doubles the allocation of a task whenever it fails by over-consuming its current allocation. For example, if a task is allocated with $1/8^{\text{th}}$ of a machine and fails, then such task is re-allocated with $1/4^{\text{th}}$ of a machine and so on before finally being forced to run on a whole machine. Under the assumption that a machine can serve multiple tasks at a time, then this strategy is an improvement over the whole machine strategy since tasks will be allocated at most $1/8 + 1/4 + 1/2 = 7/8^{\text{th}}$ of a machine before finally being forced to run on a whole machine. However, this strategy requires the initial value of allocation as a hyper-parameter that must be configured prior to workflows' executions.

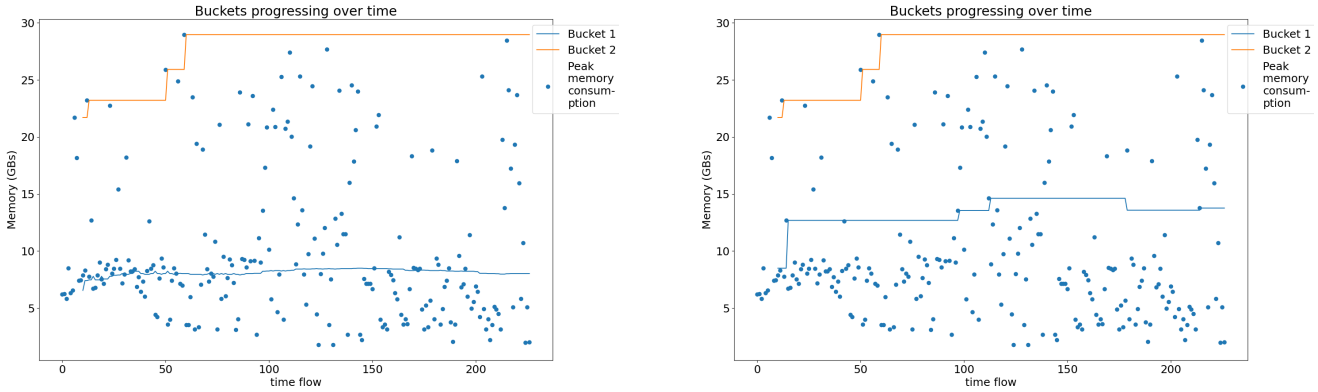
B. Info-Aware Strategies

1) User declaration: In this strategy, users allocate the same amount of resources for each task in a workflow. Any task failing due to over-consumption is retried using a whole machine. While this strategy incorporates users' prior knowledge about the workflow, users' predictions of tasks' resource consumption are prone to errors, costly to acquire, and thereby potentially increase the waste due to resource mis-allocation as tasks that fail by incorrect guess must be executed on a whole machine.

2) Quantized bucketing: Figure 6a demonstrates how quantized bucketing works on the Colmena-XTB dataset. For each type of resources (cores, memory, disk), this strategy aims to find a sequence of increasing amount of resources to allocate a new task given a list of resource consumption of completed tasks. A task is initially allocated with the smallest amount of resources in the sequence. Whenever a task fails due to under-allocation of any types of resources, it is allocated with the next amount of resources in the sequence, for all types of resources, until it has to be allocated with a whole machine. This strategy acquires the list of resource consumption of completed tasks by first executing a small number of tasks, with each task being allocated following the whole machine strategy. To find the sequence of increasing amount of resources, quantized bucketing first sorts the list of resource consumption of completed tasks in increasing order and divides it evenly into some pre-defined number of buckets, and then forms the sequence based on these buckets. Specifically, if we have n buckets, each of which has a lower and upper delimiter, then the i^{th} bucket's lower delimiter is the $(i-1)^{\text{th}}$ bucket's upper delimiter, and the i^{th} bucket's upper delimiter is whatever value of resource consumption sitting at the $(i/n)^{\text{th}}$ percentile. The sequence is then formed by collecting all buckets' upper delimiters in increasing order. Picking the right number of buckets n is a nontrivial sub-problem, but in this paper, we simply set it to be the number of types of tasks. During the execution of a workflow, the list of completed tasks is updated whenever a task finishes its execution and reports its peak resource consumption.

In this strategy, a new task's resource consumption is predicted using the distribution of completed tasks' resource consumption. Under the assumption that the prior distribution is similar to the distribution of the entire workflow, then each task is expected to fall into one of n buckets with an equal probability of $1/n$. Thus, on average, each task is tried $\frac{1}{n} * (\frac{n*(n+1)}{2}) = \frac{n+1}{2}$ times.

3) K-means bucketing: Figure 6b shows how k-means bucketing works on the Colmena-XTB dataset. Similar to the quantized bucketing strategy, the k-means bucketing strategy also aims to find a sequence of increasing amount of resources to allocate a new task based on a list of completed tasks' resource consumption, but it tries to divide the list based



(a) Quantized bucketing

(b) K-means bucketing

Fig. 6: Bucketing strategies with 2 buckets on the Colmena-XTB dataset (memory only).
Each point represents a task's peak memory consumption. Each line represents a bucket's upper delimiter. Tasks are given incremental allocations as shown by lines until they successfully finish their executions.

on the potential clusters of tasks rather than evenly. This strategy also obtains an initial list of completed resource consumption by executing a small number of tasks, and divides the list into n even buckets, where n is a hyper-parameter. However, in each iteration after obtaining the initial list of tasks' consumption, this strategy computes the mean of all values in each bucket, and reassigns all completed tasks to whichever bucket that has the closest mean. The sequence is then obtained by collecting the maximum values of all buckets in increasing order. If a task's peak resource consumption exceeds all values in the sequence, then that task will be allocated a whole machine. Again, the list of completed tasks is updated whenever a task completes its computation and reports its resource consumption, and the hyper-parameter n is simply set to the number of types of tasks.

VI. Evaluation

In this section, we will evaluate different allocation strategies under different assumptions of users' cooperation. While our allocation strategies can operate with no information provided from users, these strategies will excel when they are supplied with at least some information about the heterogeneity of workflows. Thus, our evaluations are performed under three levels of users' cooperation:

- **Lv1:** No information provided
- **Lv2:** The number of types of tasks in a workflow provided
- **Lv3:** The associated type for each task provided

While info-oblivious strategies and the user declaration strategy don't respond to different levels of information, the two bucketing strategies will respond as follows:

- **Lv1:** With no information, bucketing strategies assume that all tasks are of the same type and set the hyper-parameter n to 1.
- **Lv2:** Bucketing strategies set the hyper-parameter n to the number of types of tasks.
- **Lv3:** Bucketing strategies group up tasks of the same type, for all types of tasks, and set the hyper-parameter n to 1 for each group.

We set other hyper-parameters for our simulations and strategies as follows:

- All strategies are given machines with a maximum capacity of 16 cores, 64 GBs of memory, and 64 GBs of disk.
- The double allocation strategy uses $1/8^{\text{th}}$ of a machine capacity as the initial default.
- The user declaration strategy always derives an upper bound with 5% misprediction error to represent a high-quality guess from users.
- The bucketing strategies always execute 10 tasks following the whole machine strategy to obtain an initial list of resource consumption.

We will define the allocation, consumption, and waste of a resource for a task T as follows:

- *allocation* $a = r_a * t$, where r_a is the amount of resource allocated to T, and t is the execution time of T.
- *consumption* $c = r * t$, where r is the peak amount of resource consumed by T.
- There are two sources where resource waste of a task T can originate: internal fragmentation, and failed allocation. We define internal fragmentation to be $w_{if} = (r_a - r) * t$ when task T executes successfully, and failed allocation to be $w_{fa} = \Sigma_{i=1}^{k}(a_i * t)$ when task T fails with k previous allocation attempts. Thus, $w = w_{if} + w_{fa} = (r_a - r) * t + \Sigma_{i=1}^{k}(a_i * t)$.

Finally, we use two metrics to evaluate our results as follows:

1) Waste Reduction Ratio (WRR): this metric measures the relative waste reduction of our strategy compared to the

TABLE I: Waste Reduction Ratio - Unit: percentage (the higher the better)

Strategies \ Datasets	Colmena	TopEFT	Normal	Uniform	Exponential	Bimodal	Trimodal
Whole Machine	0	0	0	0	0	0	0
Double Allocation	**72.9**	88.1	71.4	-7	**57.1**	17.7	30.3
User Declaration	63.5	**99.8**	84.9	56.5	0	65.3	42.8
Quantized Bucketing - lv1	62.8	98.7	**88.4**	**61.1**	2.4	68.9	48.5
Quantized Bucketing - lv2	60.3	97.9	**88.4**	**61.1**	2.4	43.4	49.3
Quantized Bucketing - lv3	*N/A*	98.7	**88.4**	**61.1**	2.4	**96.5**	**80.8**
K-means Bucketing - lv1	62.8	98.7	**88.4**	**61.1**	2.4	68.9	48.5
K-means Bucketing - lv2	65.1	98.1	**88.4**	**61.1**	2.4	87.0	48.9
K-means Bucketing - lv3	*N/A*	98.7	**88.4**	**61.1**	2.4	**96.5**	**80.8**

TABLE II: Average Task Efficiency - Unit: percentage (the higher the better)

Strategies \ Datasets	Colmena	TopEFT	Normal	Uniform	Exponential	Bimodal	Trimodal
Whole Machine	15.8	0.60	12.4	39.1	15.7	31.3	30.7
Double Allocation	**51.9**	4.90	33.1	41.6	**27.6**	37.4	43.3
User Declaration	33.2	69.1	48.4	59.6	15.7	56.7	43.7
Quantized Bucketing - lv1	34.4	85.5	**56.3**	**62.4**	16.1	59.6	46.4
Quantized Bucketing - lv2	41.9	45.3	**56.3**	**62.4**	16.1	43.4	57.8
Quantized Bucketing - lv3	*N/A*	**91.0**	**56.3**	**62.4**	16.1	**93.9**	**71.3**
K-means Bucketing - lv1	34.4	85.5	**56.3**	**62.4**	16.1	59.6	46.4
K-means Bucketing - lv2	43.9	45.9	**56.3**	**62.4**	16.1	84.2	57.5
K-means Bucketing - lv3	*N/A*	**91.0**	**56.3**	**62.4**	16.1	**93.9**	**71.3**

whole machine strategy, and is defined to be:

$$WRR(S, W) = 1 - \frac{W_S}{W_W},$$

where S and W denote the given strategy and the whole machine strategy, and W_S and W_W are the aggregate waste of resources across all tasks in a workflow from the given strategy and the whole machine strategy, respectively.

2) Average Task Efficiency (ATE): this metric measures how efficient a task consumes its allocation on average, and is defined to be:

$$ATE(S) = \frac{1}{n} \sum_{i=1}^{n} \frac{c_i}{a_i},$$

where n is the number of tasks in a workload, and c_i and a_i are task i's consumption and allocation, respectively, as defined above.

Tables I and II show the performances of strategies in allocating memory to tasks for considered datasets under different levels of users' cooperation (we skip cores and disk for the convenience of discussion.)

Workflows with tasks' resource consumption following the exponential distribution like Colmena-XTB or exponential tend to cause more waste due to their outliers. The double allocation strategy excels when given such workflows by giving most tasks a small and sufficient allocation while catching the outliers' resource consumption by doubling their allocations before resorting to a whole machine. In the exponential dataset, our info-aware bucketing strategies barely improve upon the whole machine strategy as all tasks in this dataset are of the same type. However, in the Colmena-XTB workflow with two types of tasks, we can observe the considerable improvements in the bucketing strategies as more information about the heterogeneity of tasks is provided (We lost the level-3 information of the Colmena-XTB workflow due to framework incompatibility.)

In the uniform and normal datasets where tasks have the same type of functionality, our bucketing strategies outperform other strategies even though the information on the heterogeneity of tasks meets our assumption at level 1 and thus doesn't give us any advantages on levels 2 and 3. This is because our bucketing strategies also use the historical data from completed tasks, and consequently perform slightly better.

The bucketing strategies excel the most when provided with the associated type of task for every task in heterogeneous datasets (TopEFT, bimodal, trimodal). In the TopEFT dataset, although the bucketing strategies underperform when compared to the user declaration strategy in the WRR metric, these strategies predict most tasks' consumption very accurately, resulting in 91.0% average task consumption efficiency. In the bimodal and trimodal datasets, with the exception of the quantized bucketing strategy at level 2, bucketing strategies at any level outperform all other strategies. Additionally, the more information on the heterogeneity of tasks supplied by users these strategies acquire, the better these strategies can allocate resources to tasks.

VII. Related Work

Several research groups have surveyed and compiled a substantial number of possible strategies to boost a workload's efficiency. Witt et al. [18] provided a comprehensive study on predicting tasks' consumption using black-box modeling and machine learning methods. Pupykina et al. [11] surveyed different techniques for managing tasks' memory consumption in HPC and cloud systems.

While our paper emphasizes the importance of the heterogeneity of tasks, many works focused on other properties of a workflow. Witt et al. [19] trained a regression-based model to predict tasks' consumption given their input sizes. Rodrigo et al. [12] used information about the internal dependency structure of tasks in a workflow to reduce wait times between tasks and increase the workflow's utilization. Rodrigues et al. [13] embedded a machine learning-based tool into the LSF batch scheduler that provides predictions on tasks' memory consumption given their specifications. Tanash et al. [15] implemented an ensemble of several regression-based methods to predict a task's memory consumption and run time using the metadata and resource consumption of tasks.

Several works used historical data from previous executions of workflows or resource consumption of completed tasks to perform predictions of allocations. Tovar et al. [17] introduced methods that minimize waste and maximize throughput by allowing one retry of resource allocation per task before allocating a whole machine. Zhang et al. [20] developed a cluster-based method that analyzes resource consumption of tasks of a workflow to derive an optimal resource allocation for future workflows. Fan et al. [9] focused on the problem of task scheduling and trained a deep reinforcement learning agent to dynamically choose which tasks to be allocated and executed next. Salim et al. [14] packaged tasks into a large job to leverage scheduling policies that favor larger jobs, but required that tasks' consumption must be specified in advance.

VIII. Conclusions

We have developed resource allocation strategies that can leverage users' knowledge about the nature of tasks in workflows to predict tasks' resource consumption without placing a burden on users. These strategies yield substantial improvements in reducing waste and boosting tasks' resource consumption efficiency, thereby showing that exploiting the heterogeneity of tasks in workflows is desirable in solving the resource allocation problem. Future work to solve the resource allocation problem might include: development of new allocation strategies and enhancement of current strategies; exploration of other properties of workflows that might help in predicting resource consumption of tasks; implementation of several chosen strategies into allocation engines of distributed systems or frameworks.

Acknowledgement

This work was supported by National Science Foundation grant OAC-1931348. We thank Ben Tovar and Tim Shaffer from the University of Notre Dame for comments on the draft, and Kelci Mohrman and Kevin Lannon from the University of Notre Dame for their generosity in providing us the necessary programs and resource consumption reports for the TopEFT workflow.

References

[1] Apache airflow, 2021 [Online]. Apache Software Foundation. Available: https://airflow.apache.org/docs/apache-airflow/stable/.

[2] Coffea, 2021 [Online]. Fermi National Accelerator Laboratory. Available: https://github.com/CoffeaTeam/coffea.

[3] Colmena, 2021 [Online]. ExaLearn and Parsl Teams. Available: https://colmena.readthedocs.io/en/latest/index.html.

[4] Resource monitor, 2021 [Online]. University of Notre Dame. Available: https://cctools.readthedocs.io/en/latest/resource_monitor/.

[5] Y. Babuji, A. Woodard, Z. Li, D. S. Katz, B. Clifford, R. Kumar, L. Lacinski, R. Chard, J. M. Wozniak, I. Foster, M. Wilde, and K. Chard. Parsl: Pervasive parallel programming in python. In *Proceedings of the 28th International Symposium on High-Performance Parallel and Distributed Computing*, HPDC '19, page 25–36, New York, NY, USA, 2019. Association for Computing Machinery.

[6] C. Bailey Lee, Y. Schwartzman, J. Hardy, and A. Snavely. Are user runtime estimates inherently inaccurate? In *Proceedings of the 10th International Conference on Job Scheduling Strategies for Parallel Processing*, JSSPP'04, page 253–263, Berlin, Heidelberg, 2004. Springer-Verlag.

[7] A. Basnet et al. Topeft/topcoffea: Topcoffea 0.1 (v0.1), (2021). Zenodo. Available: https://doi.org/10.5281/zenodo.5258003.

[8] P. Bui, D. Rajan, B. Abdul-Wahid, J. Izaguirre, and D. Thain. Work Queue + Python: A Framework For Scalable Scientific Ensemble Applications. In *Workshop on Python for High Performance and Scientific Computing (PyHPC) at the ACM/IEEE International Conference for High Performance Computing, Networking, Storage, and Analysis (Supercomputing)* , 2011.

[9] Y. Fan, Z. Lan, T. Childers, P. Rich, W. Allcock, and M. E. Papka. Deep reinforcement agent for scheduling in hpc, 2021.

[10] C. B. Lee and A. Snavely. On the user–scheduler dialogue: Studies of user-provided runtime estimates and utility functions. *The International Journal of High Performance Computing Applications*, 20(4):495–506, 2006.

[11] A. Pupykina and G. Agosta. Survey of memory management techniques for hpc and cloud computing. *IEEE Access*, 7:167351–167373, 2019.

[12] G. P. Rodrigo, E. Elmroth, P.-O. Östberg, and L. Ramakrishnan. Enabling workflow-aware scheduling on hpc systems. In *Proceedings of the 26th International Symposium on High-Performance Parallel and Distributed Computing*, HPDC '17, page 3–14, New York, NY, USA, 2017. Association for Computing Machinery.

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

[14] M. A. Salim, T. D. Uram, J. T. Childers, P. Balaprakash, V. Vishwanath, and M. E. Papka. Balsam: Automated scheduling and execution of dynamic, data-intensive hpc workflows, 2019.

[15] M. Tanash, B. Dunn, D. Andresen, W. Hsu, H. Yang, and A. Okanlawon. Improving hpc system performance by predicting job resources via supervised machine learning. In *Proceedings of the Practice and Experience in Advanced Research Computing on Rise of the Machines (Learning)*, PEARC '19, New York, NY, USA, 2019. Association for Computing Machinery.

[16] D. Thain. Cctools, 2021 [Online]. University of Notre Dame. Available: https://cctools.readthedocs.io/en/stable/install/.

[17] B. Tovar, R. F. da Silva, G. Juve, E. Deelman, W. Allcock, D. Thain, and M. Livny. A Job Sizing Strategy for High-Throughput Scientific Workflows. *IEEE Transactions on Parallel and Distributed Systems*, 29(2):240–253, 2018.

[18] 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*, 82:33–52, May 2019.

[19] C. Witt, J. van Santen, and U. Leser. Learning low-wastage memory allocations for scientific workflows at icecube. In *2019 International Conference on High Performance Computing Simulation (HPCS)*, pages 233–240, 2019.

[20] 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)*, pages 1–9, 2018.