On-Demand Urgent High Performance Computing Utilizing the Google Cloud Platform

Brandon Posey
Clemson University
bposey@clemson.edu

Adam Deer
Google
adeer@google.com

Wyatt Gorman
Google
wyattgorman@google.com

Vanessa July
Google
vjuly@google.com

Neeraj Kanhere
TrafficVision
neeraj@trafficvision.com

Dan Speck
Burwood Group
dspeck@burwood.com

Boyd Wilson
TrafficVision
boydw@trafficvision.com

Amy Apon
Clemson University
aapon@clemson.edu

***Abstract*—In this paper we describe how high performance computing in the Google Cloud Platform can be utilized in an urgent and emergency situation to process large amounts of traffic data efficiently and on demand. Our approach provides a solution to an urgent need for disaster management using massive data processing and high performance computing. The traffic data used in this demonstration is collected from the public camera systems on Interstate highways in the Southeast United States. Our solution launches a parallel processing system that is the size of a Top 5 supercomputer using the Google Cloud Platform. Results show that the parallel processing system can be launched in a few hours, that it is effective at fast processing of high volume data, and can be de-provisioned in a few hours. We processed 211TB of video utilizing 6,227,593 core hours over the span of about eight hours with an average cost of around $0.008 per vCPU hour, which is less than the cost of many on-premise HPC systems.**

***Index Terms*—HPC in the commercial cloud, Google Cloud Platform, disaster management, massive data processing, traffic management**

I. Introduction and Motivation

Imagine a nightmare in which more than three million people attempt to evacuate a major American city ahead of a devastating hurricane, but they do not leave soon enough. Cars stuck on the jammed highways run out of gasoline, motorists are stranded in a significant heat wave, and more than 100 people die from the combination of severe gridlock and excessive heat. This real life nightmare occurred in 2005 in Houston, Texas, as illustrated in Fig. 1.

In this research we demonstrate how high performance computing (HPC) in the commercial cloud can be utilized in urgent and emergency situations to process large amounts of traffic data efficiently and on demand, helping to provide a solution to a massive and urgent need for disaster management. Our solution launches a parallel processing system that is the size of a Top 5 supercomputer [2] using the Google Cloud Platform. Our approach to an urgent need through the use of HPC in the commercial cloud has wide applicability. For example, cities in the southeast region of the US and Texas have been evacuated regularly due to hurricanes.

This research was supported by NSF Grant #1405767 and by in-kind support from Google Cloud Platform, Burwood Group, and TrafficVision.

Fig. 1: Traffic jam caused by the evacuation from Hurricane Rita in Houston TX, September 23rd, 2005 [1]

A major hurdle to an emergency evacuation is managing traffic so that evacuation happens in timely manner [3]. During an evacuation, traffic can start to back up along the evacuation routes days before the hurricane is scheduled to make landfall, limiting the number of people that can be moved in a timely manner. The ability to create an urgent on-demand HPC environment in the commercial cloud can assist evacuation planners in evaluating when and how to begin the evacuation.

Models of evacuation traffic patterns during hurricanes are valuable tools for adding insight for evacuations. Models can project and describe evacuation scenarios for similar projected paths of the hurricane over multiple days and across the region. These models use data that has been acquired by recording video across dense camera coverage in a region. The video enables vision based traffic analytics that can be executed in batch on an HPC system. Various what-if scenarios can be evaluated, since the video can be processed according to potentially affected geographic areas and executed on demand as the projected path of the oncoming hurricane changes.

The amount of recorded video required for management of the region as the projected hurricane path changes is large. We utilize data from public cameras on the US Interstate system

978-1-7281-5995-9/19/$31.00 ©2019 IEEE
DOI 10.1109/UrgentHPC49580.2019.00008

for our experiment. An evacuation that includes departure, clean up, and return taking up to 10 days and utilizing 8500 cameras across an evacuation region generates 2M hours of recorded video and takes up to 2M hours of compute time to process. The same algorithms that are used to monitor emergency evacuations and give real time status can also utilize models built from previous evacuation studies, adding to the required computational time.

This paper compliments and extends our previous work in which we identify and propose solutions to the challenges encountered when executing a massive scale computational cluster within Amazon Web Services [4]. The natural language application in the previous work was computationally intensive and utilized more than 1.2M vCPUs, each concurrently executing a single threaded process, but the application was not data intensive. The input for the application was a small number of parameters and the output of each process was a few thousand bytes of text data stored in memory until the completion of execution. The application here is highly data intensive. We deploy more than 1.5M vCPUs, each concurrently executing a single threaded process. We utilize the Google Cloud Platform (GCP), processing about 211TB of video in 2,005,170 hours. Each process reads its own input file and writes its own output data, presenting extreme challenges to the storage, network, compute, and HPC scheduling subsystems, as well as to the underlying cloud management infrastructure.

The remainder of this paper is organized as follows: Section II describes related research. Sections III, IV, and V are organized around the software engineering principles followed in this research: Requirements Analysis and Specification, Technology Selection and System Design, and Implementation and Scalability Evaluation, respectively. Section VI reports the Integration, Performance, Efficiency and Cost Evaluation. We conclude the paper in Section VII.

II. Related Research

Three types of related research in the area of urgent computing are discussed in this section, including research focused on building infrastructure and support systems, the generation of simulations, and on creating a generalized definition of urgent computing. Our work is most closely related to the building of infrastructure and support systems, but has aspects of all three types of related research.

The building of infrastructure and support systems tends to focus on the use of existing computational resources and how to utilize a type of priority system in order to allow for urgent computing tasks to be completed in a timely manner. One system that was designed to help with the issue of urgent computing is the Special Priority and Urgent Computing Environment (SPRUCE) [5]. SPRUCE utilizes a token-based authorization system that can be utilized to facilitate and track urgent computation sessions. SPRUCE is designed to work with existing resource providers and to allow the resource providers to have full control regarding the policies around which parts of their resources can be utilized for urgent computation. SPRUCE provides a way for a group of users to launch their jobs with a higher priority than other users of the system, which means that their jobs will be executed first and in a more timely manner. This higher priority can also mean that for certain resources, a users higher priority job may "preempt" or cancel another users jobs. These preemption and priority settings are set by the resource provider. It is the decision of the administrators to determine where and how urgent computational jobs are submitted. Thus, researchers may not have control of when their jobs are executed, which could cause confusion and lead to job processing delays. The authors discuss the feasibility of utilizing commercial cloud resources. However, the paper is focused mainly on existing supercomputing centers with dedicated resources.

The use of commercial cloud is discussed as a potential solution in [6]. This work discusses the tradeoffs between utilizing different types of e-infrastructure, such as HPC or cloud resources, for different types of urgent computation. The paper discusses how most users with urgent computational needs do not have the funding for dedicated resources and how the commercial cloud could be a potential solution, depending upon the frequency and type of workload.

The work done on simulation in the context of urgent computing focuses on helping to predict the damage that could occur before an impending event happens. The events can range from hurricanes, forest fires, tornadoes, or even man-made disasters such as a chemical spill. Examples of these studies include [7], [8] and [9] that have focused on simulating different flooding events in different areas. In [7] the authors utilize an existing resource SX-ACE located at Tohoku University. The goal of the case-study was to provide information about impending tsunamis within 20 minutes of the latest earthquake. The job management system, NQS II, was enhanced to support urgent job prioritization by automatically suspending all other running jobs to allow for the tsunami prediction code to execute. In [8], the authors create a system for monitoring levees within a certain period of time. This system utilizes the Atmosphere platform for provisioning the cloud based resources. The workflows for this system are orchestrated by the HyperFlow workflow management system which executes the jobs and returns the results to the users via the graphical user interface (GUI). In [9], the authors describe a workflow to automate the process of lowering of flood gates in Saint-Petersburg Barrier. They describe how the workflow is implemented in the context of urgent computing and how this helps key decision makers make informed decisions about when and for what period of time the gates need to be closed.

Another set of related work has to do with simulating traffic, [3], [10], in regards to evacuations ahead of an impending event. In [3], the authors discuss the implementation of a generic incident model to look at the traffic incident impacts on evacuation times at large scale. They utilize the Real-Time Evacuation Planning Model (RtePM) to model two different scenarios: a terrorist attack on Washington D.C. and a hurricane at Virginia Beach. These types of simulations are very useful for emergency preparedness and can be enhanced even further with additional information from previous inci-

dents. In [10], the authors discuss a generic traffic management framework for solving large-scale constraint optimization problems. They discuss the implementation of the system in regards to both emergency evacuation and congestion pricing. In implementing this system, they utilized an HPC cluster at the University of Toronto to enable the parallelization of the two cases. This same methodology can also be used for general traffic understanding outside of emergency situations for large cities or regions over multiple day large scale traffic studies.

The last portion of the related research brings together a number of different topics into a widely accepted and generalized definition of urgent computing, as there are currently many different definitions for urgent computing. In [11] they explore the related paradigms and provide a comprehensive general version of the urgent computing definition that clarifies the differences among them. They define an updated definition that clarifies common terms, requirements, pre- and post-computation characteristics, deadline, and cost.

III. Requirements Analysis and Specification

This section examines the requirements and specifications for designing and executing our application urgently at large scale. The characteristics of the application constrain the available options and form the basis for the workflow specification.

A. User Application Definition

Monitoring the different evacuation routes for accidents and tracking vehicles requires specialized software. The software uses as input video from traditional cameras that have been placed along the different public evacuation routes. Our application is a commercial traffic analytics software, TrafficVision [12]–[15]. The software package provides incident and anomaly detection in traffic patterns from existing highway camera infrastructure. The low resolution of the cameras preserves the privacy of the motorists while enabling the monitoring of traffic flow. The software offers the flexibility to process real time video streams and also can execute using batch processing. The software has an AutoLearn feature that is an important feature for the size and scale of this project. The AutoLearn feature handles real-world hardware constraints as well as environmental/operational factors such as camera motion, varying light levels, low or even zero visibility of pavement markings, or video compression artifacts. AutoLearn helps to ensure accurate detection of vehicles at massive scale.

The processing of the video streams is an embarrassingly parallel task and can be scaled to processing very large numbers of video streams simultaneously. Each vCPU or core can process a single 15fps video stream, perform the required detection, and issue alerts to the users. The TrafficVision software is a CPU bound application that does not have a large memory footprint. This important feature means that we do not require large memory on the VMs, which helps to select technologies that will save cost during execution.

TABLE I: Comparison of "Spare Capacity" Instance Functionality as of August 23, 2019

	AWS	**GCP**	**Azure**
Reference	[19]	[20]	[21]
Unlimited Run Time	✓	-	✓
Fixed Discount	-	✓	✓
User Bidding	✓	-	-
Available Within Standard Compute Service	✓	✓	-
Custom Instance Types	-	✓	-
Maximum Discount	90%	80%	80%

B. User Application Requirements

The user requirements to run at large scale while controlling costs are key requirements that guide our selection of a commercial cloud and the selection of instance types within that cloud. The features of commercial cloud are changing rapidly. At the time of this study there were three main commercial cloud choices for large-scale execution: Amazon Web Services (AWS), Google Cloud Platform (GCP), and Microsoft Azure. Each of these providers offers a wide range of different specialized services that can be utilized by end users. All three of these commercial cloud providers offer similar "core" services such as compute, database, and object storage and differentiate themselves on the additional services that they bundle on top of these core services or the way that they present these services. For example, all three commercial cloud providers provide users with a list of specific predefined instance types, which are a combination of CPU, memory, and GPU. Within AWS and Azure, these instance types are defined by the cloud provider and the user is not able to modify them [16], [17]. GCP allows users to specify custom instance types which can be tailored to fit a specific workload, which can help to control costs [18]. GCP custom instance types allow users to specify the number of vCPUs and memory size that are allocated with their instances. This is useful for the TrafficVision workflow, which requires a large number of vCPUs but just a small amount of memory per instance.

All three public cloud providers offer access to their "spare capacity" in the form of a special type of instance. These specialized instance types provide users with discounts off the regular instance price with the caveat that the instance can be preempted, or shut down, at any time with only a short notice. A summary of the differences between each cloud provider's "spare capacity" instance types can be found in Table I.

Within AWS these instances are called *Spot instances*. The pricing is based upon a bid structure and can yield discounts of up to 90% [19]. Within this structure, users set a price that they are willing to pay to run the instances and then if the current Spot price, which is set by AWS and based upon supply and demand, drops below the price set by the user the instances will launch. However, if the Spot price increases above the users set bid price the instance will be shut down with a two minute warning. Spot instances utilize the same instance type structure as the rest of AWS. The Spot price is set on a per-instance basis and can change during the run time of the instances, as there is no time limit for how long a Spot instance can run [19]. This dynamic changing of pricing makes

the overall cost planning more difficult as users can calculate the maximum that they will spend but may not know the actual cost until after the execution due to the changing Spot price.

Within GCP these instance types are known as *preemptible instances*. Their functionality is similar to that of AWS Spot instances in that they can be shut down at any time with only a short notice, but are different in that the price of these instances is static and set by Google. This fixed discount can range up to 80% [20] depending upon the configuration of the instance type. This means that the cost of the instance types does not change during the execution time and users can know the exact discount that they will get. This allows for easier cost planning. GCP preemptible instances also allow users to specify custom instance types, which allows for even more cost savings. Users can request only the number of CPUs and amount of memory required for their application. However, unlike AWSs spot instances that have no time-limit, each preemptible instance can only run for a maximum of 24 hours [20]. As with AWS Spot instances, GCP preemptible instances can be terminated at any point by GCP with only a two minute warning. Our workflow consists of many small independent batch jobs. GCP preemptible instances meet our needs since if a single instance is preempted, other running jobs are not affected.

Azure also has a similar concept with their *Low-Priority instances*, however these instances can only be utilized as part of a VM Scale Sets or Azure Batch. This makes them less flexible than either the AWS or GCP options, which allow users to utilize these discounts in their standard compute services. Minus that difference, Low-Priority instances behave similarly to GCPs preemptible instances. The discount is set between 60%-80% and the instances will only launch when there is spare capacity for those particular instance types [21].

We chose GCP for this application as the capabilities and pricing were a good match for the TrafficVision software requirements. Utilizing the custom instance types that GCP offers allows us to provision the minimum hardware required for the TrafficVision software to run effectively. Combined with the fixed preemptible pricing, these custom instance types also allow us to have a more stable and predictable cost during the execution of the run.

Though preemptible instances work well for the application instances, this is not the case for the rest of the supporting environment such as the HPC scheduler, Login instance, Control instance, and NAT instance that are a part of the HPC software infrastructure in the cloud. These instances need to constantly be available as without access to these instances, the compute instances will be unable to receive new jobs. For HPC infrastructure instances we utilize the standard on-demand instance types so that these instances will not be preempted during the execution. By utilizing a combination of on-demand and preemptible instances, we are able to maintain our execution and scale goals while still being cost efficient.

In addition to utilizing both preemptible and on-demand instance types, we also take advantage of the GCP custom instance types to further help manage our costs. By utilizing a custom instance type that meets the minimum requirements of the TrafficVision software, we can ensure that the entire resource is being utilized and that we are not wasting CPU cycles or paying for memory that we will not utilize.

IV. Technology Selection and System Design

There are a few other technologies that must be selected for executing our TrafficVision based workflow in the cloud, including the overall HPC system design.

A. HPC Lifecycle Technology Selection

Several software infrastructure technologies are required to manage the lifecycle of the HPC environment in the cloud and the jobs running within it. We evaluated an off-the-shelf solution provided by GCP as well as our previous work, Provisioning And Workflow manager (PAW). In previous work, we evaluated alternative resource and workflow management tools for the cloud and developed a solution, the Automated Provision And Workflow Management Tool (PAW) [22].

Recently, GCP published an HPC deployment solution in collaboration with SchedMD that provisions a traditional HPC environment utilizing the Slurm HPC scheduler [23]. This solution creates a Login Node, an NFS filesystem, scheduler instance, and compute instances within either a new or pre-existing Virtual Private Cloud (VPC) network. The solution supports many of the previously mentioned GCP specific features like GCP custom instance types, attachable GPUs, and preemptible instances. It also allows users to create Slurm clusters from Google-provided disk images, which can significantly speed up the launching process of the environment as the user can pre-configure and install all the packages that their workflow requires before launching the environment. Utilizing this method allows the solution to launch a new environment of 5000 instances within 7 minutes [23]. As with all solutions though, there are some drawbacks with this approach.

One of the drawbacks to this approach is that users are only allowed to choose one specific instance type for their Slurm environment. This tends to lead users to over-provision their environments as they have to cater to workloads that utilizes the most resources. This can result in excess costs if the created resources are not properly utilized for all workloads. Another drawback of this solution is that it is a GCP-only solution and the configuration will not transfer easily to another commercial cloud provider. This can create a vendor lock-in effect which can limit users from using another cloud for their workflow in the case of a disaster-caused outage. Interoperability is very important when dealing with disasters and urgent computational needs, as users require the ability to take advantage of any available cloud resources regardless of the provider to get the data processed in a timely manner. Also, although the solution does provide an HPC scheduler to help manage batch jobs there is no native workflow management built-in to the solution. Users are expected to manage and submit the jobs either with a customized script or manually. While this is fine for smaller HPC environments, submitting and tracking hundreds of thousands of jobs can be difficult and time consuming.

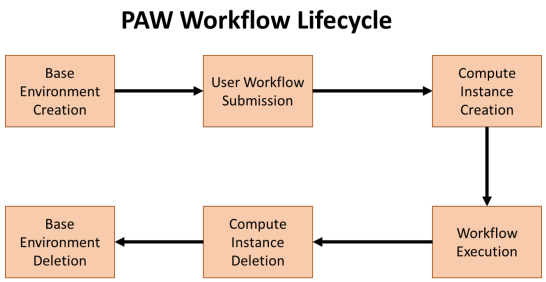

Fig. 2: A simplified view of the typical lifecycle for a PAW User-Defined Workflow.

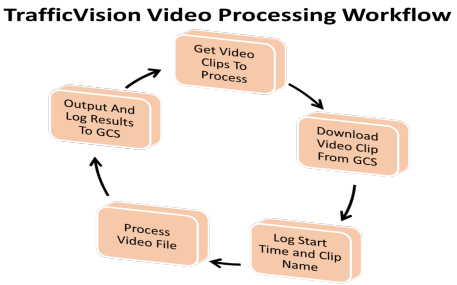

Fig. 3: A simplified view of the TrafficVision workflow utilized to process the video clips during execution.

PAW is designed to be cloud-provider agnostic and to tackle the challenges of managing the lifecycle of both the cloud resources and the scientific workflow. PAW automates the steps of dynamically provisioning a large-scale cluster environment, executing user defined workflows, and de-provisioning the cluster environment when the workflow is finished. PAW is designed to automatically perform all the key management tasks that are typically associated with the management of these large scale environments with a single command.

Along with being cloud-provider agnostic, PAW is also workflow-agnostic and is designed to complement the other off the shelf workflow management tools such as SWIFT [24], Tigres [25], and Pegasus [26]. The PAW implementation utilizes CloudyCluster [27], [28] APIs and the included meta-scheduler CCQ [29] to perform the key management tasks for the HPC environment. One advantage to utilizing PAW is that it can operate across cloud providers and has been tested at a massive scale. In our previous work, we utilized PAW to create a 1.1M vCPU HPC cluster on AWS to perform topic modeling research [22]. By utilizing a solution that has already been tested at a large scale on another cloud provider, we have the added benefit of a solution that can be deployed to multiple cloud providers which unlocks more available resources for urgent processing. We chose to modify and extend PAW to accommodate our applications requirements as we were already familiar with the solution and it provided the most flexibility.

B. System and Workflow Design

The overall design of the software and virtual hardware infrastructure in the cloud resembles the typical structure of a traditional HPC environment and batch processing workflow. The major differences are that instead of all the resources being pre-provisioned and ready to go, the resources are created on demand in the cloud when the workflow is submitted, and then are de-provisioned when execution is complete. Fig. 2 shows an overview of how a typical user workflow is deployed. Resources are provisioned and de-provisioned during the execution of the workflow.

The system design begins with the specification of a single PAW configuration file that contains the specifications for both the HPC environment to be created and the workflow to be processed, as described in [22]. This configuration file is then submitted to PAW which processes the HPC environment configuration and begins to create the specified infrastructure for processing. The infrastructure created at this stage is called the "base environment" and contains the minimum resources required for the HPC environment to operate. These resources can include a Login instance, HPC scheduler, NAT, or a shared filesystem. Depending upon the workflow characteristics, some or all of these resources may be required.

For our TrafficVision workflow, our base environment consists of a Login instance, NAT, and HPC scheduler. No shared filesystem is required, but rather we download input files from Google Cloud Storage (GCS) to local disk storage and write output to GCS directly as a part of the execution script. Fig. 3 illustrates the TrafficVision application workflow.

In order to reach the processing scale needed for managing vehicle evacuation at hurricane proportions, we utilize multiple of these base environments that, in aggregate, provide a massive processing environment. There are two reasons for utilizing multiple base environments. First, multiple environments prevent a single point of failure within an environment from causing the application to fail catastrophically. For example, if the HPC scheduler fails then any submitted jobs would fail to execute, and the whole application would fail. By creating multiple smaller environments, if one environment goes down, the rest of the environments will continue processing. A second reason is that environments may be launched across different geographical regions to handle the processing. If a cloud region itself fails, which could happen in the case of an impending natural disaster, then computation can continue even if some data centers are taken offline.

Once the base environment is in place, PAW can submit the user-defined workflow to the environment for execution and processing. During this phase, PAW reads the workflow configuration from the configuration file and launches the required processing instances for the workflow. Once these instances are in place then the rest of the workflow is submitted to the HPC scheduler just like in a traditional HPC environment. The scheduler handles all of the job execution and management from this point forward and PAW monitors the number of jobs in the scheduler queue. When there are no more jobs in the queue or executing, PAW begins to de-provision the environment so that the user is no longer charged for the resources. The deletion of the environment is an optional step that can be opted out for environments with a shared

filesystem, if the data needs to be preserved.

The PAW framework enables users to quickly and efficiently launch a large scale HPC processing environment in the cloud where they have full control and use of the resources contained within. There is no waiting for queue time which may occur in a locally provisioned resources, even with priority scheduling. The resources are "owned" by the user, who has full priority on their use. By integrating with PAW, workflow definition also provides the advantage of being able to dynamically change the specifications of the hardware for different types of processing within the same environment. This allows for greater control over costs and better overall processing efficiency.

When designing the application we were presented with two different options for accessing the data to be processed. The first option was to receive the live video feeds from a number of cameras and to process that data in real time to showcase the abilities of the system. However, as ordinary citizens we do not have access to the full scale of cameras or network access for the live video feeds that may be used in a real hurricane scenario. The second option was to record a number of publicly available video feeds for a few weeks and store this in Google Cloud Storage (GCS). This way we would ensure that we have access to enough video to emulate the video workload of a real scenario, and that we do not get our network access blocked while attempting to access such a large amount of data in a short period of time. While we ran our application using recorded video, the use of a live-feed just involves swapping out arguments to our processing script. We envision that a combination of these two approaches could be utilized simultaneously during a natural disaster to inform responsible parties during the event as well as for after-the-fact analysis to help better prepare for the next natural disaster.

We implemented our workflow as a user-defined workflow within the PAW tool. The parameters specified in the PAW configuration file are used by PAW to create a batch script to be submitted to the HPC scheduler. This batch script creates a work queue on each of the instances within the HPC environment. The required pre-recorded video clip is copied from GCS to local instance storage for processing. Each video clip is processed as a single "job" on the instance. When that work item (i.e., video clip) has been processed, a new work item is pulled from the queue and processing continues. The video clips are formatted into one hour chunks. Each clip is processed as though it is done in real-time and takes about an hour to process. This ensures that all of our video clips are processed during the execution. In the case of a real-time implementation, the work queue would still be the same but instead of copying a video clip name from GCS and only processing for an hour, a URL pointing to a video stream would be steamed instead and that vCPU would continue to process that same URL video stream until some stopping criteria is reached.

After a video clip has been processed, the TrafficVision software uploads the results about what has been detected in the clip back to GCS where it is stored and can be analyzed with the rest of the video clips. In the real-time situation these results would be reported in real-time to a dashboard that would be monitored by first responders so that they can make decisions based upon what is happening in real-time.

Our application takes advantage of a feature of Cloudy-Cluster that is exposed via PAW that allows computation to begin before all of the instances have been created. This feature allows for the submission of the workflow when the first instance comes up. In our TrafficVision workflow, the first job that is submitted is considered a "parent" job that generates and submits a specified number of batch jobs. Then as the instances launch and are added to the HPC scheduler, they begin to process a job that is waiting in the queue. This eliminates waiting for all of the compute instances to come up before the processing begins. It provides better utilization of the resources and maximum cost efficiency.

V. Implementation and Scalability Evaluation

During our execution we did not experience many unexpected challenges. We had already identified a number of challenges in our previous work and avoided the same pitfalls during this application. However, we did find and identify two challenges that still surprised us, even with our prior experience. One of the unexpected challenges we encountered was with API rate limits per GCP Project, and the second was with the very rapid provisioning of instances on GCP that created some complex interactions with the software infrastructure.

To identify and resolve any potential issues, we developed a testing plan that would help us to evaluate our architecture at a number of different scales. We initially tested environments that were 1% of our total size which equates to about 15,000 vCPUs. After success at the smaller scale, we tested a modest medium-sized scale in which where we launched 5,000 instances per environment. This medium experiment tests the limitations of the HPC scheduler and the underlying provisioning software along with any other new limitations that we may find. We note that a 5,000-instance environment exceeds the size of many on-campus HPC clusters. Achieving success with one single 5,000 instance environment allowed us to move forward with the execution of the multiple 5,000 instance environments that were utilized in our final run.

A. API Limitations Per Project

As with all commercial cloud providers, there are API limits in place to ensure the reliability and usability of the system. The first challenge that we encountered during was a limitation on the number of certain API calls within a single GCP project. A GCP project can be thought of as an overarching "container" for a user's GCP resources. Each GCP project acts like its own GCP account, where each account has its own limits (or quotas as they are called in GCP), Identity and Access Management users and policies, Virtual Private Clouds (VPC), and resources. This concept is especially useful for organizations that have a number of different concurrent tasks that they wish to keep separate but have them all appear under the same account. Resources within a single GCP project can

be launched within any GCP region as one of the features of GCP is that their VPC networks are global and can contain resources running in multiple regions. This is in contrast to how AWS and Azure VPC networks operate as they are tied only having resources from a single region. We originally thought that this would be useful as it would allow us to run multiple environments from a single project in multiple regions. However, this turned out not to be the case.

Within a GCP project, there are a number of quotas (limits), on how many resources and API calls a user can utilize within the project. Some of these quotas are per region but there are some that are global and apply to all the resources within a specific GCP project. This is where we encountered our first challenge. When attempting to spin up multiple environments in multiple regions within the same GCP project, we found that we were hitting a number of GCP Compute Engine quotas with only one or two environments. These quotas included: List requests per 100 seconds, Read requests per 100 seconds, and Heavy-weight read requests per 100 seconds. These are the API calls that we were utilizing to create, monitor, and delete instances during our execution and upon scaling up we quickly hit these quotas which would result in throttling from GCP which would not allow us to get to our target threshold as the requests to launch new instances would fail.

Our solution to this challenge was two-fold. First we attempted the most obvious solution: submit a request to increase the quota to allow for more API calls to be made within the 100 second window. We were granted an increase in our quotas that increased our limit from 2,000 to 6,000 API calls per 100 seconds for all of those quotas. However, we still needed to launch 93,750 instances in order to get to the scale that we wanted and by allowing for 6,000 API calls per 100 seconds it would take 26 minutes of just launching instances to get us to our goal. This also does not take into account any additional calls that may be made during that 26 minute period for monitoring of the instances that have already been launched. Due to these limits being global for the entire project, they would negatively affect each environment that we launched within the same project even if it was in a different region.

In order to work around this, we decided to move to a multiple GCP project setup where we had a single project for each region that we wanted to utilize during the execution of the application. This way we could increase the quotas across multiple projects and spread out the number of API calls being made within a single project. Another additional benefit of splitting out the environments across multiple projects is that during execution we can launch environments in multiple projects simultaneously instead of having to wait a period of time for the quota to clear.

During the execution we utilized multiple projects, but we attempted to launch environments located in different regions within a single project. We found that by doing this we were limiting the number of environments that we could launch at a time. If we attempted to launch too many environments in one project we would hit the quota and the environments would not get to the proper scale. Since we utilized preemptible instances, we were executing with GCP's spare capacity, which can vary greatly from region to region depending on the date and time. This variation means that users have to be highly flexible in where they can launch their instances. If they depend on launching all instance in one specific region, that region may run out of capacity and not allow scaling to scale to their target. By attempting to have environments from multiple regions within the same project, we found that we hurt our ability to utilize all the available capacity as sometimes the most capacity would be in a project where another region had already maxed out the API quotas. Our solution was to limit projects to one region. In this case the user can assure that they will always be able to launch some environments in all regions without having the API quota consumed by another region.

Another area where we ran into these API quota issues was during the de-provisioning of the environments. As with the other cloud providers, GCP is a "pay-as-you-go" service. This means that when it comes time to shutdown the environments, say after the evacuation order has been lifted, we want the environment to shutdown as quickly as possible to avoid paying for additional compute that we do not need. However, the Compute Engine API quotas per 100 seconds still apply to delete calls as well. This is more of an issue than the one encountered with launching instances as now we are paying for compute that we are no longer using as they wait for the 100 second period to pass before new delete calls can be issued.

The solution was to utilize the pre-existing Salt master-minion setup that the underlying CloudyCluster provisioning software already has set up to issue the "shutdown" command to all of the compute instances within the environment. This "shutdown" command will cause the instance to shutdown and enter the "Stopped" state within GCP. Instances that are in the "Stopped" state within GCP are not charged for runtime, only storage which is generally a fraction of the runtime costs. Now having to wait for the 100 second period between making a batch of 6,000 API calls does not matter as much since we are not being charged for the runtime. This allows the environment to take its time deleting the instances and not have to worry about attempting to get them down as soon as possible.

B. Rapid Provisioning Of Instances

The second unexpected challenge that we encountered was with the rapid provisioning of instances within GCP. At first glance, this seems like a positive thing as the faster that instances are provisioned the sooner that they can begin doing work. However, when working at larger scales this quick provisioning means that a large number of instances all launch and begin attempting to communicate to the HPC scheduler at the same time. When these large number of instances start attempting to communicate with the HPC scheduler at the same time, it can cause a distributed denial of service (DDoS) attack on the single HPC scheduler instance.

We first encountered this issue when moving from the small 1% tests to the large 5,000 instance tests. At first everything

looked fine as a couple of instances began coming up, but as more and more instances were launched the scheduler instance quickly became less responsive until we could no longer connect to it via SSH. This was something that we had not anticipated and it took us a little while to figure out that the rapid registration of new nodes coming online was the culprit. Once the issue was identified, we developed a method to stagger the instance launch requests, utilizing the GCP Batch request API. This API allows users to issue up to 5,000 API calls within a single batch request. Utilizing it allowed us to launch instances in more manageable batches with randomized amount of time in between to help to limit the number of instances attempting to register with the HPC scheduler at the same time.

This solution did help to shrink the problem as the instances were now able to register with the scheduler but it was still taking a large amount of time for all of the instances to register and begin work. There were a small number of instances that would register right away and begin work but overall the full registration process was still taking too long. We eventually narrowed down the culprit for the delay to the configuration of Salt and the Slurm HPC scheduler within the CloudyCluster provisioning software utilized by PAW. It turned out that although we had mitigated the simulated DDoS attack on the scheduler, both Salt and Slurm were still having issues trying to add and authenticate the larger number of instances.

To fix this, we looked at two potential solutions. The first solution that we looked into was provided by the GCP SchedMD collaboration. This potential solution involved putting the Slurm configuration file on a shared filesystem that all the instances could mount and read the file from a central location. This would eliminate the need to have Salt push out the configuration file to each instance as was the case with the CloudyCluster software. While this solution does yield an overall faster launch and registration time, it is not scalable as there are known limitations to the scalability of shared filesystems [30]. Also, this would require us to create and maintain a shared filesystem within our environment which would add additional cost and complexity to the environment. For our application all of the data is stored in Google Cloud Storage (GCS) which can be accessed from any of the instances but does not have to be maintained or mounted on the instances to be utilized.

The second solution that we considered was to add additional configuration within both Slurm and Salt to try to limit the number of connections and registrations that can happen simultaneously. This approach does not yield the same drastic increase in launch time, as the Slurm configuration file still has to be pushed across the network to each of the registered instances each time new instances are added, but it does help keep the underlying architecture scalable and eliminates the requirement to create and maintain a shared filesystem within our current architecture. By performing some minor configuration file edits, we were able to drop the instance registration time from around 40 minutes to 20 minutes for all instances to be registered and computing.

VI. Integration and System Evaluation

In this section we discuss execution and evaluation of our final TrafficVision workflow, including a detailed discussion of the technical integration aspects and cost analysis required in order to get the data processed and achieve our goal of 1.5M vCPUs.

A. Performance and Efficiency Evaluation

1) System Performance: As previously mentioned, our workflow ran across a set of HPC environments that were launched within different regions and GCP Projects. In GCP it is possible to have instances from multiple regions running within the same VPC. Although this was not needed for our workflow because the regions we executed in were split across projects, there are other use cases where this would be a large advantage. One of these cases is with the use of preemptible instances. Since preemptible instances are pulled from the spare capacity in the location requested, by requesting in multiple regions users may have access to a larger pool of resources. However, there are other factors such as latency and network egress to other regions that users must consider.

We utilized a total of 4 different GCP projects and 6 different GCP regions for the execution of our workflow. Each of these projects were based within a different GCP region depending upon which regions had the most available spare capacity at the time of execution. We worked with our Google colleagues for guidance about which regions would be the best to execute in and in the end we selected *us-central1*, *europe-west4*, *us-east1*, *asia-east1*, *us-west1*, and *europe-west1*. By spreading out our workflow across these different regions we unlocked additional spare capacity and were not attempting to compete against ourselves for resources within a single region.

The workflow utilized a single GCP custom instance type, custom-16-16384, which has 16 vCPUs and 16GB of memory. This custom instance type allowed us to pay for the minimum amount of memory, which helps to keep our costs down significantly. It also was chosen due to the fact that smaller to medium instance types are less likely to be preempted due to their ability to "fit" into more slots. The more resources that a preemptible instance requests, the more likely it is to be preempted and the more difficult it is to find a location to launch it. In order to reach our goal, we needed to launch a minimum of 93,750 instances. We want our preemptible instances to be able to "fit" in as many places as possible.

The ramp summary of the total number of instances and vCPUs is shown in Fig. 4 A. During the execution of the workflow, at our peak around 9:30pm we had 93,905 instances running across 30 HPC environments totaling 1,502,480 vCPUs executing concurrently. A breakdown of the instance distribution by GCP region and zone is shown in Table II. Although we launched almost 100,000 instances, the overall preemption rate during the run remained low and relatively stable throughout. The highest peaks for preemption were around 300 instances which is less than 1% of the total.

A majority of the instances that we launched remained running until we shut them down at the end of workflow. The

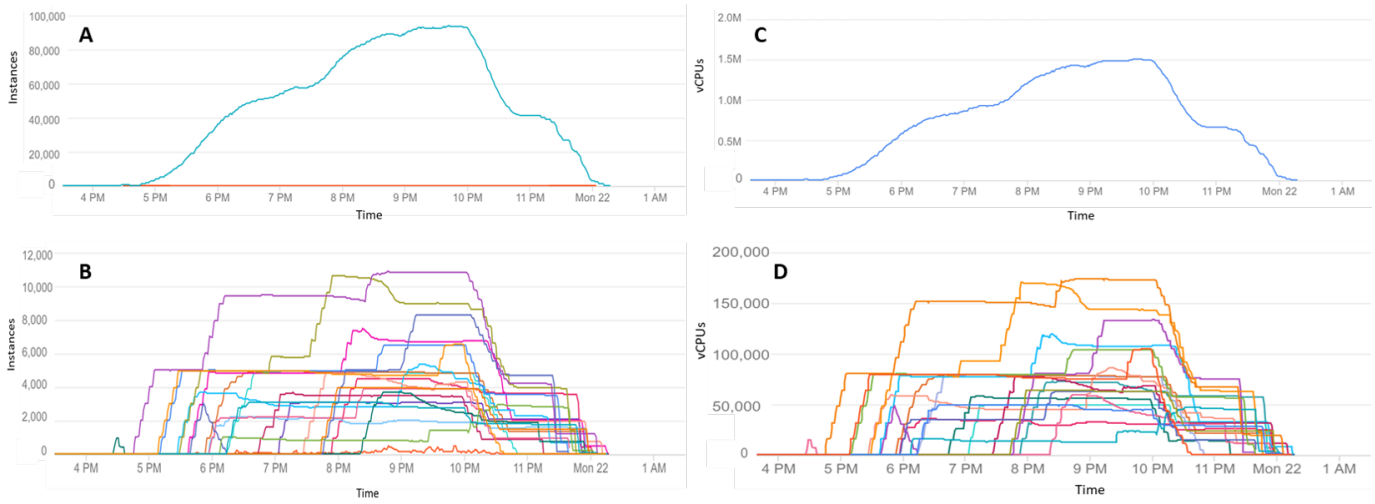

Fig. 4: The overall view of workflow execution. In graphs **B** and **D** each line represents a single GCP Zone, names are not shown for space reasons. **A)** A timeline of the total number of instances in all regions/zones launched during workflow execution. **B)** A timeline of the total number of instances launched per Zone during workflow execution. **C)** A timeline of the total number of vCPUs launched in all regions/zones during workflow execution. **D)** A timeline of the total number of vCPUs per Zone during workflow execution.

TABLE II: Breakdown of the number of instances provisioned per GCP Region/Zone at peak

Region	Zone				
	A	B	C	D	F
us-central1	2,661	3,464	4,155	-	4,904
us-east1	-	6,708	1,915	3,066	-
us-west1	4,853	4,846	4,177	-	-
europe-west1	-	10,803	1,462	8,898	-
europe-west4	6,477	8,324	3,917	-	-
asia-east1	6,509	2,780	3,904	-	-

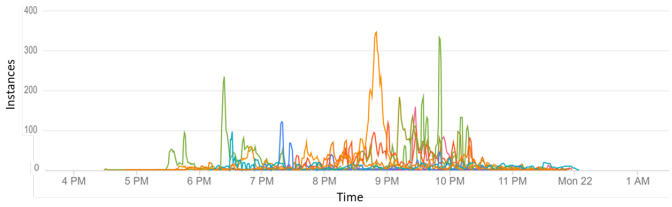

Fig. 5: A timeline of the rate of instance preemption throughout the execution of the workload. Each line represents a single GCP Zone, names are not shown for space reasons.

overall preemption rate throughout the execution is shown in Fig. 5.

There was some API throttling encountered during the execution of the experiment. The effects of this can be seen in the graphs in Fig. 4 around the 6:00-7:30pm mark during creation along with the 10:00-11:00pm mark during deletion. During these periods the number of instances that were being launched or deleted slowed down or leveled out. To move past this issue, we created additional environments and used other projects to limit the number of API calls being sent in each project.

2) GCS Performance: In order to increase the access efficiency of GCS within our workflow, we integrated the concept of GCP *Private Routes* which allow network traffic destined for other GCP services to bypass the NAT process. This eliminates a potential bottleneck in the NAT instance as all the compute instances in HPC environments can communicate directly with GCS without having to all funnel through a single instance. This is especially important at scale as we can have thousands of instances attempting to access GCS at the same time, which would be more than enough to overwhelm a single NAT instance. With the integration of the *Private Routes*, we did not hit any scalability or performance issues with accessing the pre-recorded video clips or uploading the results to GCS during our execution. The throughput that we achieved throughout our execution shows that we hit a maximum rate of 52GiB/s read and 768MiB/s write to GCS from our workflow during the execution. The overall throughput rates to GCS are shown in Fig. 6 graphs A and B.

In the eight hours that our workflow executed, we processed 2,006,170 hours (~211 TB) of video. The rate of processing completion is shown in Fig. 7. We note that there are not a lot of completions within the first hour because our our pre-recorded video is divided into one hour chunks and the results are not uploaded until the entire file has been analyzed. When running in real-time with a standard video stream, the results would be uploaded as they are found.

B. Cost Evaluation

An important consideration when utilizing the cloud for any type of computing is the overall cost and the comparison with the cost of traditional resources. These types of comparisons are complex in nature and can depend upon a large number of variables that are sometimes not well defined. There are certain costs associated with different resources that one may

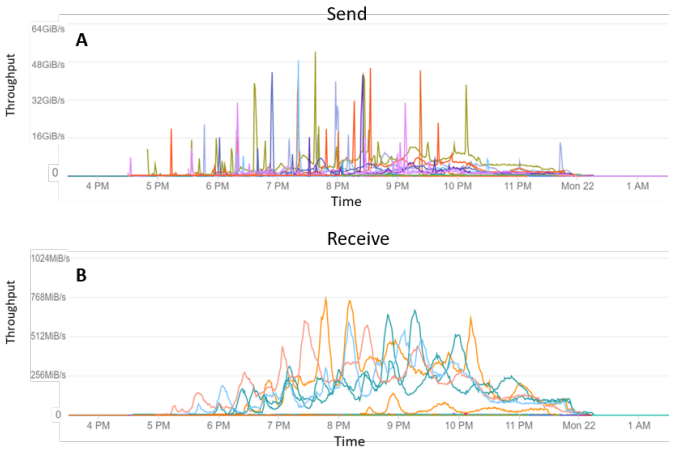

Fig. 6: The overall view of GCS performance throughout workflow execution. In both graphs each line represents a single GCS Bucket, the names are not shown for space reasons. **A)** A timeline of the throughput of data sent to the running instances during the workflow execution. **B)** A timeline of the throughput of the data sent back to GCS after being processed by our workflow.

not consider. Cost estimation at a representative university suggests that the typical local campus computation costs are under US $0.02 per core hour but perhaps not less than $0.01 per core hour. This is regardless of job type (serial or parallel) and includes all of the costs for hardware, software, land, power, cooling, labor, network, etc. However, this cost estimate does not include things such as user-support and research computing facilitation.

By utilizing our custom machine type, we were able to reduce the amount of memory utilized for each instance. At the time of this paper, the GCP standard instance type with 16 vCPUs comes with 64GB of memory and costs $0.1600 USD per hour with preemptible pricing, while our custom instance type costs $0.1264 USD per hour. This gives us a $0.0336 USD savings per instance per hour, which when multiplied with the roughly 94,000 instances required provides savings of $3,158.40 USD per hour over on-demand costs. This cost savings is particularly significant if a workflow needs to execute for an extended period of time. In addition, the preemptible discount for each instance is a set price and will not vary throughout the life of the instance. This allows users to better plan for the cost of execution. Note that the cost savings is not uniform across different regions and needs to be considered when executing across different regions.

We executed our workflow for approximately eight hours and utilized a total of 6,227,593 core hours. The overall cost was $55,044.95 USD. Our overall cost per core hour was approximately $0.008 USD. This number aligns well with the cost estimation for local resources. There are other costs that are associated with execution in the cloud that also need to be taken into consideration as well, such as the cost to store the processed data, download or transfer the processed data, etc. however these costs will vary based upon the workflow and execution model.

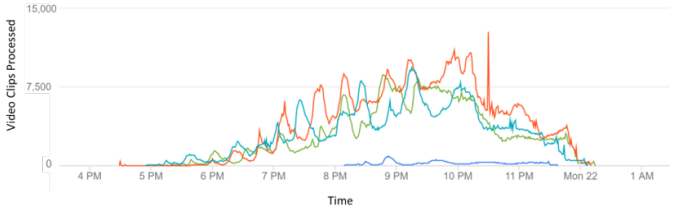

Fig. 7: A timeline view of the video clips analyzed during the execution of our workflow. Each line represents a different GCP Region, the names are not shown for space reasons.

VII. Conclusions

In this paper we present a proof of concept that commercial clouds can handle urgent HPC processing of massive data. We provisioned, utilized, and de-provisioned an HPC cluster with more than 1.5M vCPUs, or the size of a Top 500 supercomputer, in about eight hours. Our application ran in the Google Cloud Platform and demonstrates how urgent HPC can assist with evacuations in the event of an impending natural disaster. We have shown that features now offered by commercial cloud providers and for the right workload, the commercial cloud can provide a cost-effective on-demand infrastructure for urgent HPC and computing in general.

We utilized the cloud-agnostic Automated Provisioning And Workflow Management tool (PAW) to build our HPC environments and execute our workflow. Utilizing PAW and CloudyCluster APIs we showed how the commercial cloud can be utilized. We ran traffic analysis on 2,006,170 hours (~211 TB) of video. Our workflow ran for an average cost of $0.008 per vCPU hour.

We discussed some of the issues that were encountered during the execution of the project and provided solutions within the system and within our workflow. We provided an overview of how the TrafficVision workflow was developed and deployed. For the development of our workflow, we utilized a user-defined workflow within PAW. We also discussed how this workflow can easily be converted from processing pre-recorded video to processing live video with a simple change in script arguments.

We envision that this type of system can be utilized to help aid in streamlining the evacuation processes by allowing first responders or DOTs to observe and simulate traffic in real time during a major event, and also enabling a post-evacuation analysis for improving future responses.

The execution of this massive scale on-demand HPC environment across multiple geographic regions showcases the flexibility and redundancy offered by the commercial cloud along with providing a starting point for others to get started building their own workflows.

One outcome of our work here is a corpus of vehicle roadway trajectories that can be leveraged for traffic research purposes and for increasing the accuracy of these types of simulations. This dataset will be available at https://www.cs.clemson.edu/dice.

REFERENCES

[1] M. Levin, "How hurricane rita anxiety led to the worst gridlock in houston history," Aug 2017. [Online]. Available: https://www.chron.com/news/houston-texas/houston/article/Hurricane-Rita-anxiety-leads-to-hellish-fatal-6521994.php

[2] "Top500 supercomputer sites — june 2019," Jun 2019. [Online]. Available: https://www.top500.org/lists/2019/06/

[3] A. J. Collins, P. Foytik, E. Frydenlund, R. M. Robinson, and C. A. Jordan, "Generic incident model for investigating traffic incident impacts on evacuation times in large-scale emergencies," *Transportation Research Record*, vol. 2459, no. 1, pp. 11–17, 2014.

[4] B. Posey, C. Gropp, B. Wilson, B. McGeachie, S. Padhi, A. Herzog, and A. Apon, "Addressing the challenges of executing a massive computational cluster in the cloud," in *Proceedings of the 18th IEEE/ACM International Symposium on Cluster, Cloud and Grid Computing*, ser. CCGrid '18. Piscataway, NJ, USA: IEEE Press, 2018, pp. 253–262. [Online]. Available: https://doi.org/10.1109/CCGRID.2018.00040

[5] P. Beckman, S. Nadella, N. Trebon, and I. Beschastnikh, "Spruce: A system for supporting urgent high-performance computing," in *Grid-Based Problem Solving Environments*, P. W. Gaffney and J. C. T. Pool, Eds. Boston, MA: Springer US, 2007, pp. 295–311.

[6] S. H. Leong, A. Frank, and D. Kranzlmller, "Leveraging e-infrastructures for urgent computing," *Procedia Computer Science*, vol. 18, pp. 2177 – 2186, 2013, 2013 International Conference on Computational Science. [Online]. Available: http://www.sciencedirect.com/science/article/pii/S1877050913005310

[7] A. Musa, O. Watanabe, H. Matsuoka, H. Hokari, T. Inoue, Y. Murashima, Y. Ohta, R. Hino, S. Koshimura, and H. Kobayashi, "Real-time tsunami inundation forecast system for tsunami disaster prevention and mitigation," *The Journal of Supercomputing*, vol. 74, no. 7, pp. 3093–3113, 2018.

[8] B. Balis, M. Kasztelnik, M. Malawski, P. Nowakowski, B. Wilk, M. Pawlik, and M. Bubak, "Execution management and efficient resource provisioning for flood decision support," *Procedia Computer Science*, vol. 51, pp. 2377–2386, 2015.

[9] A. V. Boukhanovsky and S. V. Ivanov, "Urgent computing for operational storm surge forecasting in saint-petersburg," *Procedia Computer Science*, vol. 9, pp. 1704–1712, 2012.

[10] A. Aboudina, I. Kamel, M. Elshenawy, H. Abdelgawad, and B. Abdulhai, "Harnessing the power of hpc in simulation and optimization of large transportation networks: Spatio-temporal traffic management in the greater toronto area," *IEEE Intelligent Transportation Systems Magazine*, vol. 10, no. 1, pp. 95–106, Spring 2018.

[11] S. H. Leong and D. Kranzlmüller, "Towards a general definition of urgent computing," *Procedia Computer Science*, vol. 51, pp. 2337–2346, 2015.

[12] "Trafficvision." [Online]. Available: http://www.trafficvision.com/

[13] N. K. Kanhere, S. T. Birchfield, and W. A. Sarasua, "Automatic camera calibration using pattern detection for vision-based speed sensing,"

[21] "Pricing - batch: Microsoft azure." [Online]. Available: https://azure.microsoft.com/en-us/pricing/details/batch/

Transportation Research Record, vol. 2086, no. 1, pp. 30–39, 2008. [Online]. Available: https://doi.org/10.3141/2086-04

[14] N. K. Kanhere and S. T. Birchfield, "A taxonomy and analysis of camera calibration methods for traffic monitoring applications," *IEEE Transactions on Intelligent Transportation Systems*, vol. 11, no. 2, pp. 441–452, June 2010.

[15] N. K. Kanhere and S. T. Birchfield, "Real-time incremental segmentation and tracking of vehicles at low camera angles using stable features," *IEEE Transactions on Intelligent Transportation Systems*, vol. 9, no. 1, pp. 148–160, March 2008.

[16] "Amazon ec2 instance types - amazon web services." [Online]. Available: https://aws.amazon.com/ec2/instance-types/

[17] "Virtual machine series." [Online]. Available: https://azure.microsoft.com/en-us/pricing/details/virtual-machines/series/

[18] "Custom machine types - compute engine — google cloud." [Online]. Available: https://cloud.google.com/custom-machine-types/

[19] "Amazon ec2 spot instances." [Online]. Available: https://aws.amazon.com/ec2/spot/

[20] "Preemptible vms - compute instances — google cloud." [Online]. Available: https://cloud.google.com/preemptible-vms/

[22] B. Posey, C. Gropp, B. Wilson, B. McGeachie, S. Padhi, A. Herzog, and A. W. Apon, "Addressing the challenges of executing a massive computational cluster in the cloud," *2018 18th IEEE/ACM International Symposium on Cluster, Cloud and Grid Computing (CCGRID)*, pp. 253–262, 2018.

[23] A. Ma-Weaver and A. Blasius, "Hpc made easy: Announcing new features for slurm on gcp — google cloud blog," Mar 2019. [Online]. Available: https://cloud.google.com/blog/products/compute/hpc-made-easy-announcing-new-features-for-slurm-on-gcp

[24] M. Wilde, M. Hategan, J. M. Wozniak, B. Clifford, D. S. Katz, and I. Foster, "Swift: A language for distributed parallel scripting," *Parallel Computing*, vol. 37, no. 9, pp. 633–652, 2011.

[25] V. Hendrix, J. Fox, D. Ghoshal, and L. Ramakrishnan, "Tigres workflow library: Supporting scientific pipelines on hpc systems," in *2016 16th IEEE/ACM International Symposium on Cluster, Cloud and Grid Computing (CCGrid)*. IEEE, 2016, pp. 146–155.

[26] E. Deelman, K. Vahi, G. Juve, M. Rynge, S. Callaghan, P. J. Maechling, R. Mayani, W. Chen, R. F. Da Silva, M. Livny *et al.*, "Pegasus, a workflow management system for science automation," *Future Generation Computer Systems*, vol. 46, pp. 17–35, 2015.

[27] "Cloudycluster documentation." [Online]. Available: http://docs.cloudycluster.com/

[28] B. Posey, "Dynamic HPC clusters within amazon web services (aws)," *Masters Thesis*, 2016. [Online]. Available: https://tigerprints.clemson.edu/all_theses/2392/

[29] "Ccq and hpc jobs." [Online]. Available: http://docs.gcp.cloudycluster.com/ccq-and-hpc-jobs/

[30] A. W. Apon, P. Wolinski, and G. Amerson, "Sensitivity of cluster file system access to i/o server selection," in *2nd IEEE/ACM International Symposium on Cluster Computing and the Grid (CCGRID'02)*. IEEE, 2002, pp. 183–183.