Automating Edge-to-cloud Workflows for Science: Traversing the Edge-to-cloud Continuum with Pegasus

Ryan Tanaka[†], George Papadimitriou[†], Sai Charan Viswanath[†], Cong Wang[*], Eric Lyons[§], Komal Thareja[*], Chengyi Qu[‡], Alicia Esquivel[‡], Ewa Deelman[†], Anirban Mandal[*], Prasad Calyam[‡], Michael Zink[§]

[†]Information Sciences Institute, University of Southern California, CA, USA
[*]RENCI, University of North Carolina at Chapel Hill, NC, USA
[§]Electrical and Computer Engineering Department, University of Massachusetts at Amherst, MA, USA
[‡]Electrical Engineering and Computer Science Department, University of Missouri Columbia, USA

***Abstract*—In this paper, we describe how we extended the Pegasus Workflow Management System to support edge-to-cloud workflows in an automated fashion. We discuss how Pegasus and HTCondor (its job scheduler) work together to enable this automation. We use HTCondor to form heterogeneous pools of compute resources and Pegasus to plan the workflow onto these resources and manage containers and data movement for executing workflows in hybrid edge-cloud environments. We then show how Pegasus can be used to evaluate the execution of workflows running on edge only, cloud only, and edge-cloud hybrid environments. Using the Chameleon Cloud testbed to set up and configure an edge-cloud environment, we use Pegasus to benchmark the executions of one synthetic workflow and two production workflows: CASA-Wind and the Ocean Observatories Initiative Orcasound workflow, all of which derive their data from edge devices. We present the performance impact on workflow runs of job and data placement strategies employed by Pegasus when configured to run in the above three execution environments. Results show that the synthetic workflow performs best in an edge only environment, while the CASA-Wind and Orcasound workflows see significant improvements in overall makespan when run in a cloud only environment. The results demonstrate that Pegasus can be used to automate edge-to-cloud science workflows and the workflow provenance data collection capabilities of the Pegasus monitoring daemon enable computer scientists to conduct edge-to-cloud research.**

***Index Terms*—Pegasus, edge computing, workflows, workflow management systems, clouds, distributed systems.**

I. Introduction

Over the years, workflows have served as a useful abstraction for computational experiments in fields such as astronomy, physics, biology, and seismology. A workflow, which is usually structured as a directed acyclic graph (DAG), is an abstraction that encapsulates an experiment's computational tasks and the data or control dependencies between each task [1]. Workflows have traditionally been executed using workflow management system (WMS) software on campus clusters, high performance computing (HPC) systems, or in the cloud. The primary role of a WMS is to orchestrate the distributed execution of workflow tasks on available resources based on dependencies present in the workflow. Additional features such as fault tolerance, handling of data movement between tasks, and monitoring capabilities may be offered by a WMS to provide robustness, debugging, and facilitate easy experimentation. The most common types of applications for which WMS have been used include data analysis, parameter sweeps, and instrument data processing [2]. In this paper, we focus on leveraging one WMS in particular, the Pegasus workflow management system (Pegasus WMS or Pegasus) [3], [4].

In recent years, the Internet of Things (IoT) has gained tremendous popularity, leading to an increase in the use of smaller, internet-enabled computing devices located at, or along the path to the network edge. This is in contrast to cloud resources, which are located at and beyond the network core. This geographical diversity of where compute capable devices reside has led to the emergence of edge computing as a new computing paradigm and accordingly, the Edge provides a new execution environment.

Edge computing, which has its roots in content delivery networks, involves moving computations closer to the source of data. These computations take place on edge devices that are often equipped with sensors and serve as the source of data for applications, which enable monitoring, data analytics, and information sharing capabilities [5]. An edge device can be loosely defined as any device along the network path between the source of the data and any compute or storage infrastructure [6]. In this paper, an edge device is defined as a computing device, which is not part of the cloud and one which serves as the source from which data is initially derived.

Edge computing provides four notable advantages over cloud computing: reduced response times, potential for reduced cumulative ingress bandwidth for core cloud systems, the ability to enforce data privacy constraints, and the capacity to mask core cloud outages [7]. Through data-aware job placement and measured consideration of the execution environment for storing intermediate workflow outputs, a WMS can potentially orchestrate workflow runs in such a way that edge computing is best utilized.

Hybrid edge-to-cloud computing environments comprise of compute resources at both locations: the network edge and in

the cloud. Such environments enable applications and systems to utilize the advantages offered by both computing paradigms: low latency, data locality and cost savings at the edge, and scalability, high availability and reliability provided by cloud systems. However, effectively utilizing both computing paradigms within a complex execution environment poses several key challenges in terms of scheduling, resource visibility, mitigating edge constraints, compute and data movement costs, resource provisioning considerations, failure handling, and overall automation of workflow executions that span the edge-to-cloud continuum.

In this work, we address some of the above challenges. We target edge-to-cloud workflows that process potentially large amounts of instrument data generated by networks of sensors outside of a cloud environment. Because of their data processing needs, these workflows can benefit from the computational capacity provided by cloud environments. We describe how we extended the Pegasus WMS to automatically orchestrate edge-to-cloud workflows, and in particular a set of target applications.

The paper makes the following contributions:

- Development of an automated hybrid edge-to-cloud workflow management solution.
- Development of three edge-to-cloud workflows that can be used for experimentation (synthetic [8], CASA-Wind [9], [10], and the Ocean Observatories Initiative (OOI) [11] Orcasound workflow).
- Evaluation of the ability of the system to orchestrate workflows in three different execution environments: cloud only, edge only, and hybrid edge-cloud.
- Performance evaluation of the target applications facilitated by the workflow system, taking into account job placement and intermediate data placement.

The remainder of the paper is structured as follows. Section II discusses challenges in edge-to-cloud hybrid environments, and introduces the Pegasus WMS and HTCondor [12], the job scheduler used by Pegasus. In that section, we also expand upon two key features of HTCondor: *classads* and *matchmaking*, which enable different job placement strategies to be employed by Pegasus, and then present our edge-to-cloud workflow solution. Section III details our workflows and experimental setup on the Chameleon Cloud [13], [14] testbed. Section IV presents a detailed performance evaluation of the experimental runs. Section V covers related works on edge-cloud job scheduling strategies and existing systems that facilitate running jobs on edge-cloud hybrid infrastructures. Section VI concludes the paper.

II. Orchestrating Workflow Runs on Resources from Edge to Cloud

A. Challenges of Edge-to-Cloud Execution

Edge-cloud computing environments make it possible for applications and systems to capitalize on the desirable advantages offered by both computing paradigms: faster response times, data locality, and cost savings at the edge, and scalability, high availability and reliability provided by the cloud. Effectively utilizing both computing paradigms within such a complex execution environment for a given application presents a number of challenges. First, available resources and their states need to be visible in order to make scheduling decisions. Some environments with IoT devices may experience churn due to limited power and network connection. Second, scheduling decisions must be made. When running in the cloud, both compute and data movement costs may need to be considered. Incorporating the edge may involve taking into consideration energy consumption, limited compute capacity, and storage constraints. In addition to scheduling decisions, there may be resource provisioning decisions which can be made to better accommodate varying levels of expected load. Such provisioning can happen at the edge, for example in a cloudlet or on idle edge devices. Third, software systems must be in place to execute computations at both ends and automatically handle failures when they occur. Finally, the ability to capture fine grained performance metrics, or provenance data, is indispensable to optimizing executions on an edge-to-cloud continuum.

B. Edge-to-Cloud Workflow System Design

In order to orchestrate workflows that span edge and cloud resources, we have chosen Pegasus WMS as the basis of our solution. Pegasus has a number of key features that make it a particularly good candidate to provide the automation needed to span the edge to cloud continuum. Most importantly, it has the notion of an abstract workflow. This is a workflow description that is resource independent and captures the workflow at the science level: the codes used for the computations, the data needed and generated by the workflow tasks. Pegasus takes this abstract workflow description and maps it to the available resources, generating the necessary resource-dependent scripts for job submission and adding the necessary data movement between jobs by invoking appropriate data transfer protocols. As a result, Pegasus generates an executable workflow that HTCondor's DAGMan [15] can execute. DAGMan takes the Pegasus-generated scripts and submits them to HTCondor for execution.

Pegasus' architecture that includes the separation between the abstract workflow and the executable workflow enabled it to easily move from Grids to Clouds back in 2008 [16]. This architecture and the reliance on proven and versatile technologies such as HTCondor allowed us now to extend to the edge.

C. HTCondor

HTCondor [12] is a job management and execution system that aggregates computing resources into a single pool, referred to as a "condorpool", of nodes suited for distributed high throughput computing (HTC) workloads. A condorpool may comprise of heterogeneous resources such as desktop machines, cloud resources, and high performance computing (HPC) resources. Resources within a given pool are assigned one or more of the following three roles: submit, execute,

and central manager. Jobs are submitted through a machine assigned with the submit role. The central manager negotiates between compute resources (machines assigned the execute role) and resource requests (jobs submitted through machines assigned the submit role) to ensure that each job is placed on a machine with the required hardware and software configurations (e.g. at least 2 GB of RAM, 512 MB disk space, the latest Java runtime, etc.).

The negotiation process handled by the central manager, referred to as matchmaking [17], serves as the underlying mechanism by which HTCondor can be used to employ job placement strategies. Within a condorpool, resources advertise their characteristics, while queued jobs advertise their requirements. The central manager opportunistically matches jobs with resources as they become available, and once a match is made, a job may run on the matched resource.

In the context of data intensive workloads that are to be executed in an edge-cloud hybrid infrastructure where the majority of data is initially produced at the edge, HTCondor's matchmaking can be used to leverage data locality, minimize data movement between edge and cloud, and improve overall execution time. This can be accomplished by setting up a condorpool incorporating both edge and cloud resources. Next, job requirements for each compute task need to include a target host on which the job is to be run. Workflow execution can be done through a higher level workflow management system such as Pegasus [3], [4].

D. Pegasus WMS

Pegasus is a workflow management system (WMS) which allows researchers to develop, run, monitor, and debug large scale scientific workflows. Researchers can develop workflows, which are represented as directed acyclic graphs (DAG), using the Python Pegasus API. Once developed, a workflow is planned into an executable one, specific to the underlying execution environment on which it will be executed. Pegasus is built upon HTCondor DAGMan, and heavily utilizes it as the execution engine. During Pegasus's workflow planning process, the abstract workflow is translated into a format readable by DAGMan. Additional jobs for staging data in and out of storage resources, file cleanup, and registration jobs (used for tracking files throughout the lifespan of the workflow) are added into the planned workflow by Pegasus. Once the workflow is executing, DAGMan submits jobs from the workflow description to be run according to its topological ordering. Workflow provenance information is collected by Pegasus through the pegasus-monitord daemon [18], allowing users to monitor the workflow execution as it unfolds. In the event that job or file transfer failures occur, automatic retries are attempted. Data movements are handled by pegasus-transfer, a tool that supports a host of transfer protocols like HTTP, S3, GridFTP, SCP, etc.

Pegasus can be used to run jobs on any system compatible with HTCondor. Historically this has been on HPC systems such as campus clusters and DOE or NSF funded national cyberinfrastructures, in the cloud on Amazon Web Services [19] and Google Cloud Platform [20], and on grids like the Open Science Grid [21], [22]. Almost all internet enabled devices at the network edge can be used as an HTCondor execute node, making Pegasus a potential WMS for executing workflows on edge-cloud infrastructures. Up until now, Pegasus has not been used in such an environment.

E. Architecting the Edge-to-Cloud Workflow Solution

HTCondor can run on any edge or cloud resource running Linux, macOS, or Windows. For example, given a set of small devices such as Raspberry Pis, desktop machines, and cloud instances, one can create a hybrid edge-cloud infrastructure.

In order to match jobs specifically with edge or cloud resources, we added an additional attribute, *MACHINE_RESOURCE_TYPE = {edge,cloud}*, to the machine *classad* of each HTCondor worker, which indicated whether or not that resource was an edge or cloud resource. When creating the Pegasus workflows for each of the three execution environments, we indicated which type of resource the job should be matched to using the Python API for describing Pegasus workflows. Internally, HTCondor takes into account this requirement in addition to other job requirements such as required number of CPUs, RAM, disk space, etc.

Data movement operations for each workflow used HTTP, SCP, and local file system operations. These were managed by the *pegasus-transfer* utility. Pegasus-transfer is invoked for each job to handle staging in necessary input data and staging out generated data products. For jobs that are scheduled on locations where input data already resides, *symlinks* are used by *pegasus-transfer* to avoid unnecessary data movements and reduce overall disk usage. One notable advantage of *pegasus-transfer* is that data movement operations are decoupled from the jobs themselves. For each of these workflows, no additional code for the jobs were written to pull from or push to data stores. A change in the locations of initial input files would only require a workflow specific configuration change with Pegasus.

III. Experimental Setup

A. Workflows

Synthetic. The synthetic workflow [8], (see Fig. 1), was developed to represent data aggregation and analytics applications, which run in edge-cloud environments. For such applications, initial input data is derived at the edge from multiple instruments such as a cameras and sensors. Each input goes through preprocessing steps before being aggregated by a single job that outputs the final result. This workflow is modeled after a video analytics application which can search for and trace the route of a missing object or person using surveillance footage aggregated from security cameras and other IoT devices geographically scattered across an urban area [6], [7].

Each of the 32 initial input files in this workflow is 1024 MB. There are 3 levels of jobs. Jobs at levels 1, 2 and 3 are labeled keg_1, keg_2, and keg_merge respectively. Output files

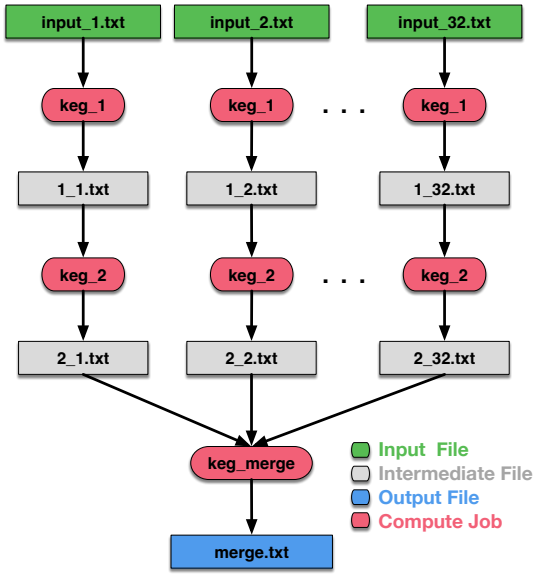

Figure 1. Synthetic Workflow.

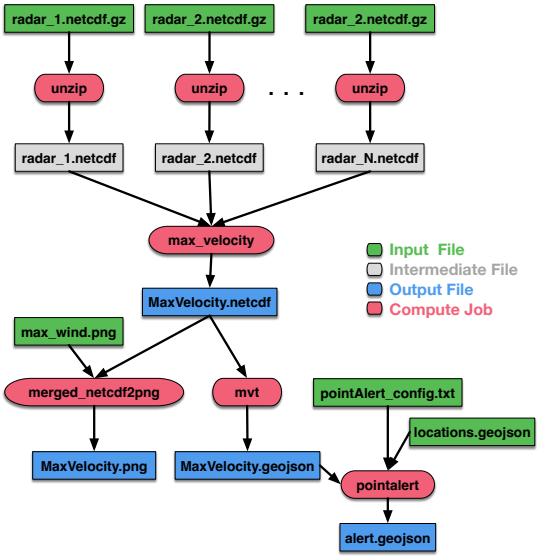

Figure 2. Casa Wind Workflow.

from jobs at level 1, 2, and 3 are 500 MB, 250 MB, and 250 MB for the final output file. Workflow jobs are implemented using the *pegasus-keg* executable, a stand-in for a typical executable which can be configured to simulate computation and IO. In total, there are 86 jobs in this workflow. Job runtimes vary based on their level in the workflow. Jobs at level 2 run 50% faster than jobs at level 1. The final job at level 3, keg_merge, runs 50% slower than jobs at level 1.

CASA-Wind. The CASA Wind workflow [9], [10], as depicted in Fig. 2, is designed to identify areas of maximum observed wind magnitudes from a network of overlapping Doppler weather radars. Seven radars contribute to this product, asynchronously sending NetCDF formatted data files containing a 360 degree azimuthal sweep at a fixed antenna tilt above the horizon. Single radar files are stored in a polarimetric relative coordinate system and must be regridded into a common coordinate system. At a centralized location, the workflow periodically takes any available scans collected over a given interval, and creates a new file in a World Geodetic System 1984 (WGS 84) latitude/longitude projection representing highest winds that have been observed in the time period. Because this information is of importance to users of the system, additional steps are required as part of the workflow to communicate risk. PNG images are created from the resultant grid and sent to a webserver for map overlay and display. Contoured geofences are extracted from the grid, representing polygonal areas where observed winds are exceeding thresholds of importance, for example where winds are severe and likely to cause damage. Finally, comparisons of the geofenced thresholds with GIS information provided by users are performed, for example denoting when the thresholded wind contours overlap or approach areas of critical infrastructure or population centers. These trigger email alerts and app notifications that the CASA system sends out to users for situational awareness. With its basis of sensors deployed in the field, a centralized, asynchronous data ingest process from multiple sources, and series of operations that must be performed in sequence, the CASA wind workflow is appropriate for the evaluation of edge-to-cloud workflow orchestration.

Orcasound Workflow. The Ocean Observatories Initiative (OOI) [23], through a network of sensors, supports critical research in ocean science and marine life. Orcasound [24] is a community driven project that leverages hydrophone sensors deployed in three locations in the state of Washington (San Juan Island, Point Bush, and Port Townsend) in order to study Orca whales in the Pacific Northwest region. Throughout the course of this project, code to process and analyze the hydrophone data has been developed, and machine learning models have been trained to automatically identify the whistles of the Orcas. All of the code is available publicly on GitHub, and the hydrophone data are free to access, stored in an AWS bucket. In this paper, we have developed an Orcasound workflow using Pegasus. This version of the pipeline is based on the GitHub Actions Orcasound workflow [25], and incorporates inference components of the OrcaHello AI notification system [26]. The Orcasound Pegasus workflow (Fig. 3) processes the hydrophone data of one or more sensors in batches for each timestamp, and converts them to a WAV format. Using the WAV output it creates spectrogram images that are stored in the final output location. Furthermore, using the pretrained Orcasound model, the workflow scans the WAV files to identify potential sounds produced by the orcas. These predictions are merged into a JSON file for each sensor, and if data from more than one sensor are being processed, the workflow will create a final merged JSON output for all. In our experiments, we used data from a single hydrophone sensor over the span of a day. The workflow consumed 8641 recordings with a total size of 1.5GBs and median size of 181KBs.

B. Testbed Setup

We used the Chameleon Cloud testbed [13], [14] for our experiments. Chameleon is a large, deeply programmable testbed

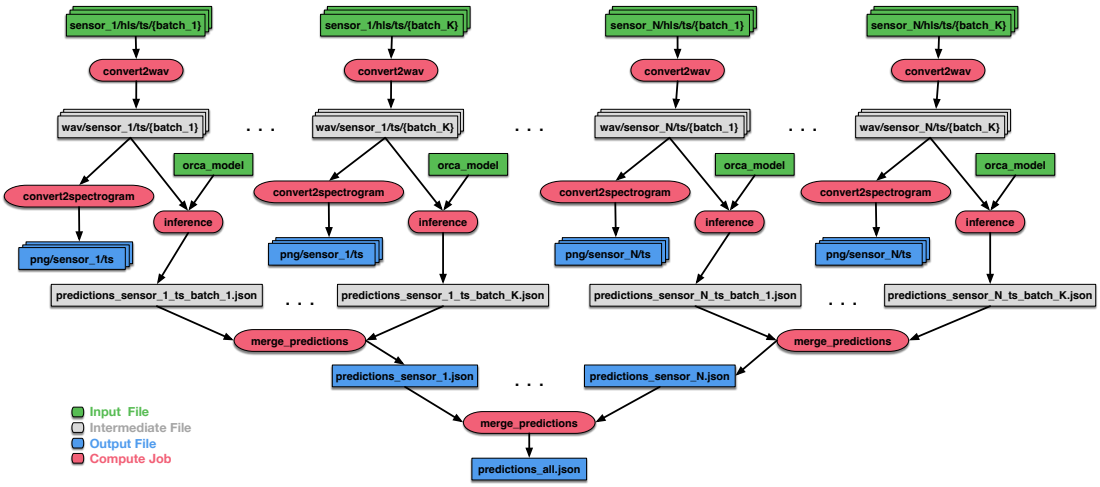

Figure 3. Orcasound Workflow.

designed for systems and networking research. It leverages OpenStack to deploy isolated slices of cloud resources for user experiments. Chameleon provides over 15K cores and 5 PB storage, hosted across two sites, the University of Chicago (UC) and Texas Advanced Computing Center (TACC). Users can provision bare metal compute nodes or VMs with custom system configuration connected to user-controlled OpenFlow switches operating at up to 100 Gbps. In addition, Chameleon networks can be stitched to external resources.

For our experiments, we emulated an edge-to-cloud scenario, and provisioned nodes from both Chameleon sites as shown in Fig. 4. In TACC, we deployed our cloud site, where we assumed we could get unlimited resources. There we created our workflow submit node and two worker nodes (48 cores and 192GB of RAM each), which were connected using a 10Gbps network. In UC, we deployed our edge host (24 cores and 192GB of RAM), which was hosting all the data and was offering compute capability via compute slots that were dynamically provisioned using Docker containers. The data was attached to the compute slots using volumes and the produced outputs were sent back to the submit node using SCP. In case input data was needed for computations by the cloud worker nodes, they were served directly via HTTP. The two sites were connected using a 1Gbps network. To execute our workflow scenarios we used Pegasus v5.0 and we created an HTCondor pool to manage the resources. Additionally, because we wanted to emulate a less powerful machine on the Edge, we used Docker to throttle the CPU usage of the compute slots to 67%. Finally, to maintain a consistent environment across all nodes, we used the more lightweight Singularity containers to create the environment for the workflow jobs.

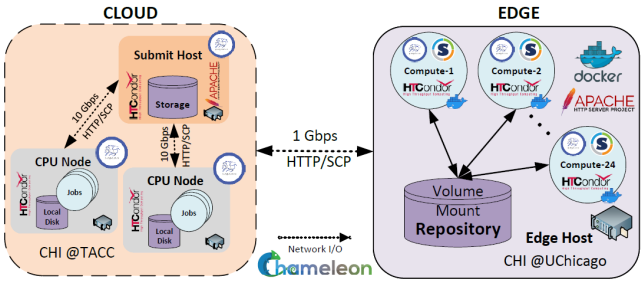

Figure 4. Experimental Setup on Chameleon.

IV. Evaluation

A. Workflow Performance

The synthetic, CASA-Wind, and Orcasound workflows were each run 10 times for each of the three execution scenarios: edge only, edge-cloud, and cloud only. The following metrics were averaged over 10 runs for each workflow: makespan (Fig.5), cumulative job walltime (Fig.6), cumulative job walltime including queuing delays (Fig.8), cumulative time spent transferring data between edge and cloud (Fig.7), and total amount of data transferred between edge and cloud (Fig.9).

Makespan is the wall clock time of the entire workflow run. Cumulative job walltime is the sum of all compute times for each job not including queuing delays and time to stage input and output files. Fig.8 includes queuing delays and data staging times. Cumulative time spent transferring data between edge and cloud is the sum of all time spent transferring data between the edge host (CHI@UChicago) and the cloud (CHI@TACC). Note that for the edge only case, final output files generated from the workflow are transferred back to the cloud. For the cloud only case, all initial input files to the jobs are transferred from the edge to the cloud.

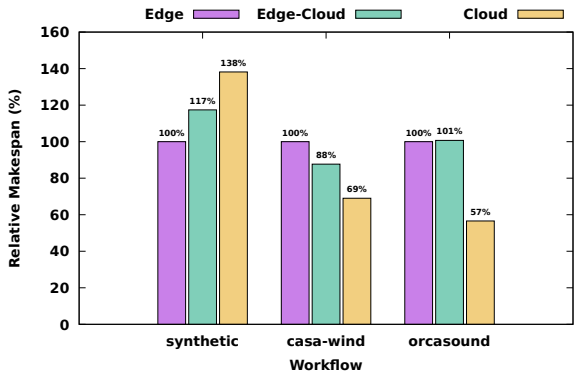

Figure 5. Workflow makespans.

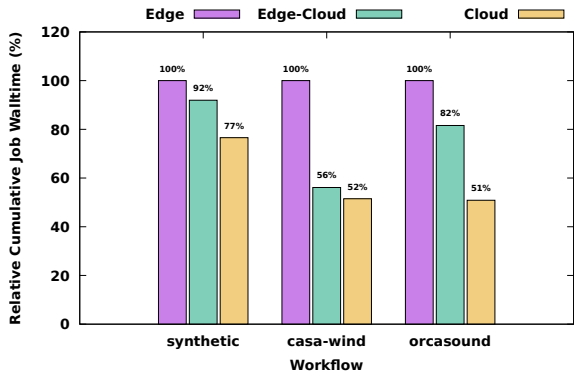

Figure 6. Cumulative job walltime.

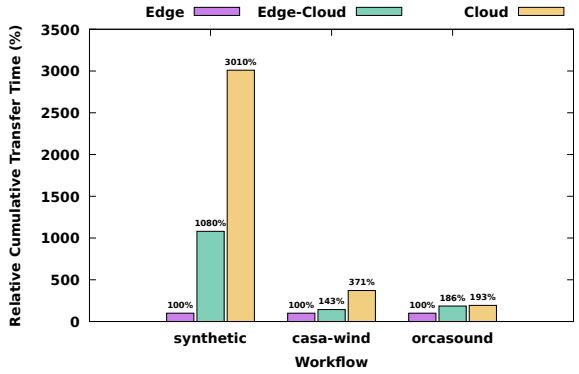

Figure 7. Cumulative time spent on transferring data over WAN.

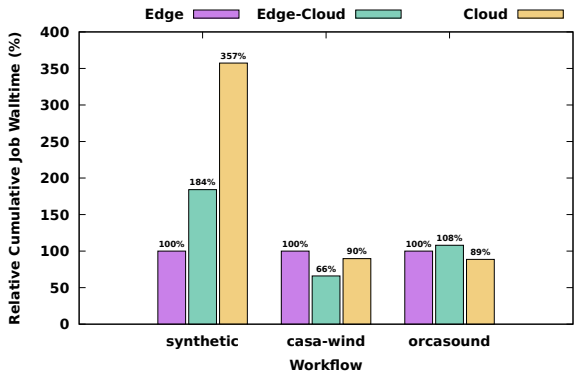

Figure 8. Cumulative job walltime as observed from the submit node.

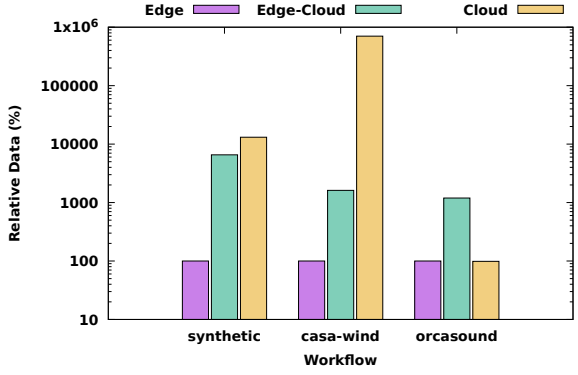

Figure 9. Total data transferred over WAN (yaxis in logscale(2)).

Each metric uses the edge only scenario as a baseline. In our experiments, we analyze each metric on a per workflow basis with respect to the execution scenario, as each workflow has varying characteristics: number of jobs, job runtimes, dependency structure, and amount of data moved.

Synthetic. The synthetic workflow has the shortest makespan when run in an edge only environment, despite having the longest cumulative job walltime among the other execution environments. In the edge only scenario, only 250 MB needs to be transferred between edge and cloud as opposed to the 16 GB and 32 GB when run in edge-cloud and cloud respectively. When run using edge-cloud the makespan is 17% slower, stemming from the almost 6300% increase in the amount of data which needs to be transferred between the edge and the cloud. The workflow was configured for this environment such that all initial workflow jobs, denoted as keg_1 in Fig. 1, were scheduled on the edge host where the initial input files are already stored. All subsequent jobs (keg_2 and keg_merge) were scheduled to run in the cloud requiring all keg_1 outputs to be sent over the WAN.

CASA-Wind. The CASA-Wind workflow exhibited best performance in terms of makespan when run only in the cloud, yielding a 31% improvement over running at the edge. Both edge-cloud and cloud only see a decrease in cumulative job walltime by about 46%, due some or all jobs running in the faster cloud. When factoring in queuing delays and time to stage data (Fig. 8), we see that edge-cloud performs the best with a 34% improvement over edge-only and 24% improvement over cloud only due to the amount of data that needs to be transferred over the slower WAN. 5914.80 MB must be transferred between the edge and the cloud when only cloud compute resources are used in contrast to 13.60 MB when both edge and cloud are utilized.

Orcasound. When run in a cloud only environment, the Orcasound workflow runs about 43% faster compared to running in edge only and edge-cloud environments. Compute jobs performing inference, which make up a significant amount of computation required by the Orcasound workflow, take an average of 36.29 seconds on the cloud compared to 82.26 seconds when run on the edge. Additionally, running in an edge-cloud environment requires roughly 10 times more data to be moved between the edge and the cloud. In terms of cumulative time spent performing data movement over the WAN, there was only a 7% difference between the edge-cloud and cloud only environment runs relative to the edge only environment, indicating that the 43% improvement in makespan was likely due to the 49% decrease in cumulative job walltime when running in the cloud.

B. Discussion

The CASA-Wind and Orcasound workflows both perform the best in regards to makespan when run in a cloud only environment while the synthetic workflow runs the fastest at the edge. When data movement and queuing delays are factored into cumulative job walltimes (Fig. 8) it is evident

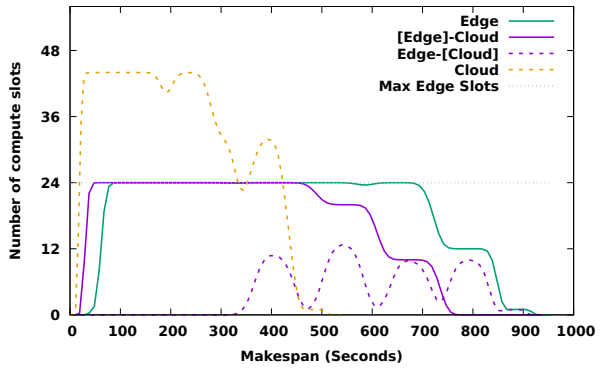

Figure 10. Compute slot utilization of an Orcasound workflow on the Edge and Cloud sites. Edge usage in solid lines and cloud usage in dashed.

that these factors can heavily impact the performance improvements realized by running on faster hardware (illustrated in Fig. 6). Furthermore, the amount of data moved between the edge and the cloud varies vastly depending on the execution environment and job placement. Parallelism must also be considered when identifying the impact of a particular execution environment. Fig. 10 illustrates HTCondor worker utilization over time for the Orcasound workflow when run in each scenario. Because the cloud is assumed to be scalable, the cloud only environment provided maximal parallelism, thus vastly contributing to improved makespan. In contrast, the edge only and edge-cloud environments provided a fixed set of edge resources (24 compute slots in this case), therefore resulting in roughly a 2 times slowdown with respect to the cloud only case. In our experiments, we see that computing strictly at the edge resulted in a minimal amount of data being moved between the two sites while computing in the cloud resulted in the most amount of data being moved between the two sites for two out of the three workflows. Moreover, the slowest compute times are expected to be at the edge versus fastest in the cloud. Lastly, assuming a highly available and scalable cloud, shorter queuing times and greater levels of parallelism can be expected when running on the cloud as opposed to the edge with fixed resources. Together, these characteristics bring into perspective three of the many trade offs which need to be taken into consideration when developing a scheduling heuristic.

Impact. This work demonstrates that Pegasus can enable domain scientists to efficiently automate their workflows in an edge-cloud environment and to utilize the power of edge computing. Our results also show that Pegasus can be used to explore different tradeoffs in managing and executing these edge-to-cloud workloads. Hence, this will also enable new computer systems research in scheduling, monitoring, performance analysis, etc. For example, it will enable research in the area of scheduling, typically performed through modeling and simulation, by allowing the exploration of job placement and scheduling strategies through HTCondor's matchmaking functionality. By using the monitoring and performance data capture features in Pegasus, rich workflow execution traces from experiments run in these hybrid edge-cloud environments can be collected and used by other researchers.

V. Related Works

There have been various software systems and frameworks which have been developed to facilitate application deployment and job execution specifically in edge-cloud environments. KubeEdge [27] is one such framework. Built as an add on to Kubernetes, KubeEdge extends native containerized application orchestration and device management to hosts at the Edge. KubeEdge also facilitates point to point communication between edge nodes through a virtual private network. AWS IoT Greengrass [28] is an edge runtime and cloud service for building, deploying, and managing IoT devices. The edge runtime allows AWS Lambda functions to be run on target edge devices in addition to the cloud. Similarly, Azure IoT Edge [6] also provides an edge runtime that allows application containers to be deployed to edge devices. Steel [29] is a system that automates deployment across edge-cloud environments and provides monitoring capabilities. RACE [30] is an edge-cloud framework that allows cloud-edge applications to continuously correlate or join data from multiple edge devices by means of a novel cost-based optimizer developed to minimize communication time.

In contrast to Pegasus, these frameworks focus on general purpose deployment rather than large scale scientific workflows. Systems like KubeEdge (and Kubernetes) share similarities with HTCondor in terms of managing resources, but differ in that HTCondor can execute jobs as well as facilitate data movements through HTCondor's built in file transfer protocol. Vendor provided IoT services such as AWS Greengrass and Azure IoT Edge provide great functionality out of the box, however do not provide cloud agnostic solutions for developing edge-cloud hybrid compute infrastuctures as HTCondor or KubeEdge does. Furthermore, while these frameworks monitor the state of each resource being used, they do not expose fine-grained performance metrics of individual jobs to the extent that Pegasus can. More recently, GPU monitoring extensions have been developed in pegasus-kickstart to provide researchers with better insights into the performance of workflows incorporating machine learning components [31].

VI. Conclusions and Future Work

With edge computing being increasingly used for data intensive applications, our work presents how we extended Pegasus to automate the execution of scientific workflows in hybrid edge-to-cloud execution environments by leveraging the matchmaking feature of HTCondor. This enables domain scientists to use Pegasus to efficiently execute their workflows in the entire edge-to-cloud continuum.

We also analyzed three representative edge-to-cloud workflows, synthetic, CASA-Wind and Orcasound, using three computing environments, edge only, edge-cloud, and cloud only. Results show that data movement times, placements of jobs, relative computational capabilities at the edge and the cloud, and other factors have significant impact on the overall

performance of our representative edge-to-cloud workflows. Pegasus can now be used to explore many of these tradeoffs and enable new systems research in the areas of scheduling and performance analysis. The workflows developed and workflow execution traces are made publicly available, and can be used by the research community.

Given the inherent tradeoffs between compute time, time spent peforming data movement, and queing delays, in our future work, we will implement new scheduling heuristics in Pegasus such that one or more of the above factors can be optimized. Additionally, factors such as energy consumption and cloud costs were not addressed in our experiments. We will consider energy consumption in future work for systems which utilize edge devices, particularly those with limited power reserves. Finally, we will investigate job placement strategies that take into account costs that can be incurred for using cloud resources and storage services.

ACKNOWLEDGMENT

This work is funded by NSF awards #2018074 and #1664162. Results in this paper were obtained using the Chameleon Cloud testbed supported by NSF. Chameleon support was provided by Jason Anderson. Support for building and containerizing HTCondor was provided by the HTCondor team at the University of Wisconsin-Madison.

REFERENCES

[1] C. S. Liew, M. P. Atkinson, M. Galea, T. F. Ang, P. Martin, and J. I. V. Hemert, "Scientific workflows: Moving across paradigms," *ACM Comput. Surv.*, vol. 49, no. 4, Dec. 2016. [Online]. Available: https://doi.org/10.1145/3012429

[2] R. Ferreira da Silva, R. Filgueira, I. Pietri, M. Jiang, R. Sakellariou, and E. Deelman, "A characterization of workflow management systems for extreme-scale applications," *Future Generation Computer Systems*, vol. 75, pp. 228–238, 2017, funding Acknowledgments: DOE DE-SC0012636, DE-AC52-07NA27344.

[3] E. Deelman, K. Vahi, G. Juve, M. Rynge, S. Callaghan, P. J. Maechling, R. Mayani, W. Chen, R. Ferreira da Silva, M. Livny, and K. Wenger, "Pegasus: a workflow management system for science automation," *Future Generation Computer Systems*, vol. 46, pp. 17–35, 2015, funding Acknowledgements: NSF ACI SDCI 0722019, NSF ACI SI2-SSI 1148515 and NSF OCI-1053575. [Online]. Available: http://pegasus.isi.edu/publications/2014/2014-fgcs-deelman.pdf

[4] E. Deelman, R. Ferreira da Silva, K. Vahi, M. Rynge, R. Mayani, R. Tanaka, W. Whitcup, and M. Livny, "The pegasus workflow management system: Translational computer science in practice," *Journal of Computational Science*, vol. 52, p. 101200, 2021, funding Acknowledgments: NSF 1664162.

[5] I. Lee and K. Lee, "The internet of things (iot): Applications, investments, and challenges for enterprises," *Business Horizons*, vol. 58, no. 4, pp. 431–440, 2015. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S0007681315000373

[6] Weisong Shi et al., "Edge computing: Vision and challenges," *IEEE Internet of Things Journal*, vol. 3, no. 5, pp. 637–646, 2016.

[7] M. Satyanarayanan, "The emergence of edge computing," *Computer*, vol. 50, no. 1, pp. 30–39, 2017.

[8] R. Tanaka and G. Papadimitriou, "Pegasus synthetic edge workflow," Jan. 2022. [Online]. Available: https://doi.org/10.5281/zenodo.5889198

[9] E. Lyons, G. Papadimitriou, C. Wang, K. Thareja, P. Ruth, J. Villalobos, I. Rodero, E. Deelman, M. Zink, and A. Mandal, "Toward a dynamic network-centric distributed cloud platform for scientific workflows: A case study for adaptive weather sensing," in *15th International Conference on eScience (eScience)*, 2019, pp. 67–76, funding Acknowledgments: NSF 1826997.

[10] G. Papadimitriou and S. C. Viswanath, "Casa wind workflow," Jan. 2022. [Online]. Available: https://doi.org/10.5281/zenodo.5889207

[11] G. Papadimitriou, "Orcasound workflow," Jan. 2022. [Online]. Available: https://doi.org/10.5281/zenodo.5889225

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

[13] K. Keahey, J. Anderson, Z. Zhen, P. Riteau, P. Ruth, D. Stanzione, M. Cevik, J. Colleran, H. S. Gunawi, C. Hammock, J. Mambretti, A. Barnes, F. Halbach, A. Rocha, and J. Stubbs, "Lessons learned from the chameleon testbed," in *Proceedings of the 2020 USENIX Annual Technical Conference (USENIX ATC '20)*. USENIX Association, July 2020.

[14] K. Keahey, J. Anderson, M. Sherman, Z. Zhen, M. Powers, I. Brunkan, and A. Cooper, "Chameleon@edge community workshop report," 2021.

[15] "The directed acyclic graph manager," https://research.cs.wisc.edu/htcondor/dagman/.

[16] C. Hoffa, G. Mehta, T. Freeman, E. Deelman, K. Keahey, B. Berriman, and J. Good, "On the use of cloud computing for scientific workflows," in *2008 IEEE Fourth International Conference on eScience*, 2008, pp. 640–645.

[17] R. Raman, M. Livny, and M. Solomon, "Matchmaking: Distributed resource management for high throughput computing," in *Proceedings of the Seventh IEEE International Symposium on High Performance Distributed Computing (HPDC7)*, Chicago, IL, July 1998.

[18] D. Gunter, E. Deelman, T. Samak, C. Brooks, M. Goode, G. Juve, G. Mehta, P. Moraes, F. Silva, M. Swany, and K. Vahi, "Online workflow management and performance analysis with stampede," in *7th International Conference on Network and Service Management (CNSM-2011)*, 2011.

[19] "AmazonWebServices". Amazon web services. [Online]. Available: https://aws.amazon.com/

[20] "Google". Google cloud platform. [Online]. Available: https://cloud.google.com/

[21] R. Pordes, D. Petravick, B. Kramer, D. Olson, M. Livny, A. Roy, P. Avery, K. Blackburn, T. Wenaus, F. Würthwein, I. Foster, R. Gardner, M. Wilde, A. Blatecky, J. McGee, and R. Quick, "The open science grid," in *J. Phys. Conf. Ser.*, ser. 78, vol. 78, 2007, p. 012057.

[22] I. Sfiligoi, D. C. Bradley, B. Holzman, P. Mhashilkar, S. Padhi, and F. Wurthwein, "The pilot way to grid resources using glideinwms," in *2009 WRI World Congress on Computer Science and Information Engineering*, ser. 2, vol. 2, 2009, pp. 428–432.

[23] "The ocean observatories initiative." [Online]. Available: https://oceanobservatories.org/

[24] "The orcasound project." [Online]. Available: https://www.orcasound.net/

[25] "The orcasound action workflow." [Online]. Available: https://github.com/orcasound/orca-action-workflow

[26] "The orcahello ai notification system." [Online]. Available: https://github.com/orcasound/aifororcas-livesystem

[27] Y. Xiong, Y. Sun, L. Xing, and Y. Huang, "Extend cloud to edge with kubeedge," in *2018 IEEE/ACM Symposium on Edge Computing (SEC)*, 2018, pp. 373–377.

[28] "AmazonWebServices". Aws iot greengrass. [Online]. Available: https://aws.amazon.com/greengrass/

[29] S. A. Noghabi, J. Kolb, P. Bodik, and E. Cuervo, "Steel: Simplified development and deployment of edge-cloud applications," in *10th USENIX Workshop on Hot Topics in Cloud Computing (HotCloud 18)*. Boston, MA: USENIX Association, Jul. 2018. [Online]. Available: https://www.usenix.org/conference/hotcloud18/presentation/noghabi

[30] B. Chandramouli, J. Claessens, S. Nath, I. Santos, and W. Zhou, "RACE: real-time applications over cloud-edge," in *Proceedings of the 2012 international conference on Management of Data - SIGMOD '12*. Scottsdale, Arizona, USA: ACM Press, 2012, p. 625. [Online]. Available: http://dl.acm.org/citation.cfm?doid=2213836.2213916

[31] H. Casanova, E. Deelman, S. Gesing, M. Hildreth, S. Hudson, W. Koch, J. Larson, M. A. McDowell, N. Meyers, J.-L. Navarro, G. Papadimitriou, R. Tanaka, I. Taylor, D. Thain, S. M. Wild, R. Filgueira, and R. F. da Silva, "Emerging frameworks for advancing scientific workflows research, development, and education," in *2021 IEEE Workshop on Workflows in Support of Large-Scale Science (WORKS)*, 2021, pp. 74–80.