Predictive Execution of Workflows in a HPC+Cloud Environment

Subhendu Behera[1], Jae-Seung Yeom[2], Daniel Milroy[2], Marc Niethammer[3], Frank Mueller[1]
[1]*North Carolina State University*, ssbehera@ncsu.edu, mueller@cs.ncsu.edu
[2]*Lawrence Livermore National Laboratory*, yeom2@llnl.gov, milroy1@llnl.gov
[3]*University of California, San Diego*, mniethammer@ucsd.edu

***Abstract*—Meeting deadlines for data-intensive workflows on HPC systems is challenging as jobs experience varying wait times before resources become available. This impact is significant in hybrid HPC+Cloud scheduling, which can lead to resource idleness, deadline violations, and higher costs.**

To address these issues, we propose scheduling data-intensive workflows over a combined HPC+Cloud hybrid environment in a deterministic manner by scavenging unused HPC resources. We predict resource availability (RA) of HPC systems, and exploit this prediction to dynamically split resource allocation between HPC's unused and Cloud's on-demand resources to complete a workflow by a given deadline. The deterministic resource allocation allows for preloading input data for workflow tasks, avoiding execution delays. Further, we develop an adaptive scaling algorithm that effectively backs up the targeted HPC allocation on Cloud facilities to avoid workflow execution delays in the event of incorrect RA estimation. Experiments show that our scheduling technique imposes minimal impact on HPC production jobs, saves cost for >75% workflow runs, suggests accurate budgets with a mean 7.11% to 14.75% cost estimation error, and finishes a mean 98% to 99.4% of tasks before deadlines.

***Index Terms*—High-Performance Computing, Convergence Computing, Workflow Scheduling, Cloud Computing, Workflow Management System, Resource Availability**

I. Introduction

Workflows represent a wide array of applications in data analysis, knowledge discovery, and complex simulations in both Cloud and HPC environments [1]–[4]. Cloud workflows often handle big data in science and commerce (including many medical applications) while HPC workflows tend to involve multi-science numerical problems spanning different abstraction levels or are part of large device experiments, such as high-intensity lasers [5], [6]. Many modern-day scientific and big data workflows are also growing in scale, resource requirement diversity, and complexity [7].

Many studies have focused on scheduling workflows on the Cloud with deadline and cost constraints. However, data-intensive workflow execution on the Cloud incurs significant monetary cost. To avoid such a cost, users schedule workflows on HPC systems in multiple ways. Typically, Workflow Management Systems (WMS) split workflows into tasks and schedule them as individual jobs or through executors that already run as jobs while satisfying dependency constraints [8], [9]. However, HPC jobs are delayed when resources are not allocated immediately. Entire workflows can be scheduled as a pilot job to reduce the wait times for individual tasks, but the larger resource requirement delays execution startup. We can avoid the startup delay for pilot jobs by scheduling the pilot job on a reservation with a special privilege, which needs to be approved by the system administrators and cannot meet the deadline constraint. Further, pilot jobs may cause resource wastage with over-allocation. Recent works [10], [11] in converged computing indicate a growing interest in utilizing on-demand Cloud resources to address the ever-growing computation demand of HPC workflows.

Cloud Bursting (CB) is a technique primarily used by private enterprises to scale up resources in the Cloud on demand, in addition to their own permanent on-premises private Cloud [12]. However, applying CB to workflow scheduling is different from the traditional CB for Cloud workloads. Traditional CB usually relies on future workload prediction to allocate or scale additional resources (bursting) on the Cloud for task offloading. However, this methodology has two drawbacks. First, HPC system schedulers are complex and employ advanced strategies such as backfilling for better resource utilization. Predicting workloads only characterizes users' job submission behavior, not a system's scheduling behavior. So task offloading to the Cloud — based on workload behavior — may not utilize HPC resources optimally as it cannot estimate the RA on an HPC system from batches of jobs submitted by users with inaccurately estimated job durations. CB is dependent on workload behavior, such as spikes or drops in user requests, to perform bursting, assuming dedicated and on-demand local resources, which are not guaranteed on HPC systems. Second, because of the dynamic generation of input data for workflow tasks and on-the-fly offloading tasks to the Cloud in CB, data needs to be migrated just before execution begins. The lack of guidance (which and when) for offloading tasks to the Cloud prohibits data from being preloaded before execution begins.

We contribute a novel combination of HPC and Cloud in a hybrid, orchestrated manner by splitting computations across domains for a workflow to reduce delays resulting from on-the-fly data movement. Specifically, we build an execution schedule for a workflow in a HPC+Cloud environment by predicting the unused HPC resources complemented by the guaranteed on-demand Cloud resources to meet the deadline. Concerted deterministic resource allocation and task assignment allow better utilization of unused HPC resources and preload the dynamically generated intermediate input data on

HPC and Cloud to avoid execution delays. Further, we use deterministic resource allocation to compute Cloud cost by considering the required Cloud resources, storage, and inter-domain data transfer.

We developed an adaptive resource allocation/scaling method to reserve resources both on Cloud and HPC before executing a workflow per a schedule. This is accomplished by requesting the determined number of resources on HPC and Cloud on an hourly basis, with a runtime limit of one hour, thereby allowing the HPC system scheduler to allocate limited resources instantly, typically for as many single-node requests as can be accommodated by the back-filling algorithm. These underutilized resources may otherwise remain idle and are hence considered free in our model. Without a bounded wait time for HPC resource allocation, we back up the requested HPC resources with an equivalent redundant allocation on the Cloud to avoid execution delays. As the requested HPC resources are allocated, the redundant backup Cloud instances are reduced (de-allocated). This migration of computation from the Cloud back to HPC allows us to limit cost for Cloud resources to a predictable level while satisfying deadline constraints.

In summary, we make the following contributions:

• We identify contemporary challenges of scheduling large-scale workflows exclusively on HPC systems and analyze their root causes.

• We propose and develop a novel solution for scheduling large-scale workflows deterministically in that both cost and deadline can have highly reliable guarantees that can be traded off. This is accomplished by predicting RA of an HPC system, which facilitates the determination of resource requirements split between HPC and Cloud for different deadlines and avoids execution delays due to on-the-fly data movement.

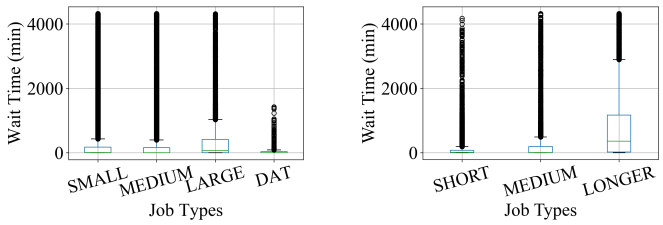
• We developed an elastic resource allocation/scaling algorithm to scale up/down heterogeneous resources on HPC and Cloud such that HPC resources are backed up by the Cloud to guarantee deadlines while avoiding RA mispredictions.

• We evaluated our method by developing a simulator for HPC+Cloud hybrid execution. We first validate our simulator by running a limited number of experiments in an HPC+Cloud setting to assess its performance. We subsequently use the validated simulation as the cost of Cloud execution in our runs would be prohibitive for the experiments we conduct. Our evaluation is based on factors such as meeting deadlines, correct estimation of cost, cost savings, and impact on other production jobs.

II. Background

To investigate the challenges and opportunities in scheduling large-scale workflows, we analyzed production jobs submitted to the Lassen supercomputer [13].

Data Collection: Lassen, a Top500 [14] supercomputer installed at Lawrence Livermore National Laboratory (LLNL), comprises 795 compute nodes. Each node is equipped with an IBM Power9 CPU and four NVIDIA V100 GPUs. The job records from the IBM CSM database [15] provide detailed



(a) Wait time vs Job Size on Lassen (b) Wait time vs Job Run Time

Fig. 1: Challenges with Large Job Scheduling

information about the job lifecycle, resource utilization, performance statistics, and other metrics. We used data from two years and two months [16], which includes production jobs submitted by real users and subject to the primary allocation algorithm. The jobs are categorized based on the requested number of nodes: small (1-7), medium (8-63), large (64-256), and DAT (256+). Notably, Dedicated Access Time (DAT) jobs are exempt from the primary allocation. Instead, DAT jobs are submitted with special privileges and are scheduled at a predetermined time.

To gain insights into the scheduling challenges, we examined the wait times for all job categories. Figure 1a illustrates the distribution of wait times in minutes for jobs waiting to run. We focused on jobs with wait times within three days and excluded those outside this range (only 2.2%) to improve visibility of the plots. The data reveals that DAT jobs have the shortest wait times (quartiles Q1 and Q3: 0-34) due to their privileged status. Small and medium jobs have wait time quartiles (Q1 and Q3): 0-171 and 0-158, respectively. In contrast, large jobs exhibit the longest wait times with quartiles (Q1 and Q3): 0-411, primarily due to their increased node requirements and scheduling under the primary allocation algorithm. Next, we categorized the jobs as SHORT ($<$2 hours), MEDIUM (2-6 hours), LONG ($>$ 6 hours) according to their requested run-time limits and analyzed their wait times as shown in Figure 1b. We observe that as the requested time limit increases, the wait times also increase. SHORT and MEDIUM jobs have lower wait times quartiles (Q1 and Q3), i.e., 0-75 and 1-194 minutes, respectively, followed by LONG jobs with quartiles (Q1 and Q3) of 22-359 minutes.

These findings suggest that large jobs submitted without special privileges (non-DAT jobs) face significant wait times. Large jobs, in terms of both run time limit and requested node size, experience significantly higher wait times than other job categories. Our scheduling technique considers each single node with a one-hour time limit so that jobs can maximize resource allocation on HPC while offloading required execution to the Cloud. Further, the communication overhead between HPC and Cloud systems can cause significant delays in application execution [17]. We address these challenges with our predictive and adaptive execution framework as described next.

III. System Design

A. Execution Model

In our execution model, the workflow tasks are processed on the executors launched by a WMS or workflow scheduler. Each executor runs on an allocated HPC or Cloud resource and executes tasks sequentially from an assigned stage in the workflow. Executors are limited to a one-hour time limit, as jobs with a small runtime limit are more likely to be allocated faster, as shown in Section II. Before running out of their allocated hour, HPC executors checkpoint their task, where the checkpoint size is assumed to be equal to the input data size. In contrast, Cloud executors exit after completion of their current task, even after the one-hour time limit expires. We maintain two mutually exclusive queues per stage in the workflow: one for HPC and one for the Cloud. The Cloud and HPC executors pull a task for execution from its assigned stage's respective queue along with the required input data location. If the input does not exist in the same domain (HPC or Cloud), it copies the data before executing. An input data is kept alive on any domain until all the tasks dependent on the data complete execution. Shared filesystems on both HPC and the Cloud are used to store the intermediate output files, whereas the final output is stored on the HPC filesystem.

The WMS runs on a reserved node of the HPC system, where we store runtime information such as task queues, resource allocation details, and the storage location of the task's outputs. This information facilitates resource allocation/de-allocation and input data transfer between HPC and the Cloud.

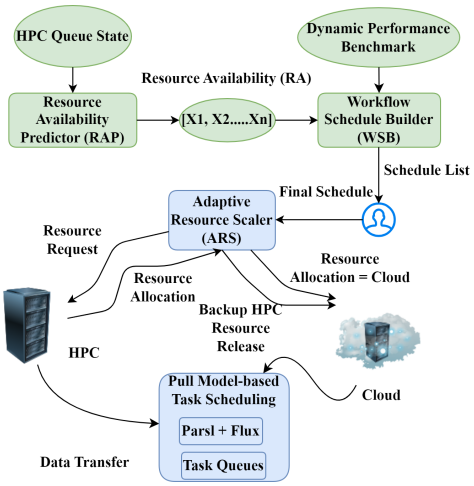


Fig. 2: Overall System Design

B. Design

Figure 2 depicts an overview of the design and identifies the different modules of our scheduling framework and their relationship. Our framework consists of two phases, namely workflow schedule construction (green) and workflow execution (blue). To build a workflow schedule, our system needs the workflow's execution profile and future RA prediction. We assume that the execution profile is unknown to the user before scheduling. Hence, our framework runs a small-scale dynamic performance benchmark (DPB) for all the stages in the workflow on the platforms resident on HPC and Cloud. Next, we predict RA on the HPC system for a deadline using our developed RA predictor (RAP) and feed it to the schedule builder (WSB) along with the DPB. The WSB calculates the additional Cloud resource requirements for each given RA to complete the execution before a given deadline. We also calculate the associated Cloud cost with each deadline replayed to the user as information to assist in the tradeoff between deadline and cost. Once the user selects an execution schedule, execution begins. Our Adaptive Resource Scale (ARS) allocates resources per the split between HPC and Cloud in the schedule on an hourly basis.

Our framework is built using Parsl [8] and Flux [18] for resource allocation and communication. The Parsl library supports parallelization of Python and Bash functions with multiple execution models across various platforms, such as an HPC system, Cloud, and a local computer. We use Parsl to launch task executors on both HPC and Cloud. Flux is a workload and job management system that can be nested within another system scheduler or itself at multiple levels while providing various communication patterns among the instances of Flux. Our scheduler and the task executors use Flux capabilities to exchange scheduling information and relevant notifications. Next, we describe various components of our framework and their methodology in detail.

Resource Availability Predictor (RAP): The RAP is an integral part of our scheduling technique. The RAP takes the HPC system's job history as input and applies Machine Learning (ML) techniques to predict future RA. The output of the RAP is a vector, X, where each element x_i represents the minimum number of unused/free nodes during the i^{th} hour. End-to-end training and the prediction procedure are described in Section IV.

Dynamic Performance Benchmark (DPB): To estimate the resource allocation split between HPC and Cloud for a workflow, the WSB needs to know the comparative performance of each stage on multiple platforms of HPC and Cloud. The DPB provides the required performance model along with a stage's input/output size. It is constructed by dynamically running and recording the performance benchmark for all stages of the workflow on all platforms (including HPC and Cloud) before completion of either one hour of runtime or 5% of total tasks. Since HPC nodes may not be immediately available, we continue building the pending HPC benchmarks. However, we assume a default five-minute execution time for the missing benchmarks in the DPB for the initial schedule construction and cost estimation. Once HPC nodes are allocated and their performance benchmarks are constructed, they are integrated into the DPB. We then update workflow schedules dynamically as discussed later. Since building such a multi-faceted DPB is expensive, we execute these runs on HPC platform resources [19] listed in Table I. We assume that these are part of the Cloud along with associated costs.

Cloud Cost Model (CCM): The CCM represents the hourly cost of using on-demand platforms from a Cloud provider. We use multiple platforms, as shown in Table I to build the multi-

faceted DPB. Each platform has an associated per-hour cost for running workloads on it. Usually, GPU platforms have higher per-hour reservation costs compared to CPU-only platforms. The CCM is static and does not change over time as we use on-demand Cloud resources.

TABLE I: Cloud Cost Model

HPC Platform	AWS Platform	Cost ($/hr)	Compute
Epyc Rome	c5a.4xlarge	0.616	CPU
Intel Skylake	c4.4xlarge	0.796	CPU
Intel Broadwell	c5n.4xlarge	0.864	CPU
Nvidia RTX 2060 Super	g4ad.4xlarge	0.867	GPU
Nvidia RTX 4060 ti16g	g3.4xlarge	1.14	GPU
Nvidia RTX 4060 ti8g	g5.xlarge	1.006	GPU

Building Workflow Schedule: Given the RA, CCM, and DPB, the WSB utilizes the following steps to build a schedule to meet the required deadline with a targeted task size. The output schedule contains hourly information about how many resources are required to run the executors on HPC and Cloud to meet the deadline.

In our execution model, each stage is executed as a bag-of-tasks with input data dependency between stages. This results in a pipeline execution as seen in Fig. 3, where a stage's execution is comprised of slots from start to end. Stages with parents face a delay in execution start because of dependencies, e.g., stage j in Fig. 3. We apply depth-first search to find such delays to decide the start time for each stage. Further, there are infeasible slots of parent stages whose output tasks cannot be processed by their children due to a lack of available slots before the deadline, as shown for stage i. We remove such infeasible slots to determine the end times of each stage and the deadlines.

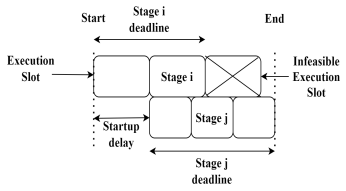


Fig. 3: Pipelined execution of the stages in workflow

After deducing the start and end times, the next stage of schedule building is to allocate resources with constraints in Eq. 1 and Eq. 2. Eq. 1 states that the sum of hourly throughput, $TP(p)_i$, for each stage, p, in the workflow is greater than or equal to the task size to ensure all tasks are executed before the deadline D. This is complemented by Eq. 2, which ensure that two stages p and q with data dependency have almost equal hourly throughput so that the task pipeline of the workflow never remains empty to avoid resource idleness.

$$\forall p \in \mathbb{W},\ \sum_{i=1}^{D} TP(p)_i \geq N \tag{1}$$

$$\forall p \in W,\ \forall q \in Parent(p),\ TP(p)_i \approx TP(q)_i \quad \text{for } i \in \{1, \ldots, D\} \tag{2}$$

To build the schedule for the entire workflow, we start with the stage with the highest Cloud cost per Eq. (3) to reduce allocation cost by preferring HPC resources. Cloud cost is derived from the sum of storage, data transfer cost, and the minimum execution cost on all the selected Cloud platforms, given by set R for stage p per Eq. 3.

$$storage\_cost(p) + xfer\_cost(p) + \min_{\forall r \in R} exec\_cost(p) \tag{3}$$

We start with the allocation of HPC throughput (number of slots available times number of nodes available) on an hourly basis in proportion to nodes in the predicted RA. Let TP_i^{HPC} be the HPC throughput during the i^{th} hour and TP^{HPC} be the total throughput available for the costliest stage. We calculate the hourly throughput target on HPC using Eq. 4, where T is the total number of tasks. If we cannot execute all T tasks solely on HPC, then tasks need to be offloaded to the Cloud. In that case, we distribute the Cloud throughput (i.e., the remaining tasks beyond the HPC target) among the hours where the allocated HPC throughput is under-allocated due to a lack of HPC resource availability. This is driven by our aim to have a balanced throughput during the execution. We skip the details of the Cloud target distribution for brevity. After the HPC and Cloud throughput targets are decided, the required amount of resources on HPC and Cloud can be calculated by dividing the throughput target TP_i^{HPC} by the number of available execution slots during the hour. Specifically, for Cloud, we choose the most cost-efficient resource from the DPB.

$$Target_i^{HPC} = min(T \times TP_i^{HPC} \div TP^{HPC}, TP_i^{HPC}) \tag{4}$$

Next, we allocate the targets for the other stages by traversing from the costliest stage and applying the conditions shown in Figure 4. Here, we calculate the throughput targets at the slot level for the immediate parents and children. In scenario A, a parent stage's execution slot is longer than that of its child. The tasks produced by a parent's slot should be processed next in the child's slots, which start executing during the parent's following slot (color-coded). In scenario B, the parent stage's execution slot is smaller than the child's. In this case, the child's slots should process all the tasks produced by the parent's slots that start during the child's previous slot.

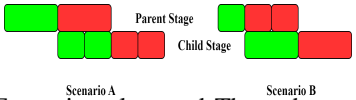


Fig. 4: Execution slots and Throughput allocation

We calculate the hourly throughput by accumulating the throughput from the slots. Based on the hourly targets, we calculate the required allocation from HPC, if available, and Cloud. The final step is to combine HPC and Cloud allocations to form the workflow schedule. This method produces one schedule given the start time and deadline for a workflow.

Cost Estimation: After constructing a schedule, we estimate the cost of workflow execution by considering Cloud resource reservations in the schedule, deriving the Cloud filesystem operations, and data transfers between HPC and Cloud. Since HPC execution utilizes only otherwise unused single nodes via backfilling, its cost is considered free. The

Cloud resource reservation cost is the sum of hourly costs of allocation for all stages in the workflow over the schedule, given the CCM. Further, we compute the amount of data read and written for the Cloud filesystem due to task execution. To estimate the data transfers and the associated filesystem operations, we compute the difference in Cloud throughput and HPC throughput between a stage and its parents. This gives us the direction and amount of data movement per stage. These calculations are done on an hourly basis and aggregated to be fed into the CCM model to get a dollar cost quote. The resource reservation and filesystem costs are modeled based on Amazon EC2 platforms and Amazon's EFS, respectively. We only account for Cloud to HPC data movement cost since data movement into the AWS cloud is free.

C. Data Preloading Strategy

Our splitting of workflow between HPC and Cloud may result in delayed execution of tasks when large input data does not reside in the same domain. This results in delayed workflow execution and increased cost because of resource idleness. As a mitigation, we built an adaptive preloading strategy to place input data near the targeted computation. To this end, we create the task-to-domain mapping as soon as a task spawns a stage in the workflow. The task-to-domain mapping indicates whether a task executes on Cloud or HPC. We rely on the domain's task load, i.e., task queue size/number of executors, for a stage to determine the mapping. If the load is less than 1, we map the task to the corresponding domain. To break ties, we prioritize Cloud to reduce resource idleness cost and data movement latency, both contributing to execution time savings. After the mapping, we copy the relevant input data to Cloud or HPC asynchronously without affecting computation. To facilitate this, we maintain two separate task queues, each for HPC and Cloud per stage in the workflow. Although the task-to-domain mapping dictates task enqueuing, our strategy is not strict with respect to the mapping. Idle resources can still execute tasks from the other domain's queue when that domain's load exceeds 1.

D. Adaptive Resource Scaler (ARS)

The second phase of workflow execution relies on the ARS to make adaptive and elastic resource requests for both HPC and Cloud per the workflow schedule selected by the user. The scaling mechanism of ARS executes at two different configurable intervals: long and short.

ARS at long intervals: ARS requests for resources on HPC and Cloud with a reservation time limit that is equal to this longer ARS interval based on the schedule constructed using the RA prediction. When a misprediction occurs, e.g., when over-estimating the available HPC resources, we may miss the deadline due to a lack of resources. Further, an imbalance in resource allocation for workflow stages can lead to resource wastage. When HPC resources are requested, there is no guarantee that they are allocated immediately, unlike Cloud resources with modeled availability at a $\approx$ two-minute delay [20]. HPC resources are released automatically, whereas Cloud resources require manual termination.

ARS at short intervals: To avoid computation shortage from RA misprediction and delayed HPC allocation, we request additional Cloud allocations on top of the scheduled ones. These excess Cloud allocations mirror the scheduled HPC allocations at the longer intervals in the schedule. This allows us to begin the execution of the stages without waiting for HPC resource requests to be met and allocated. This also addresses the issue of mispredictions that overestimate the available HPC resources. After scheduled requests are made at longer intervals, ARS executes at short intervals (five minutes by default) to continuously check if HPC resource requests are allocated. As the requested HPC allocations are granted, an equivalent amount of additional Cloud resources is released as they are no longer needed during the remainder of the long interval. This incurs additional up-front Cloud reservation costs for backing up the HPC allocations in a schedule, yet only for a short time if HPC resources are granted; any remaining Cloud resources contribute to workflow allocation. This up-front Cloud cost is subject to evaluation in our study.

Rebalancing Resource Allocation: We also adapt our allocation strategy at long intervals by rebalancing the resource allocation based on the deficit (if any) on achieved throughput in the past. We keep track of the targeted throughput and the achieved throughput on an hourly basis and over-allocate Cloud resources as required.

IV. Resource Availability Predictor (RAP)

The RAP is the key component that enables elastic workflow scheduling on HPC systems. By predicting resource availability, RAP constructs a schedule for a given workflow, providing informed decision-making capabilities regarding resource allocation and estimating the cost of execution. We describe the end-to-end build procedure of RAP on the Lassen HPC system, using data collected in Section II.

A. Feature Data Set

HPC systems typically distinguish between two primary job classes for scheduling: wait and run. When a job is submitted, it first enters the wait queue. Once deemed ready to run, the scheduler removes the job from the wait queue, enqueues it in the run queue, and starts executing it. The wait queue tracks a job's requested number of nodes, submission time, and requested hours of runtime. Similarly, the run queue tracks a job's allocated number of nodes, execution start time, and time left to completion. The information in the run queue indicates the current number of nodes running and those expected to be released soon. Conversely, the wait queue reveals the current and future demand for nodes. By combining data from both queues, we can predict resource availability on an HPC system.

We record states of the run and wait queues instead of their complete snapshots, as this was sufficient in our experiments to make predictions. The captured state of the run queue represents the number of busy nodes expected to be available in the future. We use a one-dimensional vector data structure to represent this information up to 64 hours, with a maximum limit based on the fact that this is the maximum number of hours of any job requested in the data logs collected from

the CSM database (likely also the upper bound per system configuration). The state of the wait queue represents the number of requested nodes for different hours of runtime. Given a maximum of 64 hours, we represent the state of the wait queue as a one-dimensional vector of length 64.

Example: Consider three jobs waiting in the Wait queue with a descriptor of (jobid, no. of nodes requested, no. of hours requested):
`(job4, 3, 1), (job5, 5, 5), (job6, 10, 2)`.

Given these jobs, the wait queue state will be `[3,10,0,0,5]`. Similarly, consider three jobs in the run queue with a description (jobid, no. of nodes requested, no. of hours remaining):
`(job1, 3, 2), (job2, 5, 1), (job3, 10, 5)`.

Given these jobs, the run queue state will be `[5,8,8,8,18]` with vector size 64.

The job submission and scheduling on Lassen is managed by IBM's LSF [21] workload manager. LSF employs multiple scheduling policies, such as FCFS, Fair Scheduling, Backfill, SLA, and preemption, to efficiently accommodate users' resource requests. The Lassen supercomputer employs fair share-based scheduling that requires user information and their past usage. Since user information is unavailable, we assume that FCFS (primary) + Backfill (secondary) scheduling policies are applied to the jobs submitted. We simulate the production jobs history on our HPC simulator (see Section V-A) and collect the feature data representing the temporal state of the wait and run queues. We also use the day of the week and the hour of the day as feature variables during ML training and prediction. Finally, we apply sampling to the collected feature data set with 30 and 60-minute frequencies.

B. Training and Results

To predict RA, we apply both time series and regression-based ML techniques to train our prediction models. Specifically, for time series-based ML, we use the Recurrent Neural Network (RNN)-based Long Short-Term Memory (LSTM) method [22]. With LSTM, an input sample is comprised of the history (input steps) of states within the run and wait queues, including the day of the week and the hour of the day. The output indicates RA for up to 24 hours (output steps). Further, we use different combinations in a tuple consisting of (resample frequency, training loss function, and optimizer) during training as shown in Table II.

TABLE II: LSTM Training Parameters

Parameter	Values
Input Steps	[6, 12, 18, 24]
Output Steps	[6, 12, 18, 24]
Resample Frequency	[30, 60] (minutes)
Loss Function	[MAPE, RMSE, RMSLE, HUBER]
Optimizer	[SGD, ADAM]

We also trained ensembles of Decision Trees (DT) using Gradient Boosting via the XGBOOST library [23]. XGBOOST parallelizes Gradient Boosting to train ensembles of DTs to improve performance while scaling. To train the DTs, we first transform the time series input samples into regression data. We do this by considering a snapshot of the run and wait queue states as an input sample and RA as an output sample (up to 24 hours). For DTs, we use combinations of output steps, resample frequencies, and loss functions (without MAPE) from Table II as training parameters.

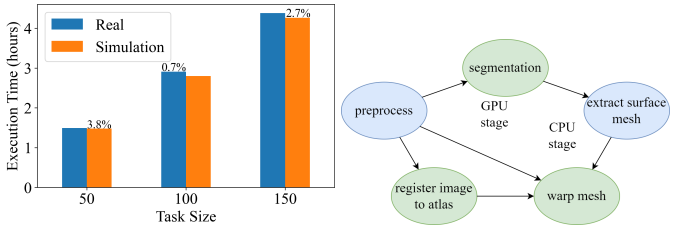
To verify the prediction accuracy and compare the ML models, we use Mean Absolute Error (MAE) as a metric during validation. We consult Lassen's records of jobs from September 18, 2018, at 12 PM to November 18, 2020, at 11 AM. We use the records until May 18, 2020, 12 AM (18 months) for training of the models. The records from May 18, 2020, 12 AM until October 18, 2020, at 12 AM (5 months) are used for testing, resulting in an 80%/20% split between training and testing data. We observed that the MAE for LSTM (73.49-92.20) is higher than for DT (51.61-75.07) with the Huber loss function and a 60-minute resample frequency. Consequently, we choose DTs trained with Huber as the RA predictor for simulation/validation of our scheduling technique. We re-train the DTs by concatenating the training and testing data to improve accuracy. We reserve the unseen records from October 18, 2020, at 12 AM until November 18, 2020, at 11 AM (1 month) for simulation in Section V.

V. Evaluation

A. HPC+CLOUD Simulator

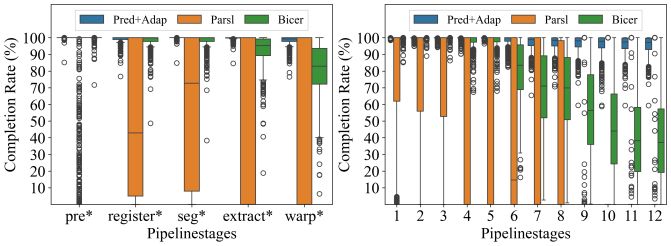
To evaluate our solution, we developed a simulation framework utilizing Python-based SimPy [24], a process-based discrete event simulation library. Given the prohibitive cost of conducting extensive experiments on commercial Cloud resources, we opted for simulation as an alternative. In this framework, we simulate an HPC system with wait and run queue managed by the FCFS+Backfill scheduling policy. Further, the task executors (both HPC and Cloud) are simulated using SimPy event generator functions, which pull tasks from the task queue and simulate the times of (a) execution derived from the DPB, (b) reading input files, and (c) writing output files. The scheduling algorithm on our simulator creates resource requests to the HPC scheduler for HPC executors and directly creates Cloud executors to simulate the execution of a workflow.

AWS Cloud Validation: We validated our simulator by running the OAI Analysis workflow (see Section V) with our Flux+Parsl scheduling framework on an 80-node HPC cluster and AWS cloud. We ran the workflow with static resource allocations on the HPC cluster and AWS cloud with input data preloading support (see Section III-B). We utilized GPUs on the HPC system (NVIDIA RTX 2060 Super) and CPUs on AWS EC2 (c4.large and c4.xlarge). We fed the traces from the execution to our simulator to run the workflow. Fig. 5a shows the execution times (y-axis) of the workflow on the HPC-AWS Cloud hybrid platform and our simulator for task sizes ranging between 50 and 150 (x-axis), with annotations indicating the percentage difference between the two. The results show that our HPC+Cloud simulator resembles the workflow execution on the HPC-AWS cloud system with good accuracy.



(a) AWS Cloud Validation (b) OAI Analysis Workflow

Fig. 5: Validation and OAI Analysis Workflow



(a) OAI Analysis (b) RNASEQ

Fig. 6: Task Completion Rate

B. Results

Experiments are conducted to simulate the HPC+Cloud hybrid execution of two real-world scientific workflows, OAI analysis and RNASEQ, on Lassen backed up by Amazon AWS cloud. We evaluate multiple scheduling algorithms, including ours:

- *Pred+Adap: Our* new scheduling technique.
- *Bicer:* [25] This technique dynamically allocates resources on the public Cloud and HPC by considering unprocessed tasks, data transfer latency, and available HPC nodes. *Bicer* [25] developed two algorithms to complete bag-of-task applications either within a given deadline or a budget constraint. We implemented the algorithm for a given deadline. *Bicer* requests HPC nodes with runtime limit equal to the remaining of the deadline.
- *Parsl-HPC:* [8] We also compare it to an HPC-only solution based on Parsl's execution model. In this technique, we request a node for one hour runtime limit and assign the node, when allocated, to an incomplete stage following a topologically sorted order.

OAI Analysis: The Osteoarthritis Initiative (OAI) collected large-scale imaging data to investigate knee osteoarthritis. We build on developed analysis workflows [5], [6] for the analysis of 3D magnetic resonance images (MRIs) of the knee, which include imaging data such as segmentation, thickness measurement, atlas-registration, and 3D to 2D mapping of thickness maps, all based on the knee MRIs. An input image needs to be processed by all the stages in the workflow, creating a task per stage. Fig. 5b shows the DAG (Direct Acyclic Graph) of the workflow along with the data dependencies among its nodes, and different stages require either CPU or GPU resources for computation. The blue stages run on CPUs while green ones execute on GPUs.

RNASEQ: RNASEQ is an RNA sequence analysis pipeline [26]. We use data from the bladder cancer cells study as input [27]. The workflow contains a total of 12 stages, each of which is executed on a CPU. RNASEQ is larger compared to OAI in terms of the number of stages and input/output data size.

Both workflows need significant data transfer, with some stages needing input data of size 240 MB and 14.5GB for OAI Analysis and RNASEQ, respectively. The data preloading technique and dynamic task assignment work together to minimize the impact of data transfer delays.

To simulate workflow execution, we first collected the execution traces on different platforms (see Table I). On each platform, we process 200 images for OAI Analysis and 100 images for RNASEQ, and record the execution times of tasks per stage. We use this execution time distribution to resemble task execution during simulation. Further, we build multiple test cases with different combinations of the number of tasks, deadlines, and workflow execution start time (see Table III). For HPC system simulation, we used job records from October 18, 2020, at 12 AM till November 18, 2020, at 11 AM (one month). Since our analysis is based on the distribution of results data, we primarily use quartiles (Q1,Q3) to describe the quantitative results unless mentioned otherwise.

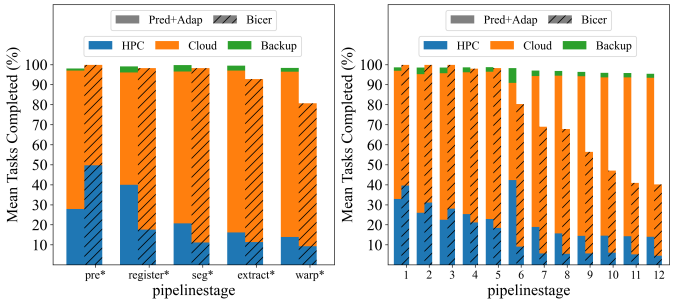
TABLE III: Test Cases

Workflow	Input Sizes	Deadlines (hour)	#testcases
OAI Analysis	12,000-96,000	6, 12, 18, 24	1,000
RANSEQ	1,000-8,000	6, 12, 18, 24	1,000

Observation 1: Our scheduling technique's ability to accurately predict RA and make adaptive resource allocation via backing up HPC allocation on Cloud achieves a high task completion rate for a given deadline.

To assess the efficiency of our scheduling technique, we compared the task completion rates of workflow runs. Fig. 6 depicts boxplots of completion rates in percentage of total number of tasks (y-axis) for all workflow runs over the different scheduling techniques for the stages (x-axis) in the workflow. We observe that our scheduling strategy *Pred+Adap* achieves the highest completion rate of (97.79%,99.99%) and (94.44%,99.99%) for OAI and RNASEQ, respectively. Our RA prediction-based scheduling strategy not only estimates the HPC+Cloud resource requirement via RA prediction, but also handles uncertain delays in resource allocation for HPC with temporary redundant resources on Cloud, which results in a high rate of task completion for the entire workflow, with a few outliers. *Parsl-HPC*'s scheduling strategy has the lowest completion rate (0%,100%) and (0%,0%) for OAI and RNASEQ, respectively, due to a lack of HPC resource availability. Further, *Bicer*'s scheduling strategy also exhibits a lower completion rate of (72.1%,93.6%) and (36.01%, 77.77%) for OAI and RNASEQ, respectively. As *Bicer* requests longer HPC jobs and lacks the knowledge of the uncertainty of HPC allocation, it results in lower HPC throughput than required. Further, we avoid resource idleness with our data preloading and resource allocation rebalancing strategies.

To better understand the importance of resource allocation strategy driven by RA prediction, we analyzed the tasks processed by HPC, Cloud, and Backup HPC allocation. Fig. 7 shows the average percentage of total tasks (y-axis) com-



(a) OAI Analysis (b) RNASEQ

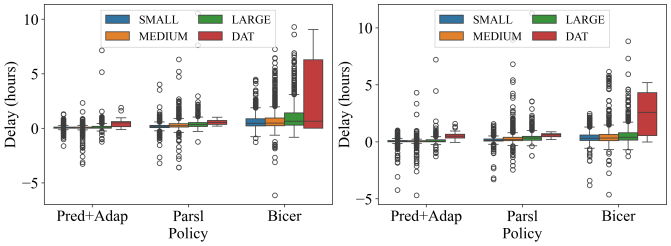
Fig. 7: Percentage of tasks completed by resources

pleted by different resource groups in the applied scheduling techniques for all the stages (x-axis) of a workflow. Bicer's model can allocate the required HPC resources to execute the tasks for the initial stages of both workflows, OAI Analysis (first) and RNASEQ (first three). However, for the later stages, Bicer's model cannot obtain the necessary HPC resources, which leads to lower throughput by Cloud resources due to idleness. The incorrect assumption made by Bicer's model that the required resources are always available means it cannot allocate enough HPC resources for all the stages in the workflow. Our Pred+Adap model's ability to predict RA yields better HPC resource allocation and adapts well with backup HPC resources.

Observation 2: Estimating RA accurately and backing up HPC resource requirements via Cloud reservations minimizes the impact on the schedule of HPC production jobs.

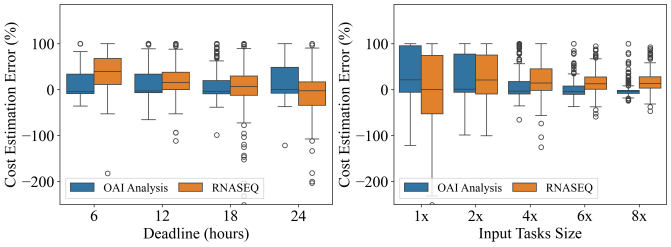
To measure the impact of our RA prediction-based hybrid scheduling on production jobs (jobs other than our backfilled workflow), we assess the change in wait times of the production jobs that were submitted by other users on Lassen during the execution of our workflow. Fig. 8 shows the delays in hours (y-axis) experienced before starting production jobs over different job groups (x-axis) over all 1000 workflow runs for all the scheduling strategies. A positive value means a production job was delayed because of resource usage by our workflow. Conversely, a negative value indicates an earlier start of a production job.

We observe that our scheduling technique *Pred+Adap* imposes small delays of (0, 0.12) hours in most cases on production jobs across all job groups. However, *Bicer* affects the production HPC jobs by causing a larger delay of (.23, 0.99) hours for all jobs, as it does not limit the number of HPC resources due to a lack of RA prediction. *Parsl-HPC*'s delay impact on production jobs is a little higher than *Pred+Adap* with (.08, 0.40) hours. This impact is lower than that of *Bicer* as we request an HPC resource only after the previous request is complete, resulting in shorter wait queues. In contrast, *Bicer*'s requests for resources with maximum runtime limit (up to the deadline), while we benefit from our one-hour runtime limit for both *Pred+Adap* and *Parsl-HPC*. We also observe that the delay in start times increases with the node size of production jobs for all the scheduling techniques. This is because larger jobs like DAT and LARGE are more sensitive to the FCFS + backfilling HPC scheduling strategy.



(a) OAI Analysis (b) RNASEQ

Fig. 8: Delay of Production Jobs



(a) Over Deadline (b) Over Tasks Size

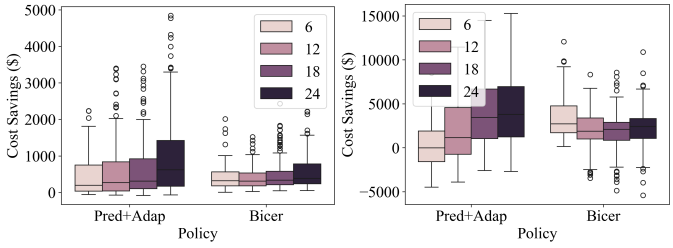
Fig. 9: Difference between Predicted and Actual Costs

Observation 3: We provide cost predictions of workflow execution in the HPC+Cloud hybrid environment with a mean underestimation of 14.75% and 7.11% for OAI and RNASEQ, respectively.

To measure the accuracy of our cost estimator, we measured the percentage difference between predicted and actual costs relative to (divided by) actual cost for all workflow runs. A positive value indicates cost underestimation. In contrast, a negative value indicates cost overestimation. Fig. 9 shows the distribution of cost error in percentage (y-axis) for all workflow runs. Fig. 9a provides the cost estimation error with respect to the workflow deadline. Fig. 9b depicts results for the underestimated cost relative to input size. We removed 45 outliers out of 2000 data points for better visibility of the data.

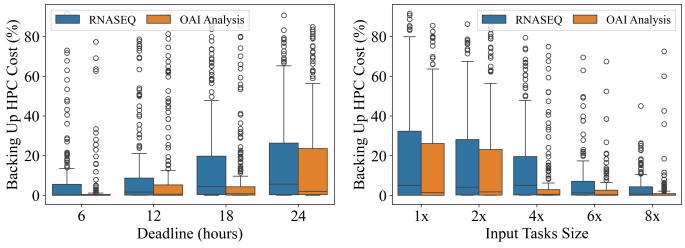
Our cost predictor results in errors that most commonly lie between (-7.95%, 38.47%). However, we observe a cost estimation error of (-6.69%, 44.21%) for RNASEQ compared to OAI's (-8.22%, 30.38%). Cost underestimation primarily happens because we do not account for the backup HPC resource allocations on the Cloud. In contrast, cost overestimation is primarily due to overestimating the Cloud allocation cost that results from an incomplete DPB. For a larger workflow, its DPB construction may remain incomplete for the stages with a longer execution start. As we assume a five-minute default execution time for the missing stages (mostly with less than 5 minutes of execution times) in DPB, we may end up overestimating the cost.

Fig. 9a shows that the cost estimation error (y-axis) distribution for the OAI Analysis workflow remains steady as the deadlines become longer (x-axis). However, for RNASEQ, cost overestimation grows with the deadline. Larger HPC usage due to longer deadlines can also result in cost overestimation when the DPB for the HPC resources is incomplete and Cloud allocation cost is overestimated, as it happens for RNASEQ. Fig. 9b indicates that cost estimation error (y-axis) decreases as input task size increases (x-axis). This is due to



(a) OAI Analysis (b) RNASEQ

Fig. 10: Cost Savings



(a) Backup Cost in % over Deadline Length (b) Backup Cost in % over Tasks Sizes

Fig. 11: Cost of Backing up HPC in the Cloud

the cost of backup HPC resources that we do not include in our predicted cost. For smaller input task sizes, the predicted dollar cost is so small that it is often close to the cost of backing up HPC resources on the Cloud (i.e., approaching 100%), which can be further amplified by data movement cost during execution.

Observation 4: *Pred+Adap*'s scheduling approach in an HPC+Cloud hybrid environment yields significant cost savings.

To assess the dollar-cost benefits of our proposed scheduling technique, we compared the cost of executing a workflow run completely on the Cloud against our *Pred+Adap* but exclude *Parsl-HPC* since it executes only on HPC. Fig. 10 shows the distribution of cost savings in dollars (y-axis) for all workflow runs for different deadlines. A positive value suggests a lower cost for a scheduling technique compared to Cloud-only scheduling; conversely, a negative value implies a higher cost. As can be seen, *Pred+Adap* yields better cost savings with a mean savings of 23.96% and 28.59% for OAI and NASEQ, respectively, compared to *Bicer*'s 16.82% and 26.5% for OAI and RNASEQ, respectively. The higher savings for *Pred+Adap* culminate from better usage of HPC resources than *Bicer*. Our scheduling technique predicts RA to better understand how much Cloud resources will be required and also addresses the uncertainty in HPC resource allocation with backup resources on Cloud. Further, we observe that savings increase with longer deadlines as a larger number of unused HPC nodes (void of dollar cost) is allocated. Overall, from Observations 4 and 5, we deduce that the higher RA of HPC can yield better cost savings, but can also result in a higher cost overestimation error.

Observation 5: Accurate RA prediction leads to lower cost of backing up HPC resources via the Cloud.

We back up requested HPC resources via Cloud reservations (see Section III-B) to avoid RA mispredictions, which may delay workflow execution. To measure the accuracy of our RA prediction, we calculated the reservation cost of the additional Cloud resources due to backing up delayed HPC resources. Fig. 11 shows this backup cost (y-axis) as a percentage of the total Cloud cost. Fig. 11a depicts this cost for different deadlines (x-axis) while Fig. 11b plots the cost over different input task sizes (x-axis). We observe that the additional backup cost is only ≈0-4% and ≈0-14% for 75% of the workflow runs for OAI and RNASEQ, respectively. The overall average additional backup cost is ≈10.5%. This suggests that our HPC resource requests are immediately satisfied thanks to accurate RA predictions and the lower runtime limit of one hour, which contributes to a better utilization of the backfilling capacity for the HPC scheduler. Considering this accuracy, combined with the minimal impact our scavenging method has on production HPC jobs, our RAP predictor and adaptive scheduler can be deemed highly effective in executing large-scale workflows in an HPC+Cloud hybrid manner without perturbing normal HPC operations.

We further observe that larger deadlines lead to small increases in backup cost (see Fig. 11a). This is due to increased HPC resource usage with longer execution times. The results further indicate that backup cost decreases as the input size grows (Fig. 11b). This is due to the increment in Cloud resource requirements as task sizes become larger, leading to a lower HPC-to-Cloud resource usage ratio.

Observation 6: The importance of techniques such as data preloading, backing up HPC, and rebalancing resource allocation increases as workflow size grows.

Although our scheduling approach is primarily based on RA prediction, it is supported by multiple techniques, including data preloading, backing up of HPC resources, and rebalancing of resource allocation. We performed an ablation study to measure the impact of these techniques on our model. Specifically, we compare the task completion rates (Q1, Q3) for our full model against models without one of these techniques. The results, as shown in Table V, show that all the techniques have a significant performance impact, improving the task completion rate to meet deadlines. More importantly, the impact of these techniques grows as the workflow size (both node and data) grows, with lower completion rates of (85.17%, 99.28%), (84.11%, 97.3%) and (91.32%, 99.9%) without data preloading, rebalancing and backup HPC, respectively, for RNASEQ compared to (93.27%, 99.97%), (89.11%, 97.15%) and (97.18%, 99.99%) for the OAI Analysis workflow. Further, data preloading and resource rebalancing techniques are more impactful than the backup HPC technique for both workflows.

Observation 7: Backup resources on Cloud for HPC allocations have minimal impact on task processing.

To measure the impact of backup resources on the workflow scheduling, we analyze their average reservation time and the number of tasks processed by them. A backup resource for HPC has an average reservation time of 0.4 hours (1 hour time limit) and contributes only to 2.7% of total tasks on average for OAI Analysis. For RNASEQ, the reservation time is 0.5 hours and the processing contribution is 3.26%. This suggests that our RA prediction is effective in minimizing the impact

TABLE IV: Hybrid Scheduling Comparison

Contributions	Model	Scheduling Strategy	Predictor	Deadline Constraint	Cost Reduction	Cost Estimation
Our Work	HPC+Cloud	scavenge unused HPC nodes	HPC Resource Availability	✓	✓	✓
[25], [28]	HPC/Private+Public Cloud	offload to Cloud	N/A	✓	✗	✗
[12]	Private+Public Cloud	offload to Cloud	future workload	✗	✓	✗
[29]	Local Cluster+Cloud	optimized placement	N/A	✗	✓	✗
[30]	HPC+Cloud	optimized placement	N/A	✗	✓	✗

TABLE V: Task completion rates (Q1, Q3) without Data Preloading, Rebalancing and Backup HPC

Model	OAI Analysis	RNASEQ
Full Model	(97.79%, 99.99%)	(94.44%, 99.99%)
w/o Data Preloading	(93.27%, 99.97%)	(85.17%, 99.28%)
w/o Rebalancing	(89.11%, 97.15%)	(84.11%, 97.3%)
w/o Backup	(97.18%, 99.99%)	(91.32%, 99.9%)

on HPC production jobs while providing a higher rate of task completion.

Observation 8: Our dynamic approach to building performance benchmarks with the DPB is economical and avoids a bloated budget.

Instead of relying on expensive benchmarks, we built the DPB with a cumulative target of only 5% of total task size or 1 hour of run time, whichever finishes earlier, for all the platforms. The cost and duration of building the DPB are included in the final cost and the set deadline, respectively. The average cost of the DPB is only 3.24% and 3.13% for OAI and RNASEQ, respectively. The (Q1, Q3) values are (.26%, 1.48%) and (0.25%, 1.45%) for OAI and RNASEQ, respectively.

VI. Related Work

Workflow Management Systems (WMS), such as Pegasus [31], Nextflow [9], Parsl, and Snakemake [32], provide support for scaling and scheduling workflows across HPC and Cloud Environments. While they perform several scheduling optimizations, such as task clustering and data-aware scheduling, the decision to offload tasks from HPC to Cloud is left to the user's discretion. WMSs do not provide insights or heuristics for informed decision-making regarding scheduling and scaling based on factors such as RA, deadline, and cost.

Several techniques have been developed to schedule workflows in an HPC+Cloud or private-public Cloud hybrid environment to meet deadlines. Aneka [28] and Bicer et al. [25] employ a dynamic approach where resources are scaled up/down at run time by periodically calculating available resources from the local cluster, the pending tasks, data coordination overhead and the required additional resources from Cloud. However, their assumption of on-demand HPC resources seems unrealistic as it often results in delayed allocation and insufficient computational capacity, leading to idle Cloud resources and lower completion rates. In contrast, our technique mitigates the resource availability issue through RA prediction and backing up of HPC resources on Cloud with temporary (until HPC requests are met) redundancy ensuring the necessary computational capacity is maintained on both HPC and Cloud platforms. Another difference is that both Aneka and Bicer overallocate resources to compensate for data movement overhead, whereas our technique applies data preloading to avoid such overhead, thus not allocating additional Cloud resources.

Guo et al. [12] primarily performs CB on private Cloud by offloading requests to public Cloud while predicting when the private Cloud is overloaded using a workload forecaster. Further, it optimizes the cost of offloading by selecting applications with lower Cloud costs and primarily focuses on resource scaling under high workloads without a deadline constraint for workflows. Their work does not focus on complex workflows and relies on workload behavior to scale up/down public Cloud resources on demand. Gupta et al. [30] utilizes HPC jobs' performance metrics and resource requirements for optimal job placement on HPC and Cloud. Assuncao et al. [29] evaluated the cost of using additional Cloud resources to supplement the lack of resources on a local cluster. They evaluated multiple job placement strategies based on various performance metrics such as job wait times, deadline violations, and the cost of Cloud. These studies primarily focus on optimizing the local cluster scheduler by determining the optimal placement (local or cloud) for all jobs on an HPC system. They do not account for workflows with complex data dependencies or data movement costs. Additionally, they lack mechanisms for estimating future resource availability and rely on users to estimate the required resources. In comparison, our work focuses on large-scale workflows to split them across HPC and Cloud while taking advantage of unused resources on HPC.

Existing studies either do not consider deadline constraints, assume that resources are available on demand, which is not true in HPC, or do not consider data dependencies between jobs. Therefore, none of them is directly applicable to solve the deterministic scheduling problem. Deterministic scheduling of large-scale workflows with deadline requirements has the following requirements: estimation of required resources, estimation of resource availability, considering data overhead in case of Cloud offloading, and estimation of cost of execution. Our work satisfies all of them. Table IV compares our work with other models and illustrates the uniqueness of our approach.

VII. Conclusion

We identified the challenges of scheduling large-scale workflows with deadlines on HPC systems instead of solely utilizing costly Cloud systems. A lack of knowledge about future RA and complex data dependencies in workflows limits the ability of WMS/users to make informed decisions on scheduling workflows with respect to cost and deadline. To mitigate these challenges, we developed a novel scheduling

framework to orchestrate large-scale workflow execution in an HPC+Cloud hybrid environment guided by the user's choice of a schedule. To this end, we developed RAP and WSB to generate multiple schedules of workflow execution, which helps users make informed decisions utilizing factors such as cost and deadline. Further, our adaptive scheduler addresses issues such as misprediction of RA and delayed HPC resource allocation by temporarily backing up HPC resources on the Cloud. We use simulations to assess our techniques, as actual execution on Cloud resources would incur prohibitively high costs for our study. The evaluations show that our scheduling framework yields a mean 98%-99.4% rate of task completion, and with a mean 7.11% to 14.75% cost estimation error of the final cost of execution in comparison to a mean of 74.77% to 93.98% and 45.35% to 51.1% task completion rates by Cloud bursting and HPC-only solutions, respectively. Furthermore, our framework saves cost for more than 75% of diverse workflow runs. Combining minimal impacts on production jobs and minimal backup cost of HPC resources on the Cloud, our HPC+Cloud co-scheduling methodology shows good accuracy of our RAP model while keeping cost at bay and considering deadlines.

ACKNOWLEDGEMENT

This work was supported in part by grants NIH-1R01AR072013-01, NSF CISE-2316201, and a subcontract from Lawrence Livermore National Laboratory. This work was performed under the auspices of the U.S. Department of Energy by Lawrence Livermore National Laboratory under Contract DE-AC52-07NA27344. (LLNL-CONF-871565)

REFERENCES

[1] L. Ward et al., "Cloud services enable efficient ai-guided simulation workflows across heterogeneous resources," in *2023 IEEE International Parallel and Distributed Processing Symposium Workshops (IPDPSW)*, 2023, pp. 32–41.

[2] R. F. da Silva et al., "A community roadmap for scientific workflows research and development," in *2021 IEEE Workshop on Workflows in Support of Large-Scale Science (WORKS)*, 2021, pp. 81–90.

[3] R. B. Roy, T. Patel, and D. Tiwari, "Daydream: Executing dynamic scientific workflows on serverless platforms with hot starts," in *SC22: International Conference for High Performance Computing, Networking, Storage and Analysis*, 2022, pp. 1–18.

[4] R. B. Roy, T. Patel, V. Gadepally, and D. Tiwari, "Mashup: making serverless computing useful for hpc workflows via hybrid execution," in *Proceedings of the 27th ACM SIGPLAN Symposium on Principles and Practice of Parallel Programming*, ser. PPoPP '22. New York, NY, USA: Association for Computing Machinery, 2022, p. 46–60. [Online]. Available: https://doi.org/10.1145/3503221.3508407

[5] Z. Shen, X. Han, Z. Xu, and M. Niethammer, "Networks for joint affine and non-parametric image registration," in *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, 2019, pp. 4224–4233.

[6] C. Huang et al., "Dadp: Dynamic abnormality detection and progression for longitudinal knee magnetic resonance images from the osteoarthritis initiative," *Medical Image Analysis*, p. 102343, 2022.

[7] E. Deelman et al., "The future of scientific workflows," *Int. J. High Perform. Comput. Appl.*, vol. 32, no. 1, p. 159–175, Jan. 2018.

[8] B. Yadu et al., "Parsl: Pervasive parallel programming in python," in *28th ACM International Symposium on High-Performance Parallel and Distributed Computing (HPDC)*, 2019. [Online]. Available: https://doi.org/10.1145/3307681.3325400

[9] P. Di Tommaso et al., 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, 2017.

[10] D. J. Milroy et al., "One step closer to converged computing: Achieving scalability with cloud-native hpc," in *2022 IEEE/ACM 4th International Workshop on Containers and New Orchestration Paradigms for Isolated Environments in HPC (CANOPIE-HPC)*, 2022, pp. 57–70.

[11] C. Misale et al., "Towards standard kubernetes scheduling interfaces for converged computing," in *Driving Scientific and Engineering Discoveries Through the Integration of Experiment, Big Data, and Modeling and Simulation*. Cham: Springer International Publishing, 2022, pp. 310–326.

[12] T. Guo, U. Sharma, P. Shenoy, T. Wood, and S. Sahu, "Cost-aware cloud bursting for enterprise applications," vol. 13, no. 3, may 2014. [Online]. Available: https://doi.org/10.1145/2602571

[13] LLNL. (2024, 01). [Online]. Available: https://hpc.llnl.gov/hardware/compute-platforms/lassen

[14] "Top 500 list," Jun. 2023, http://www.top500.org/.

[15] N. Besaw et al., "Cluster system management," *IBM Journal of Research and Development*, vol. 64, no. 3/4, pp. 7:1–7:9, 2020.

[16] LLNL. (2024, 10). [Online]. Available: https://github.com/LLNL/LAST/tree/main

[17] M. Ben Belgacem and B. Chopard, "A hybrid hpc/cloud distributed infrastructure: Coupling ec2 cloud resources with hpc clusters to run large tightly coupled multiscale applications," *Future Generation Computer Systems*, vol. 42, pp. 11–21, 2015. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S0167739X14001514

[18] D. Ahn et al., "Flux: Overcoming scheduling challenges for exascale workflows," 11 2018, pp. 10–19.

[19] AWS. (2024, 01). [Online]. Available: https://aws.amazon.com/ec2/instance-types

[20] M. Mao and M. Humphrey, "A performance study on the vm startup time in the cloud," in *2012 IEEE Fifth International Conference on Cloud Computing*, 2012, pp. 423–430.

[21] IBM. (2024, 01). [Online]. Available: https://www.ibm.com/docs/en/spectrum-lsf/10.1.0

[22] Y. Yu, X. Si, C. Hu, and J. Zhang, "A Review of Recurrent Neural Networks: LSTM Cells and Network Architectures," *Neural Computation*, vol. 31, no. 7, pp. 1235–1270, 07 2019. [Online]. Available: https://doi.org/10.1162/neco_a_01199

[23] S. Developers. (2024, 01). [Online]. Available: https://xgboost.readthedocs.io/en/stable/

[24] S. Team. (2020, 02) Simpy: Discrete-event simulation for python. [Online]. Available: https://pypi.org/project/simpy/

[25] T. Bicer, D. Chiu, and G. Agrawal, "Time and cost sensitive data-intensive computing on hybrid clouds," in *2012 12th IEEE/ACM International Symposium on Cluster, Cloud and Grid Computing (ccgrid 2012)*, 2012, pp. 636–643.

[26] P. A. Ewels et al., "The nf-core framework for community-curated bioinformatics pipelines," *Nature biotechnology*, vol. 38, no. 3, pp. 276–278, 2020.

[27] J. L. Green et al., "Molecular characterization of type i ifn-induced cytotoxicity in bladder cancer cells reveals biomarkers of resistance," *Molecular Therapy-Oncolytics*, vol. 23, pp. 547–559, 2021.

[28] A. N. Toosi, R. O. Sinnott, and R. Buyya, "Resource provisioning for data-intensive applications with deadline constraints on hybrid clouds using aneka," *Future Generation Computer Systems*, vol. 79, pp. 765–775, 2018.

[29] M. D. de Assuncao, A. di Costanzo, and R. Buyya, "Evaluating the cost-benefit of using cloud computing to extend the capacity of clusters," in *Proceedings of the 18th ACM International Symposium on High Performance Distributed Computing*, ser. HPDC '09. New York, NY, USA: Association for Computing Machinery, 2009, p. 141–150. [Online]. Available: https://doi.org/10.1145/1551609.1551635

[30] A. Gupta et al., "Evaluating and improving the performance and scheduling of hpc applications in cloud," *IEEE Transactions on Cloud Computing*, vol. 4, no. 3, pp. 307–321, 2016.

[31] E. Deelman et al., "Pegasus, a workflow management system for science automation," *Future Generation Computer Systems*, vol. 46, pp. 17–35, 2015. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S0167739X14002015

[32] J. Köster and S. Rahmann, "Snakemake—a scalable bioinformatics workflow engine," *Bioinformatics*, vol. 28, no. 19, pp. 2520–2522, 2012.