5-2016

Dynamic HPC Clusters within Amazon Web Services (AWS)

Brandon Posey
Clemson University, bposey@g.clemson.edu

DYNAMIC HPC CLUSTERS WITHIN AMAZON WEB SERVICES (AWS)

A Thesis
Presented to
the Graduate School of
Clemson University

In Partial Fulfillment
of the Requirements for the Degree
Master of Science
Computer Science

---

by
Brandon Posey
May 2016

ABSTRACT

Amazon Web Services (AWS) provides public cloud computing resources and services and is one of the largest cloud computing providers in the world. However, in order to get started using AWS, one must spend many hours overcoming the steep learning curve and terminology associated with AWS. This is especially true for researchers looking to create and utilize a High Performance Computing (HPC) cluster within AWS. This is due to the massive amount of AWS services and AWS resources that must be created and linked together in order to create a fully functional HPC cluster with AWS. The Dynamic AWS HPC Cluster Project aims to help simplify the steps needed to create a fully functional dynamic HPC cluster within AWS. The user simply completes a simple wizard that specifies the details of the HPC cluster that they want: the size and type of the shared filesystem, the type of HPC scheduler, the number of Compute Instances, what IP addresses they want the cluster to be accessible from, and the number of Login/Head Instances required. After all this has been specified, the Dynamic AWS HPC Cluster project makes the required calls to the AWS APIs in order to create all the required AWS resources. After the resources have been created, they are all automatically configured, networked together, and have the usernames and passwords pushed out to all of the cluster instances for SSH login. The user can then run their jobs and when they have no more jobs left to run they can “pause” the cluster, which means they do not pay for compute charges, and then when they have more jobs to run “resume” the cluster and run their jobs. This allows users to only pay for the cluster when they need it which can help save them money.

ACKNOWLEDGMENTS

First, I want to thank my wife Sandra for her understanding and patience throughout this long process. I would also like to thank my entire family for their unwavering support and encouragement as without their love and support none of this would have been possible.

Secondly, I want to thank Boyd Wilson for the constant guidance, support, and suggestions he has given me throughout this project. I also want to thank Dr. Amy Apon and my committee members Dr. Jim Martin and Dr. Brian Malloy for all of the invaluable feedback, editing, and recommendations that they have provided me. I also want to thank Omnibond for providing funding for this project as well as the NSF for supporting this research through NSF Award #1212680.

Thirdly, I want to thank my co-workers and friends Richard Gillette, Aaron Crawford, Zach Muir, Jeff Denton, Nick Watts, and anyone else whom I forgot to mention for their major contributions to this project and for the support and sanity they have given me. Without them and their expertise this project would never have been possible.

Last but not least, I want to thank all of my professors that I have had up to this point, but especially Dr. Paulus Wahjudi, without him I would have never ended up at Clemson University and would have never had this opportunity. I also must especially thank Dr. Barr von Oehsen for initially setting me up with this project and for supporting me throughout my journey at both Marshall University and Clemson University.

TABLE OF CONTENTS

Table of Contents (Continued)

LIST OF TABLES

LIST OF FIGURES

CHAPTER I

INTRODUCTION

Motivation

Many academic researchers utilize some type of computing resources in order to perform their research. The resources utilized by these researchers can range from simply running jobs on their own laptop to running jobs on a High Performance Computing cluster (HPC cluster) if they have access to one. However, not many researchers have turned to cloud resources, like Amazon Web Services (AWS), for performing their research even though the cost of these resources has gone down and the performance of these same resources has drastically increased. This is mainly due to the fact that AWS and other cloud providers have extremely steep learning curves that need to be dealt with before the researcher can even start running their experiments. This can be a major issue for a researcher who is pressed for time and more than likely not an experienced system administrator. They simply do not have the appropriate amount of time required to learn how to use these services and therefore tend to avoid them all together.

This research aims to help to eliminate the steep learning curve associated with these cloud resources and allow researchers to be able to utilize the flexibility, cost efficiency, and performance that the cloud provides. This research will enable researchers who currently do not have access to a HPC cluster to be able to dynamically create their own HPC cluster to utilize for their simulations and other jobs within AWS. This will lead to more researchers having access to HPC cluster resources, which will allow for more research to be accomplished in shorter periods of time leading to more discoveries.

User Requirements

There are a few different user requirements that must be considered in order for this project to be beneficial to the end user. Many of the user requirements can be found in just about any software development project such as security and ease of use. While the others center on customizability and how the user can create an HPC cluster that is well suited to their specific needs. This section gives more specific context on these requirements and explains how they fit into the larger picture of the project.

Limited AWS Knowledge Required

One of the major user requirements for this project centers on the concept that the end user will have very limited AWS knowledge when they start using the project. So in order to accommodate these users, the Dynamic AWS HPC Cluster Project must limit the use of technical AWS terminology used throughout the project. The project also must limit the end user interaction with the actual AWS Console. There are many different things that a user can get overwhelmed with when they first look at and use the AWS Console, this project needs to enable a clear and simple way to launch the application and present some of the information found in the AWS Console in an easier to understand fashion. This will limit the amount of interaction between the end user and the AWS Console keeping the user from getting frustrated and giving up.

Dynamic and Customizable Clusters

Another major requirement for the Dynamic AWS HPC Cluster Project is that it has to be able to create dynamic HPC clusters that can be easily configured and then modified after the initial cluster creation. The initial cluster configuration will be accomplished through the use of Quick Start and Advanced options within the Dynamic AWS HPC Cluster Project. The Quick Start option will provide end users with pre-made cluster options that have a preset number of each type of cluster instances and are ready to go in just a few clicks. The Advanced option will allow the user more control over exactly how many of each cluster instance type get created and even allow users not to create certain types of cluster instances if they so choose. In addition to the different initial configuration options, there are also options to add and delete cluster instances after the initial configuration of the cluster. This allows greater customization and flexibility to the end user and allows them to customize the AWS HPC cluster to their own personal needs.

Adding of Additional Software and Storage

Traditionally HPC clusters have allowed users to add their own customized software to the suite of software that is preinstalled on the HPC cluster. This is due in part to the very stringent requirements that certain research applications have in regards to library versions and dependencies. This process of adding in the additional software is usually accomplished through the use of the module add command in order to “add” the software to the user’s working environment. The Dynamic AWS HPC Cluster Project must be to support these types of software additions as well as support an easy way to install the software on all of the instances in the entire cluster without having to go to each instance and install the software on each of them individually. There needs to be a centralized software repository that all of the instances can access and utilize in order to make it look like the software is natively installed on each instance.

Security

Security is quickly becoming one of the most critical issues for everyone who uses a computer on a daily basis. This is especially true for researchers who are working on confidential research that has not been published yet as researchers do not want their hard work to fall into the hands of someone else that may try and take credit for their work. The Dynamic AWS HPC Cluster Project needs to make sure that the data and computing resources that it creates are secure and conform to modern security standards. In order to accomplish this, there are many different concepts that need to be utilized. One of the first lines of defense is the use of SSL encryption between the public facing cluster instances and the outside world. The user also must be able to specify which IP addresses are allowed to access the actual cluster instances which can drastically help reduce the risk of attack. The Dynamic AWS HPC Cluster Project should also only allow authorized users to access the cluster resources; this includes data access permissions so that the data on the cluster is not visible to any other entity within AWS. This ensures that the researcher’s data is protected inside AWS in the same ways that it is protected on their “on premise” clusters

Developer Requirements

In addition to the user requirements, there are also some developer requirements that must be taken into consideration during this project. In order to ensure that the project can be continued and supported for years to come, many of the developer requirements center on the areas of interoperability, maintainability, and simplicity. These areas are the main concepts that shaped the way that the project was coded. In order to provide more context on these requirements, this section outlines each of the requirements and how they fit in the overall design of the project.

Scheduler Independent

In the HPC space, there are many different options available when it comes to HPC job schedulers. Each one provides different features and often times have completely different syntax for commands and for the structure of the job scripts that are submitted to them. Researchers each have a particular HPC job scheduler that they are used to and their job scripts are specifically formulated for this particular scheduler. Hence it is of the upmost importance that the user is able to choose which type of HPC job scheduler they want when they are launching the cluster. Therefore the Dynamic AWS HPC Cluster Project needs to be developed in a scheduler independent manner. In order for this to be accomplished, the code needs to be structured in a way that lends itself to the easy addition of the different HPC job schedulers. It is also of vital importance to the developers that the Dynamic AWS HPC Cluster Project operates and performs the same general functionality regardless of which scheduler is chosen and created during the cluster launching. By crafting the code to allow for the easy addition of HPC job schedulers later on, the code gains some extra flexibility as well as increasing the overall maintainability of the project.

Scale Out Filesystem

Traditional HPC clusters have a lot of shared storage between the different cluster instances that allow the user's a central repository for data that will allow the data to be accessed on each of the other cluster instances. The users assume this to happen automatically, but there are a few developmental considerations that must be taken into account to ensure that this storage performs like the end user anticipates. The filesystem must be able to scale to large capacities in order to accommodate the data that is produced by the researchers. This scale that is being referred to is on the order of terabytes and sometimes even on the orders of petabytes depending on the type of research being conducted and the data being stored. The filesystem must also be redundant and configured in a high-availability configuration to ensure that the filesystem is always available when the researcher needs it. It also needs to be dynamically and automatically mounted to each of the cluster instances after a cluster has initially launched, paused, or resumed. This process should be transparent to the user and the user should notice minimal performance differences between the scale out/shared filesystem and the local filesystem that is found on the AWS cluster instances themselves.

Web User Interface (UI) Driven

Although many researchers utilize the command line in order to submit their HPC jobs currently, many of them are not completely comfortable with using the command line and prefer to be able to utilize a graphical user interface (GUI) instead. So for these reasons, the Dynamic AWS HPC Cluster Project needs to be written in such a way that the creation of these AWS HPC clusters is driven through a simple and user friendly web user interface. This will allow many researchers who may not exactly have a lot of command line experience to be able to quickly get started with an AWS HPC cluster and provide them with a complete visual representation of the different resources that are being created for them automatically. The visuals will also help them to gain a better rudimentary understanding of just what it takes to create a fully functional AWS HPC cluster.

Cluster Creation Wizard

The Web UI must have a simple and intuitive way for researchers to be able to create AWS HPC Clusters. The cluster Creation Wizard must provide the users multiple options from which to choose ranging from advanced options for technical researchers all the way to simple “templates” for the non-technical user who is just getting started with HPC. It must also keep the AWS terminology usage to a minimum while still obtaining all of the information required in order to create a cluster. It has to provide a simple “One-Click” cluster launching solution, and the cluster created must be data-driven by the information that was entered by the user as they stepped through the Wizard.

Graphical View of Cluster

Another major feature that the Web UI must contain is the ability to show a graphical overview of the cluster during creation, while the cluster is running, and while the cluster is deleting. It must contain an up-to-date representation of a cluster and the resources associated with it while providing the user enough information about the resources that they do not have to go look through the AWS Console in order to find certain pieces of information. It must also contain a way to start, stop, resume, and delete the cluster that is being displayed in order to allow the user to easily manage and control the state of the cluster. Color coded states should show the different statuses of the cluster resources and should change if the state of a resource changes so that the user can be made aware of what is happening.

Web Browser Terminal Access

One of the major advantages of AWS and the cloud is that the resources are available everywhere. This is one aspect that the Dynamic AWS HPC Cluster Project really wanted to capture and embrace because mobility and easy access are features that users want. So in order to facilitate these wants, the Dynamic AWS HPC Cluster Project needs to provide a mobile terminal that can provide SSH access to all the cluster instances straight from any web browser. It must support user authentication via an AWS key pair file or simple username and password combination. The terminal must also be compatible with mobile devices and allow for researchers to be able to have access to their cluster on the go so that they can manage their jobs and check their results from anywhere.

Federated Login

Today everyone has many different accounts for different websites and services that they are trying to manage that it becomes nearly impossible to remember them all. Rather than becoming just another service that requires a username/password combination, the Dynamic AWS HPC Cluster Project utilizes federated login with some of the major websites and identity providers in order to allow the user to take advantage of the accounts that they may already have. The initial list of federated login providers that the Dynamic AWS HPC Cluster Project should support is: InCommon, Google, Twitter, and Facebook. While the other services are self-explanatory, InCommon is a service that many have not heard of. InCommon is an identity provider that is used by many Universities throughout the country for user access management [1]. By enabling federated login though InCommon, many researchers can use their university credentials just like they would on their local campus, thus eliminating the need to remember extra passwords.

CHAPTER II

SIMILAR WORK

CycleCloud

Cycle Computing is a company that provides software that leverages cloud resources to make computation in the cloud productive at any sale [2]. Cycle Computing has a piece of software called CycleCloud that helps users create HPC clusters within the AWS Cloud, which is the very same functionality as the Dynamic AWS HPC Cluster Project aims to provide to users as well. However, even though the end goal is the same for both pieces of software and there are some similarities between the two, the methods used by the two pieces of software are also very different. These similarities and differences between the two pieces of software will be explained in the following paragraphs in order to show just how the Dynamic AWS HPC Cluster Project and CycleCloud accomplish these similar goals while retaining their individuality.

Similarly to the Dynamic AWS HPC Cluster Project CycleCloud utilizes the AWS Virtual Private Cloud (VPC) construct for the launching and security of the cluster. CycleCloud also allows for the integration with a number of different HPC Job Schedulers such as Open Grid Scheduler, HTCondor, Torque, and Cycle Computing's own Jupiter Scheduler that can be created on demand [3] just like schedulers can be created on demand by the Dynamic AWS HPC Cluster project. Both CycleCloud and the Dynamic AWS HPC Cluster Project also each support a wide variety of AWS services ranging from Networking to Storage to Compute and Database services. Both CycleCloud and the Dynamic AWS HPC Cluster Project have a Web UI that can be used to launch and view the current clusters that have been created. While the Web UIs are vastly different, the overall purpose is the same, they exist simply to allow users to be able to start, terminate, and manage their resources from a centralized location.

The first major difference between the Dynamic AWS HPC Cluster Project and CycleCloud is that much of the CycleCloud Cluster specific information is contained within a template file that needs to be modified depending upon the type of cluster that the user is trying to create. This leads to users having to constantly modify the template file each time they want to change the type, number of instances, or AWS features that they want utilize in the cluster [4]. This is the opposite approach taken by the Dynamic AWS HPC Cluster Project where there is no template file that needs to be edited and the user does not need to add entries to the template in order to take advantages of different AWS features such as Enhanced Networking, Placement Groups, and others. Instead of having to modify a template to change the parameters of the cluster all that is needed in order to modify the type of cluster created using the Dynamic AWS HPC Cluster Project is simply just selecting a different option in the Wizard when the user goes to launch a new cluster. Also, many of the more advanced AWS features can be enabled, or are even automatically enabled through the Dynamic AWS HPC Cluster Project without having to do any other configuration. For example, Placement Groups and Enhanced Networking features do not cost anything to use in AWS so if the instances being launched support these features, the Dynamic AWS HPC Cluster Project automatically creates a Placement Group and enables Enhanced Networking on the instances. In contrast in order to utilize Placement Groups in CycleCloud, the user must manually create the Placement Group and then add the resource id of the placement group to the template file for the cluster in order to be able to utilize the Placement Group [5].

Another major difference between CycleCloud and the Dynamic AWS HPC Cluster Project is that CycleCloud requires a license to be bought from Cycle Computing which does not come as an offering on the AWS Marketplace. CycleCloud also requires a special “CycleCloud Server” that does the heavy lifting for the cluster creation [6]. This is similar yet slightly different to the approach taken by the Dynamic AWS HPC Cluster Project. The Dynamic AWS HPC Cluster Project does have a central “Control Instance” that does most of the heavy lifting during the cluster creation; however, it will not require the purchase of an external license or the installation of any extra software to use out of the box.

There are also a few different things that CycleCloud can do that are not yet features of the Dynamic AWS HPC Cluster Project, such as the ability to support multiple cloud providers such as Google Cloud Platform and Microsoft Azure [7]. Also, CycleCloud has the ability to import “Cluster definitions” from Star Cluster, which is another software product that is similar to the Dynamic AWS HPC Cluster Project and will be discussed later on in this section.

CfnCluster

CfnCluster is a framework to deploy and maintain HPC clusters on AWS that was developed by AWS [8]. CfnCluster is an open source project that creates and manages the different parts of the HPC cluster. It is a command line and configuration file based software utility that requires the user to input all the information into a configuration file or into the command line utilities in order to create a cluster. This end goal is yet again the exact same as the Dynamic AWS HPC Cluster Project, but yet again it is the difference in implementation and execution that sets the two pieces of software apart.

CfnCluster and the Dynamic AWS HPC Cluster Project do share some similarities however, both launch the cluster within an AWS VPC, both support multiple schedulers, and both utilize many different AWS Services. Another similarity that CfnCluster and the Dynamic AWS HPC Cluster Project share is the programming languages and libraries that were used to create them. Both CfnCluster and the Dynamic AWS HPC Cluster Project utilize the Python programming language along with the Boto APIs [9]. In both projects, Python is the main glue of the application that relates the different Boto API calls to each other and makes sure that the information from one Boto API call is passed to the other Boto API calls that need to reference the information returned by the previous calls. However, recently the Dynamic AWS HPC Cluster Project has started to move away from Boto and instead has started to utilize Botocore, which leads us to the discussion of the differences between CfnCluster and the Dynamic AWS HPC Cluster Project.

One of the largest differences between CfnCluster and the Dyanmic AWS HPC Cluster Project is the underlying API calls that are used in order to create the actual cluster. CfnCluster utilizes the AWS Cloud Formation Template (CFT) construct in order to create the cluster, while the Dynamic AWS HPC Cluster Project does not utilize these templates but instead uses many different AWS API calls instead. The reason that this is important is because with a CFT, there is limited flexibility available after the template has been created. Once the template has been created there is no way to add new resources to the same “stack” that the other resources were created in, existing resources can be updated but nothing new added [10]. This creates an issue for users who need the ability to add new groups of instances or even another filesystem after the initial creation of the cluster. This brittleness and static nature of these templates is the reason that the Dynamic AWS HPC Cluster Project utilizes APIs directly instead of the CFTs. The Dynamic AWS HPC Cluster Project wants to allow the user to be able to dynamically change the cluster even after creation and this just was not feasible with the CFTs, although during the initial system design stages of the Dynamic AWS HPC Cluster Project, CFTs were considered.

Another major difference that separates CfnCluster and the Dynamic AWS HPC Cluster Project is that CfnCluster is completely command line and configuration file driven. This is similar to the way that CycleCloud operates but instead of having separate template files for each cluster, you can define multiple clusters within the CfnCluster configuration file. This is yet again where a user would have to go in order to enable some of the more complex AWS features as well as to change the makeup of their cluster. This is in contrast to the dynamic and free flowing nature of the Dynamic AWS HPC Cluster Project interface that allows these features to automatically be enabled without the user having to do anything special or set any configuration values in a file. This helps to eliminate some of the learning curve for users who may not be as familiar with AWS as others.

CfnCluster does add a nice and convenient feature that is currently not available in the Dynamic AWS HPC Cluster Project. The support of primitive job scaling features. Basically this means that CfnCluster will create or remove compute instances based upon the number of jobs that have been submitted to the scheduler. This is accomplished through the monitoring of the number of submitted jobs via AWS CloudWatch which call different scaling policies associated with an AWS Autoscaling Group in order to create or remove compute instances based upon the number of jobs in the job queue [11]. This is a feature that is planned to be added into the Dynamic AWS HPC Cluster Project and is described further in Chapter VI.

Star Cluster

Star Cluster is an open source cluster-computing toolkit for AWS EC2 that was developed at the Massachusetts Institute of Technology (MIT) [12]. Star Cluster similar to CfnCluster, is a command line based tool that operates off of a configuration file that is used in order to determine the cluster’s features. This means that tweaking the configuration file is an important part of customizing the cluster that you want to create. Yet again the goal of Star Cluster is quite similar to the Dynamic AWS HPC Cluster Project and although the two projects do share some goals, the differences between the two are the things that stick out the most.

Both Star Cluster and the Dynamic AWS HPC Cluster Project aim to help an end user be able to generate an HPC cluster with AWS and do so by utilizing the Python programming language [12]. Both projects support the starting, stopping, and resuming of clusters as well as the ability to view the running clusters and the different instances that make up each of the clusters from a central location. But besides these few similarities, the implementation of the clusters is drastically different between the two projects.

One of the main differences that users of the two systems will notice right away is the fact that the Dynamic AWS HPC Cluster Project utilizes a Web UI while Star Cluster is a command line utility. Star Cluster does help a little bit more with making the command line user friendly then does CfnCluster, as it provides simple user-readable host names for SSH and also provides simple commands for viewing running clusters and for managing them [13]. However the interface can still be intimidating if you are not used to the terminology or how AWS works in general. Whereas the main goal of the Dynamic AWS HPC Cluster Project is to provide an interface that is much less intimidating for new users who are just trying to get started using an HPC cluster or the command line.

Another difference between the two is that the Dynamic AWS HPC Cluster Project utilizes OrangeFS for shared storage across all of the cluster instances where Star Cluster utilizes the more traditional Network File System (NFS) for its implementation of shared storage [13]. NFS is pretty standard across the HPC industry but the one advantage that OrangeFS has over NFS is the fact that OrangeFS supports parallel I/O operations that can drastically cut down on the time that I/O operations take which can drastically decrease the runtime of I/O intensive jobs.

Another difference between the two is that Star Cluster is a software utility that is installed on a user's local machine. This means that the user's local machine is the one doing all the heavy lifting in order to create all of the AWS resources and means that the utility must be installed in order for the user to be able to use it to access the cluster. This can cause issues for users who have multiple machines and move back and forth between them. It can be extremely difficult to keep the configuration files consistent between machines which can make cluster management difficult. In contrast, the Dynamic AWS HPC Cluster Project Web UI runs on AWS and is accessible from any device that has a web browser on it. This makes cluster administration and access on the go much easier than having to install and configure a separate software package on each of the machines.

Star Cluster also supports many extra functions that are currently not supported by the Dynamic AWS HPC Cluster Project. One of these features is that Star Cluster supports a Python based plug-in system where users can create their own Python based plug-ins that can then be integrated into the Star Cluster utility in order to allow them to perform customized actions [14]. Another one of these features is that Star Cluster provides some command line based wrappers for performing particular AWS tasks that are required in order to be able to use Star Cluster. Another added feature that Star Cluster has is extremely similar to the job autoscaling provided by CfnCluster. Star Cluster supports the expanding and shrinking of the cluster based upon the number of jobs that are in the job queue [15]. This helps to minimize costs while also maximizing productivity.

CHAPTER III

SUPPORTING ENVIRONMENT

Operating Systems

The most critical supporting tool that needs to be considered for any HPC cluster is the operating system that the cluster will run. There are many factors to be considered when choosing an operating system: the package manager, available software, licensing costs, and also the frequency of security patches just to name a few. For the Dynamic AWS HPC Cluster Project two different operating system options were chosen: Red Hat Enterprise Linux and CentOS 7.

Red Hat Enterprise Linux

Red Hat Enterprise Linux (RHEL) is a commercial version of Linux that is based off of Fedora but is highly tuned for stability, security, and performance [16]. Unlike many distributions of Linux, RHEL is not freely available for download and requires a subscription in order to use many of the core features. Usually this would be a large issue when utilizing it on an HPC cluster, but AWS has taken care of this issue for the end user by including the cost of the subscription in the instance cost of the AWS instance. RHEL utilizes the yum package manager and many of the tools that are used by the research community are available for install via yum. This coupled with the large amount of detail paid to security makes RHEL a great distribution of choice for researchers who need a stable and secure operating system and don’t mind the small added cost.

CentOS

CentOS is the other operating system that is fully integrated into the Dynamic AWS HPC Cluster Project. CentOS is based off of RHEL but it is free to modify and does not require a subscription to use. It also utilizes the yum package manager and has a large suite of scientific applications that are available for use. Since there is no additional subscription cost, CentOS is actually cheaper to run on AWS and it also supports an AWS concept of Spot Instances which RHEL does not at this time. Spot Instances are AWS instances that a user can purchase at a lower price than usual with the caveat that the AWS instance can disappear at any time if another user offers to pay more for a Spot Instance. The compatibility with Spot Instances makes CentOS an appealing option for researchers who are looking to try and get the most for their money.

OrangeFS

HPC clusters typically are configured with a large shared filesystem that is quick and is great for storing large data sets and other user data. The Dynamic AWS HPC Cluster Project utilizes the OrangeFS filesystem to achieve this purpose. OrangeFS is a parallel distributed network filesystem that has native support for parallel I/O operations that can be utilized by MPI applications in order to drastically speed up filesystem reads and writes. The Dynamic AWS HPC Cluster Project allows a user to specify exactly what size of a filesystem that they want and then dynamically creates and configures the requested filesystem. How this takes place is described in a later section of this paper.

Once created, the filesystem is automatically mounted on each instance of the cluster so that all the cluster instances have access to the files. The OrangeFS filesystem also supports WebDAV for quick and convenient access to the cluster's filesystem through either a WebDAV client on the user's local machine or through the Web via an authenticated web page located on the Login Instance within the cluster.

Globus

Data transfer into and out of AWS has always been a manual process, usually involving SCP commands or the manual configuration of a data transfer tool on the AWS instance. In order to help facilitate easier and faster transfer of data into and out of AWS, Globus has been integrated into the Dynamic AWS HPC Cluster Project. Globus is a data transfer tool set that utilizes GridFTP to quickly and efficiently transfer data between two Globus endpoints [17]. Globus is widely used both at universities and other corporations around the globe. This means that many of the researchers that the Dynamic AWS HPC Cluster Project targets are used to using Globus to transfer their data to and from their local clusters. This means that by integrating Globus capability into the Dynamic AWS HPC Cluster Project, researchers can utilize the tools that they are already used to using which limits the learning curve required to start moving data in and out of an AWS HPC cluster.

The creation of Globus endpoints within AWS is not a trivial process, especially for a non-technical researcher, so the Dynamic AWS HPC Cluster Project automates the process and dynamically generates Globus endpoints automatically for the user. The knowledge needed by the user is minimal as they only need to enter their Globus account credentials and a name for the Globus endpoint and the Dynamic AWS HPC Cluster Project takes care of the rest. The Globus endpoint is generated within the users account and can easily be activated through a simple OAuth process that utilizes the local cluster account credentials so that no local usernames or passwords are ever sent to Globus's servers.

Schedulers

Schedulers are by far the most critical piece of software on an HPC Cluster. The scheduler takes user submitted jobs and then allocates the resources required to the job, monitors the status of the job, and makes sure that the output of the job ends up where it is supposed to be. There are many different HPC schedulers out currently however most of them work very differently from each other which leads many researchers to prefer one scheduler over another. This is why the Dynamic AWS HPC Cluster Project is coded to support and dynamically configure multiple HPC schedulers based upon the user's personal preference. The user simply selects the type of scheduler they want during the initial cluster creation and then the Dynamic AWS HPC Cluster Project will dynamically configure the scheduler of choice with the cluster. This includes the automatic and dynamic configuration of passwordless SSH to and from the compute instances as well as the dynamic addition of new compute instances to the scheduler if they are created after the initial cluster creation. Currently the only fully supported scheduler is Torque/Maui, but there are many others that are currently being worked on such as Sun Grid Engine (SGE), Slum, and Condor.

Base HPC Software

Another major area of concern for researchers when determining whether or not to use a certain HPC cluster is the number of pre-installed software packages that are available to them. If an HPC cluster is lacking many of the basic scientific software packages and the researcher is not proficient on how to install these packages it will end with the researcher not using that particular HPC cluster and moving to a different one. In order to help mitigate this problem, the Dynamic AWS HPC Cluster Project comes with many of these basic scientific software packages, such as Docker, R, SciPy, NumPy, and many others, pre-installed and configured so that the researcher has many tools at their fingertips as soon as the AWS HPC Cluster spins up. The Dynamic AWS HPC Cluster Project also utilizes the common Module file method of adding software to the user's environment. This is a standard feature on most HPC clusters today that basically allows the user to pick and choose exactly what software and what software version they want in their environment. This allows the researcher to spend less time installing basic packages and more time actually doing their research.

CHAPTER IV

DESIGN AND IMPLEMENTATION

Mapping of Normal HPC Subsystems to Amazon Web Services (AWS)

Traditional High Performance Computing (HPC) clusters typically consist of four different subsystems. These four subsystems are the basic services that are required for the successful operation of a basic HPC cluster: Compute, Networking, Storage, and Access Management. By mapping AWS Services to each of these four core subsystems of HPC the process of deciding which AWS services to use to create the HPC cluster on the AWS cloud begin to take shape.

Compute

The first subsystem of traditional HPC clusters that needed to be mapped to its AWS service counterpart is the subsystem of compute resources. This is the subsystem that is equivalent to all the "racks" of computers/servers within a local campus data center where they are all communicating and working together to perform most of the grunt work that the typical HPC jobs require. This concept maps perfectly to AWS's Elastic Compute Cloud (EC2) service. The EC2 service allows AWS users to "create" many different types of virtual computers/servers all from the same location and utilizing the same protocols and process. The type of virtual hardware inside of these computers/servers varies from an instance with just one or two cores to instances that contain up to thirty six cores [30], this is a great advantage over traditional HPC clusters because usually the compute resources found within a traditional HPC cluster are more static and have only a certain number of hardware options to choose from.

Another advantage that utilizing AWS EC2 resources for use within an HPC cluster is the cost and maintenance factors associated with running a traditional HPC cluster. This issue is discussed more in depth later on in this paper, but one point that needs to be mentioned here is that AWS is a “pay for what you use” service and has ways that you can “pause” the resources in your cluster so that you are no longer paying for the run time of the instances. This is extraordinarily useful due to the fact that the use of an HPC cluster for a researcher tends to be extremely bursty to where they really only need a cluster for a certain period of time and then they will not need it again for a while. EC2 is perfect for this type of scenario as it has all the cost saving functionality needed built right in.

Networking

Without the network infrastructure neither traditional HPC clusters nor an AWS HPC cluster would be possible. Unfortunately unlike the subsystem of compute resources, there is not just a single AWS service that can encompass all the functionality of the traditional HPC cluster networking subsystem. Instead the AWS equivalent is spilt between four different AWS Services: Virtual Private Cloud (VPC), Route53, Placement Groups, and Enhanced Networking. Each of these services provides a different key piece of the overall network functionality that is required by an AWS HPC cluster.

Virtual Private Cloud (VPC) is the biggest piece of the Network subsystem puzzle and can be thought of as basically the equivalent to a campus network. The VPC is where all of the previously mentioned EC2 resources will reside and is effectively what allows the different EC2 resources to be able to talk to each other. This is accomplished through the use of different VPC constructs such as Network ACLs, Routing Tables, VPC Peering Connections, Elastic Network Interfaces (ENIs), Subnets, Network Address Translation (NAT), and Security Groups. Many of these constructs should sound familiar to anyone who has administered a traditional HPC cluster as many of the concepts unsurprisingly map directly to traditional networking concepts.

However the VPC service does not quite cover everything that is needed in the Networking subsystem. One major advantage that a traditional HPC cluster has over an AWS HPC cluster is the ability to have compute resources placed in the same racks to minimize latency between the instances by limiting the number of networking devices the traffic must travel through to reach its destination. This is somewhat of a wildcard in AWS as there is no way to know where the resources will be launched within the AWS datacenters. This is where the Placement Group construct becomes useful as it allows the user to specify that certain instances be “grouped” together within the AWS datacenter to allow for lower latency and for the instances to be able to take full advantage of the underlying 10GB network within AWS [18]. This combined with the Enhanced Networking capabilities, which utilizes the SR-IOV kernel module to increase network throughput and performance on an EC2 instance, can lead to substantial networking performance increases that allow the AWS network to perform at speeds that are closer to what users are used to seeing on a traditional HPC cluster backed by 10GB Ethernet.

Lastly in the Networking subsystem within AWS we have Route53. Route53 is basically AWS's Domain Name System (DNS) solution. Route53 allows the user to create domain name(s) for certain EC2 instances that will allow users to not have to memorize the EC2 resource's IP address but instead they can use a name that is more meaningful.

Storage

The third subsystem of a traditional HPC cluster that needs mapping is the subsystem of Storage. Storage is a critical part of any HPC cluster since many of the HPC jobs require very large data sets as input and sometimes can generate even larger data sets as output. Traditionally all of these data sets have been stored on a fixed size shared or distributed filesystem that each of the compute resources can access. However by utilizing three different AWS services, an AWS HPC cluster provides multiple options for storing and retrieving these data sets. These three services are: Elastic File System (EFS), Elastic Block Storage (EBS), and S3.

Elastic File System (EFS) is the closest AWS equivalent to a local shared filesystem such as NFS. However EFS is much more flexible than just the standard implementation of NFS. NFS is limited to a set size that has to be maintained all the time even if most of the storage space is unused, whereas with EFS the filesystem size grows and shrinks dynamically according to the amount of data that resides on it. This way the user only pays for the storage when they need it and only for exactly as much storage as they need at the time.

Elastic Block Storage Volumes (EBS) are typically referred to as “EBS volumes” and are basically the AWS equivalent of a physical hard drive. An EBS volume is the type of storage that all EC2 compute resources utilize as their boot volumes as well as any other data disks. EBS volumes can be attached and detached from a running EC2 compute resource at will and are utilized for backing implementations of parallel filesystems, such as OrangeFS, on the AWS HPC cluster.

The last Storage service that is utilized by the AWS HPC cluster is S3. S3 is basically AWS’s online accessible object store and is utilized more for longer term storage or to make certain data easily accessible from outside of the AWS Cloud. S3 allows for the creation of a storage “bucket” that is used to store the file objects in. Each object within the S3 bucket is given a certain set of configurable permissions and a unique URL that can be used to download the object from the S3 bucket given that the permissions allow it. Like EFS, the user only pays for the storage space that they are actually using and the storage space dynamically grows and shrinks when a user creates or deletes an object from an S3 bucket.

Access Management

The last of the four subsystems that comprise a traditional HPC cluster is the subsystem of Access Management. This is a vastly important subsystem as this is the subsystem that determines exactly who can and who cannot have access to the HPC cluster resources at any given time. Traditional HPC clusters have some form of authentication for users; usually it corresponds to their own internal authentication system such as Shibboleth, Active Directory, etc. The mapping of the subsystem of

Access Management on an AWS HPC cluster boils down to two AWS services the Identity and Access Management (IAM) and DynamoDB.

The Identity and Access Management (IAM) service is used to control access to the AWS API calls that are used in order to dynamically generate the AWS HPC cluster. IAM makes sure that the users of the AWS HPC cluster cannot escalate privileges on the compute resources of the cluster in order to take over the underlying AWS Account. IAM also limits exactly which API calls certain cluster resources can perform successfully which also helps to reduce the damage that could be done by a rouge user.

DynamoDB is AWS's NoSQL database solution and it is used extensively during the entire process of creating and maintaining an AWS HPC cluster. DynamoDB is where all the user accounts and passwords are encrypted and stored. This allows each of the cluster resources to be able to validate user credentials via DynamoDB if needed. DynamoDB is also used extensively for the tracking of cluster meta-data and other settings that are associated with a particular cluster. This ensures that each compute resource is able to pull all the information it may need about other compute resources directly from DynamoDB at any time.

Design of a Cluster

Now that the mappings of the traditional HPC subsystems to certain AWS services have been discussed, the actual design and implementation of the AWS HPC cluster produced by the Dynamic AWS HPC Project will now be discussed. Within a traditional HPC cluster, there are numerous different types of resources that all have different roles within the cluster. The same is true with an AWS HPC cluster; there are a number of different resources that each play a certain role within the cluster. Some of these resources that are required for an AWS HPC cluster are not required for a traditional HPC cluster while other resources are extremely similar to their traditional counterparts. There are even optional resources for the AWS HPC cluster that a user can pick and choose to customize the cluster to their own liking. Figure 4.1 shows a visual diagram of the architecture and pieces behind the AWS HPC cluster. There are nine different main pieces to an AWS HPC cluster: the Control Instance, the Login Instance, the Scheduler Instance, the Compute Instance(s), the Filesystem Instance(s), the Standby Instance(s), Additional Storage, the Network Components, and the DynamoDB database.

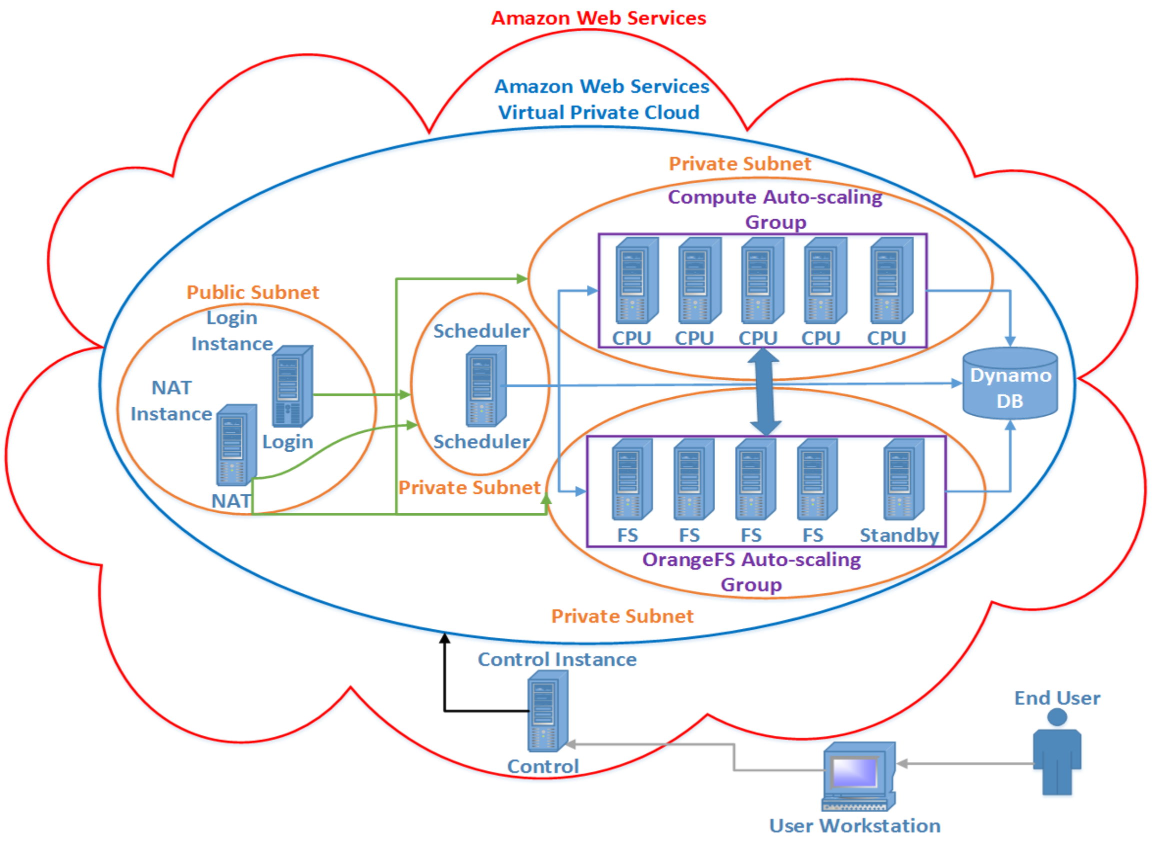


Figure 4.1: Dynamic AWS HPC Cluster Architecture

Control Instance

The Control Instance is the “brain” of an AWS HPC cluster and does not really have an equivalent within the confines of a traditional HPC cluster. The closest thing in regards to a traditional HPC cluster that a Control Instance could be related to is a system administrator. This is due to the fact that the main purpose of the Control Instance is to perform administrative tasks and manage all the other cluster resources. It is the very first resource that is created when launching an AWS HPC cluster and it performs some vital tasks. Through API calls, the Control Instance creates the DynamoDB tables that will store all the information about the AWS HPC cluster and its resources. It also runs the Web UI that allows the easy customization of the AWS HPC cluster being created as well as actually performing the API calls to create all the other resources needed by the user specified cluster configuration. The Control Instance also utilizes the previously mentioned Route53 service in order to obtain a human readable domain name that users can use to access the Web UI instead of having to always go to the IP address. The Control Instance is also the instance that pushes out all of the users and passwords to all of the other cluster resources which allows SSH access and job submission to the cluster.

Login Instance

The Login Instance can be thought of as the equivalent to the head node of a traditional HPC cluster. The head node of a traditional HPC cluster is used as the instance which users generally will login to in order to submit their HPC jobs for processing. The same is true for the Login Instance in an AWS HPC cluster; it is the publicly facing instance that users must login to in order to obtain access into the internal

AWS HPC cluster. This means that it does have a public IP address and it also has a domain name just like the Control Instance so there is no need to memorize IP addresses. The users simply SSH into the Login instance using their username/password combination that is created during the initial Dynamic AWS HPC Cluster Project set up. Once the user has logged into the Login instance, they will be able to access any of the other internal cluster instances, access the shared filesystem(s) if they have configured any, as well as being able to submit jobs to the Scheduler Instance as well.

Scheduler Instance

The Scheduler Instance is a key component to any HPC cluster and is basically the same concept for an AWS HPC cluster as it is for a traditional HPC cluster. There is a little more configuration that has to happen for an AWS HPC cluster scheduler verses the configuration of a traditional HPC scheduler but fortunately the extra configuration is done behind the scenes and the user does not have to worry about it. The Scheduler Instance dynamically configures passwordless SSH between itself and all of the Compute Instances, and even adds the Compute Instances into its scheduling pool as they spin up and their meta-data is added into the DynamoDB database. Since the Scheduler Instance is an internal instance, it is not publically accessible from the Internet; however it can talk out to the Internet to pull down security patches as well as any other files that may be needed for an HPC job. But of course, its main job is to schedule out the HPC jobs that are submitted to it to the Compute Instances so that the results can be delivered in a timely manner. If necessary AWS HPC clusters support multiple Scheduler Instances per cluster, however each Scheduler Instance has to have its own set of Compute Instance(s) they cannot share the same set due to scheduling conflicts and other issues.

Compute Instance(s)

Compute Instance(s) are in both the traditional and AWS HPC clusters the workhorse of the cluster. These are the instances that actually do the grunt work for the HPC jobs and perform all of the calculations and data processing steps required to produce the job output. These Compute Instance(s) get their job assignments from the previously discussed Scheduler Instance and then work on the job until it is completed or otherwise interrupted. These instances automatically and dynamically mount the shared filesystem(s) that were created with the AWS HPC cluster so that they can all access the files stored on the shared filesystem(s). They are also configured to have passwordless SSH between themselves as well as between the Scheduler Instance since this is a requirement of most HPC job schedulers. This is automatically configured during the launching of the instances and there is no user intervention needed in order for it to work. If it is found that after launch more Compute Instance(s) are needed in order to successfully complete all of the HPC jobs, more Compute Instance(s) can be added to the AWS HPC cluster after the initial creation and they will be dynamically configured and added to the Scheduler Instance in the same fashion as the original Compute Instance(s) were added.

Filesystem Instance(s)

The Filesystem Instance(s) are a feature that is most closely equivalent to the shared filesystem on a traditional HPC cluster. Traditionally there is some type of shared storage across all the HPC cluster resources that allows each cluster resource to access the same files. The AWS HPC cluster utilizes the OrangeFS Parallel filesystem running across multiple AWS EC2 instances in order to provide a high-availability, scalable, and fast shared storage solution. The entire process is accomplished through the use of EC2 Compute Instances, EBS Volumes, and Elastic Network Interfaces (ENIs). The entire storage piece of the filesystem is striped across multiple EBS volumes attached to a specified number of EC2 Compute Instances that have an attached ENI with a particular static IP address associated with it. This ensures that even if the EC2 Compute Instance was to die, the EBS volumes and ENI would still remain which means that the data on the filesystem is safe. This failure protection process will be discussed more in depth in the Standby Instance(s) subsection.

This filesystem is also dynamically mounted on all of the internal AWS HPC cluster instances and is even accessible through any web browser by going to the domain name of the Login_Instance/filesystem_name and logging in using your username and password (standard UNIX permissions still apply). Utilizing the OrangeFS set up also allows for the easy upload and download of data into the AWS HPC cluster via a WebDAV Client which allows a user to mount the AWS HPC cluster OrangeFS filesystem on their local machine and transfer data to and from it at will.

Standby Instance(s)

The Standby Instance(s) are the instance(s) that allow the Filesystem Instance(s) to operate in a high-availability configuration. These Standby Instance(s) are similar to having a “hot spare” in a traditional HPC cluster that is ready to go online at any time should one of the computers fail. These Standby Instance(s) are launched with the Filesystem Instances and their job is to monitor the OrangeFS service on each of the running Filesystem Instances and if it is ever determined that one of the Filesystem Instances has failed, the Standby Instance will then “take over” the failed Filesystem Instance’s identity and allow the OrangeFS filesystem to keep operating like nothing happened. This entire process is accomplished through the use of EBS volumes and ENIs. When it is determined that a Filesystem Instance has failed, the Standby Instance that detected the failure will issue API calls that will detach the EBS volumes and the ENI from the failed Filesystem Instance. Then after the ENI and EBS volumes have been detached from the failed instance, the Standby Instance will then re-attach the ENI and EBS volumes to itself through API calls and then start the OrangeFS service on itself thus assuming the identity of the old Filesystem Instance since the IP address and data on the EBS volumes will not have changed. This allows the filesystem to keep operating like nothing has happened since all the information the other Filesystem Instances care about has not changed. Once this process has completed the Standby Instance terminates the old failed Filesystem Instance and spawns a new Standby Instance to take its place.

This process works since the ENI is assigned a static IP address and all the data that was contained on the old Filesystem Instance is contained on the EBS volumes that are now attached to the Standby Instance, the other Filesystem Instances do not know that anything has changed since they are still communicating with the same IP address. This prevents having to reconfigure the OrangeFS filesystem if a Filesystem Instance dies and also provides a way to mitigate the damage caused by an inaccessible filesystem.

Additional Storage

An AWS HPC cluster also has a few other additional storage options that can complement or replace the previously discussed Filesystem and Standby Instance(s). These additional storage solutions are Elastic File System (EFS) and S3. These two AWS services have already been briefly discussed before, and at the basic level are just more places to store data. In the case of EFS, during the initial creation of an AWS HPC cluster the user has the option to create an instance of EFS and have it dynamically and automatically mounted across the entire cluster. This then allows shared files and a shared filesystem for the cluster while not having the overhead of running Filesystem and Standby Instance(s). During the initial creation, the user is also presented with an option to create an S3 Bucket that can be used for data storage. However this S3 Bucket is not dynamically mounted on all the cluster instances due to performance issues. S3 is meant more for infrequent data access as opposed to frequent reads and writes [19] and hence has some issues when trying to mount it as a native filesystem.

Adding Additional Software

The software that is installed upon an HPC cluster can make or break how useful the HPC cluster is for certain types of research. Typically HPC clusters allow users to add different software into their environment by utilizing the module add command which then loads the software that is installed in a different location to be added to the users environment and used like it was installed locally. The Dynamic AWS HPC Cluster Project enables this same concept from traditional HPC clusters by utilizing AWS's EFS. An EFS instance is created and software can be installed onto the filesystem and then it can be loaded via the module add command onto all of the instances within the cluster. This allows for the software to seem “locally” installed but without the hassle of having to go install the software one each individual machine which can quickly become a pain with hundreds of instances.

Network Components

A functioning and correctly configured network is a key piece of any HPC cluster, traditional or AWS based since without a network there is no cluster at all. The networking aspect of a traditional HPC cluster is usually all handled by a network administrator who configures and sets up the network. However for an AWS HPC cluster, it is actually the Control Instance that sets up all the networking components and configures the network. The Control Instance first creates a Virtual Private Cloud (VPC) which is a “private internal network” for the AWS HPC cluster to operate within. This makes sure that all the AWS HPC cluster instances are on the same internal network and are sectioned off from the rest of the user’s AWS account for security reasons.

Once this VPC has been created, the Control Instance then performs dynamic subnet calculation in order to divide the VPC’s address space into different subnets that are generally split across the type of instance. For example the Compute Instance(s), Filesystem and Standby Instance(s), and Scheduler(s) each have their own dynamically generated subnet created for them when they are first launched. This comes with the exception of the public subnet. The public subnet is reserved for the AWS HPC cluster resources that have public IP addresses such as the previously mentioned Login Instance(s) and the NAT Instance. The NAT instance is an instance created by the

Control Node that performs Network Address Translation (NAT) for each of the EC2 instances running within the VPC. This allows the internal instances to talk out to the Internet in order to pull down security patches and to talk with DynamoDB but does not allow for connections originating from the Internet for security purposes.

One of the other critical networking features configured by the AWS HPC cluster Control Node is VPC Peering. This is required due to the fact that the Control Instance has to be launched in a different VPC then the AWS HPC cluster will be launched into. This means that the Control Instance inherently cannot talk to any of the instances within the VPC that it creates. This is where VPC Peering comes in; it allows two VPCs to act like they are part of a larger network so that all of the instances can talk to each other just like they could if they were in the same VPC.

DynamoDB

The last of the major core subsystems of an AWS HPC cluster that will be discussed is DynamoDB. This has the equivalent of the user database and the system admin's brain in a traditional HPC cluster as this is where all the information regarding the configuration, users, passwords, and cluster meta-data is stored. DynamoDB is absolutely critical to the proper functioning of an AWS HPC cluster as each of the cluster resources depend on it for information about the rest of the cluster.

In order to help with the indexing and faster lookups of the different objects that are stored, two DynamoDB tables are used. One table is a lookup table that contains an index and then a pointer to the real object in the second table. The other table is an object table that actually stores the objects associated with the cluster and is referenced by the lookup table. This provides flexibility in creating quick and simple indexes that can then be used for faster lookups.

Each of the previously mentioned instances write their own entries to DynamoDB during their boot process in order to inform the rest of the cluster that they have successfully launched and are configured correctly. This allows flexibility and better tracking of the instance states along with a much simpler way of actually determining if an instance started up successfully or not.

The DynamoDB tables also have the ability to store multiple AWS HPC clusters at a time. This reduces the charges that come with having many DynamoDB tables as well as provides a convenient place to query to find all the information regarding all the AWS HPC clusters that a particular user has created.

Programming Languages

Each programming language has different strengths and weaknesses which need to be carefully analyzed in order to determine which language is the best fit for a particular project. However most of the time there is not just one programming language that can effectively achieve all of the goals associated with a project. This was the case with the Dynamic AWS HPC Cluster Project. After doing an analysis of many different programming languages it was determined that more than one programming language would be required in order to achieve the end goal of the project. So after much analysis three different languages were chosen for use in the project: Python, JavaScript, and C.

Python

Amazon Web Services has a very wide range of SDKs available ranging from Java, JavaScript, Go, PHP, .NET, Ruby, and Python. During the initial stages of the project many of the available SDKs were investigated and eventually it was determined that the Python SDK was the best SDK that was currently available. Utilizing Python as the main programming language for the project also allowed for the use of Bottle for interfacing the server and Web UI code. Each of the different instances/groups that were described in the previous sections accomplishes their dynamic configuration and creation through the use of a Python script that is run during the launch of the AWS Instance. Each instance contains all of the different configuration scripts for each type of instance that the Dynamic AWS HPC Cluster Project supports. The type of instance that will be created all depends upon which script the Control Instance specifies in the launch instance command as an AWS concept called "User Data". This is custom data that can be passed to an instance at launch and can even run scripts, such as the instance configuration python scripts. The instance specific script then runs, does all the heavy lifting and dynamically creates any of the extra AWS resources that may be required for the instance to fulfil its role within the cluster.

Bottle

Bottle is a lightweight WSGI micro web-framework for the Python programming languages [20]. Bottle allows for HTTP requests to be mapped to server side Python functions through the use of clean and simple directives within the Python code itself. This is the "glue" that joins the Web UI code to the server side Python code that actually does all the heavy lifting with the AWS APIs. When a user chooses their options from the Web UI and clicks “Create”, a Bottle route is then called that references a server side Python function that actually goes and calls the AWS APIs in order to create all of the resources that were requested by the user.

Boto

Boto was the initial AWS Python API set chosen to be incorporated into the Dynamic AWS HPC Cluster Project. This was due to mainly in part to the excellent documentation, constant updates, and the majority coverage of the AWS API set. Boto was used extensively throughout the server code to do all the communication between AWS and the server. There were originally some AWS API calls that were needed that could not be done with Boto so instead the equivalent AWS CLI commands were utilized for those specific instances.

Botocore

Botocore is a newer and vastly improved version of Boto that is much faster and cleaner to use. In fact Amazon’s own CLI interface that they provide and the newer version of Boto, Boto3, are both built off of the Botocore library. So instead of continuing to use Boto, it was decided that it was time to start converting the AWS API calls from Boto to Botocore since there was no reason to continue to use Boto, which is built off of Botocore, when instead Botocore itself can be used. Since it is the basis of the AWS CLI tools, Botocore is updated and maintained much more than the current Boto3 project and Botocore also gets many of the new updated features much quicker than Boto3. This means that by utilizing Botocore for the AWS API calls instead of

Boto3 it is easier and faster to integrate newer AWS services into the Dynamic AWS HPC Cluster Project.

JavaScript

JavaScript is used within this project in order to create a simple, lightweight, and responsive UI that could be used to make AWS HPC cluster creation easier than simply just leaving it as a command line only tool. The main JavaScript frameworks that were utilized for the UI were Dojo and Dojo Mama. The advantages for utilizing the Dojo and Dojo Mama frameworks are that they provide templates for navigation and provide extra widgets and specialized objects that build upon the objects that are normally available through JavaScript [21]. This allows the UI to be simple yet sophisticated and allows for easy and quick navigation between the different parts of the UI. JavaScript was also chosen due to the ease at which it could be integrated with the Python backend, through Bottle, that handled all of the AWS API calls and did much of the heavy lifting for the Dynamic AWS HPC Cluster Project.

C

Since Python is an un-compiled language it makes it much easier for attackers to go through the source code and compromise the security of the encryption and decryption that is being done to sensitive data. While Python does allow for the compilation of source files, the process is easily reversible and not very secure especially when dealing with passwords and other sensitive user data. This means that in order to make the AWS HPC cluster authentication and user data storage more secure, all of the functions that deal with sensitive data, encryption, and decryption are coded and compiled in the C programming language. While it is not impossible to reverse engineer a compiled C binary, it is a little more difficult and requires more time and effort by the attacker. The goal is to make the path that the project takes in order to encrypt and decrypt data more complex and harder to follow for anyone looking to reverse engineer the system.

CHAPTER V

TESTING AND VALIDATION

Correctness

For the Dynamic AWS HPC Cluster Project, there are two different types of correctness that must be measured in order to assure researchers that the Dynamic AWS HPC Cluster Project generated clusters can keep pace with traditional HPC clusters. One type of correctness is that the Dynamic AWS HPC Cluster Project has to create the actual cluster that the user has specified and the other type of correctness deals with the actual floating point precision of the cluster's CPUs. Both of these types of correctness have been thoroughly tested and the results of each different test are discussed below.

Floating Point Accuracy

Many of the jobs that are run on an HPC cluster are scientific in nature and rely on a certain level of floating point accuracy from the cluster's CPUs. This is because the computations are dealing with very small and precise numbers where any rounding errors could cause false results and lead to other issues with the research. This is one area where the Dynamic AWS HPC Cluster Project must be the same or better than an "on premise" cluster because otherwise the results calculated using the clusters generated by the Dynamic AWS HPC Cluster Project would be invalid.

There are many different benchmarks out there for testing the floating point accuracy of a CPU; however there are only a couple, such as Linpack and the HPC Challenge Benchmark suite, which are designed for use with HPC clusters. For this test, the HPC Challenge Benchmark suite was run on two different clusters and then compared. The first cluster was an 8 node “on premise” cluster where each node had a 2.0Ghz Intel Xeon E5-2660v2 CPU with 128GB of RAM. The second cluster was also an 8 node cluster but was generated the Dynamic AWS HPC Cluster Project and used the r3.4xlarge AWS instance type which utilizes a 2.5Ghz Intel Xeon E5-2670v2 CPU and had 122GB of RAM. Since AWS only has certain configurations for their instance types it is very hard to find an instance type that is an exact match to the “on premise” resources and this was the configuration that was most similar to the “on premise” cluster.

The HPC Challenge Benchmark suite was run on each of these clusters utilizing the Torque/Maui HPC scheduler along with the MPICH 3.0.4, BLAS, and Atlas SSE3 libraries. The HPC Challenge Benchmarking suite was run a total of ten times on each cluster and the results from each run were averaged together to get the average performance of each cluster. The HPC Challenge Benchmark suite contains many different tools, but the one that is utilized in checking the floating point accuracy is the HPL benchmark. The HPL benchmark is software that solves a random dense linear system in double precision and provides a testing and timing program to quantify the accuracy for the obtained solution [22]. HPL first generates a random dense linear system and then proceeds to solve the generated system and perform a residual check in order to make sure that the solution is within the acceptable error range. This is accomplished by utilizing relative machine precision which for all benchmark runs was taken by HPL to be 1.110223e-16 on both clusters. This value is then utilized by a residual check that HPL runs and the value is compared to the inputs to ensure its accuracy. Both clusters passed the tests for all ten runs of the HPL benchmark showing that the floating point accuracy between the two clusters is similar.

Cluster Creation Correctness

The other critical area of correctness for the Dynamic AWS HPC Cluster Project is the correctness of the cluster that it creates. If the user requests a certain cluster, they expect to get what they requested and if the Dynamic AWS HPC Cluster Project produces a different cluster than expected it would be unacceptable. This is a type of correctness that only really only applies to clusters created by the Dynamic AWS HPC Cluster Project as “on premise” clusters are pre-provisioned and are not dynamically generated. In order to test the correctness of the Dynamic AWS HPC Cluster Project, ten different cluster configurations were created and launched through the project and then the resulting clusters were compared to the original requested clusters. Table 5.1 shows the different cluster configurations that were launched and then deleted, and the pass/fail rating for both creation and deletion.

If the requested cluster was the same as the generated cluster the create test passed but if it differed just a little bit the test failed. Same for the delete test, if the generated cluster deleted all the created resources it originally created then the test passed, if it didn’t delete all the resources then the test failed. For each of the requested cluster configurations tested with the Dynamic AWS HPC Cluster Project, all of the created clusters exactly matched the requested cluster configuration and all the generated clusters successfully deleted after being created. These tests show that the Dynamic AWS HPC Cluster Project is reliable in spinning up and deleting the requested clusters and will produce the exact cluster that is requested. This is critical as each time instances are launched within AWS the user is charged for the full hour even if the AWS instance is only used for a minute. This means that if the Dynamic AWS HPC Cluster Project launches the wrong instances or number of instances the user will be charged for the full hour of use which can be expensive if they are using large AWS instance types.

Requested Cluster Configuration	**Resulting Cluster Create and Delete Pass/Fail**
4 Compute Instances, 4 Filesystem Instances, 1 Login Instance, 1 Scheduler, 1 NAT Instance	Create: Pass Delete: Pass
4 Compute Instances, 4 Filesystem Instances, 1 Login Instance, 2 Schedulers, 1 NAT Instance	Create: Pass Delete: Pass
4 Compute Instances, 8 Filesystem Instances, 2 Login Instances, 1 Scheduler, 1 NAT Instance	Create: Pass Delete: Pass
8 Compute Instances, 4 Filesystem Instances, 2 Login Instances, 1 Scheduler, 1 NAT Instance	Create: Pass Delete: Pass
2 Compute Instances, 1 Scheduler, 1 NAT Instance	Create: Pass Delete: Pass
8 Compute Instances, 8 Filesystem Instances, 2 Login Instance, 2 Scheduler, 1 NAT Instance	Create: Pass Delete: Pass
8 Compute Instances, 8 Filesystem Instances, 1 Login Instance, 1 NAT Instance	Create: Pass Delete: Pass
1 Compute Instances, 1 Filesystem Instances, 1 NAT Instance	Create: Pass Delete: Pass
1 Login Instance, 1 Scheduler, 1 NAT Instance	Create: Pass Delete: Pass
4 Filesystem Instances, 1 Login Instance, 1 Scheduler, 1 NAT Instance	Create: Pass Delete: Pass

Table 5.1 Cluster Creation and Deletion Correctness Testing

Performance

Just having the cluster spin up correctly and have a certain level of floating point accuracy does not mean anything if the cluster's performance is not very good. Performance when running HPC jobs is something that many HPC cluster users take very seriously. Many users have to constantly walk the fine line of performance and cost in order to ensure that they stay under budget while still obtaining as many and as accurate results as possible. In this section, the performance of the clusters generated by the Dynamic AWS HPC Cluster Project and the performance of a local "on premise" cluster will be compared and contrasted to show the advantages and disadvantages of utilizing AWS based HPC clusters.

Network Performance

Network speed is critical for many HPC applications and even the slightest slowdown of the network can be the difference in an HPC job running for a few hours versus the same HPC job running for a period of days. Unfortunately the network performance of AWS is not a simple issue; Amazon is very secretive about the backbone network that AWS utilizes for the EC2 instances which makes it hard to figure out exactly what network performance certain instances will have. Therefore in order to fairly cover the different types of network performance in AWS two separate benchmarks have been performed to showcase some of the differences between the upper levels of network performance within AWS.

The four levels of network performance for an AWS EC2 instance are: low, low to moderate, moderate, high, and 10GB. With each increase in the level of network performance the EC2 instance cost per hour also increases. Since AWS EC2 instances are preconfigured the user cannot choose the type of network performance that a certain instance type has, which means that the user is limited in their options if network performance is a top concern. This was an issue that was encountered when trying to compare the “on premise” cluster and the cluster created by the Dynamic AWS HPC Cluster Project. In order to most closely match the number of CPUs and RAM that the “on premise” cluster had, an instance type with only a “High” level of networking had to be chosen. All of the other instances that had the “10GB” level of networking contained either too few CPUs, to many CPUs, not enough RAM, or too much RAM. This caused the choice to be made to go with the instance type most closely matching the amount RAM and number of CPUs which handicapped the network performance of the cluster created by the Dynamic AWS HPC Cluster Project that the HPC Challenge Benchmarks were ran on.

In order to show the difference between the “High” level and “10GB” level of networking, network data transfer benchmarks were ran on two instances that had the “10GB” level of networking performance and compared against a standard 10GB Ethernet network. These benchmarks were also done utilizing some of the advanced networking features that the Dynamic AWS HPC Cluster Project utilizes that similar projects do not. The benchmarks also show how the network compares to a “bare metal” network with and without these special features enabled.

AWS 10GB Network Benchmarks

Many “on premise” HPC clusters have a finely tuned network backbone that is specially formulated for low latency and high throughput between the instances of the cluster. However, AWS is a more generalized computing environment that supports many applications and is highly virtualized so there are fewer optimizations that can take place in the network itself and instead more optimizations that have to be made to the virtualization software [23].

Two of these optimizations that AWS introduces to attempt to minimize latency and jitter in the network traffic are the concepts of Placement Groups and Enhanced Networking. Placement Groups are a way to “logically group instances within AWS that enables applications to participate in a low-latency 10Gbps network” [18]. While Enhanced networking utilizes single root I/O virtualization (SR-IOV) in order to provide higher performance (in packets per second), lower latency, and lower jitter [24]. Unlike CycleCloud, CfnCluster, and Star Cluster, the Dynamic AWS HPC Cluster Project enables Enhanced Networking, creates Placement Groups, and places compatible instances inside these Placement Groups without the user having to do anything. As the benchmarks show, this greatly increases the network performance between the cluster instances which can drastically reduce errors and long run times for certain applications.

The backbone network of AWS utilizes the Ethernet protocol and more specifically for instances that support the use of Placement Groups and Enhanced Networking, a “10GB” level of network performance based upon the 10Gbps Ethernet standard. Benchmarks were performed on both the AWS network with Placement

Groups and Enhanced Networking enabled as well as without Placement Groups and Enhanced Networking enabled. These results were then compared to the exact same benchmarks that were performed between two local "bare-metal" machines that were also connected by a 10Gbps Ethernet network backbone.   The main goal in performing these benchmarks was to determine if the network speeds offered by AWS were comparable to the "bare-metal" speeds that one would get if running the instances locally.

The benchmarks were obtained through the use of the iPerf network benchmark tool, which simply sends massive amounts of data for a set time period and records certain statistics about the speed and jitter found within the data transfer window [25]. Since internally, AWS instances can utilize Jumbo Frames, which are simply Ethernet Frames that have a maximum transmission unit (MTU) of 9000 instead of the normal 1500 MTU, the benchmarks were performed with both 9000 MTU and 1500 MTU to get a better feel for how the networks compared. Both the UDP bandwidth and TCP bandwidth were tested in order to see just how much the numbers fluctuated between the two different protocol types.

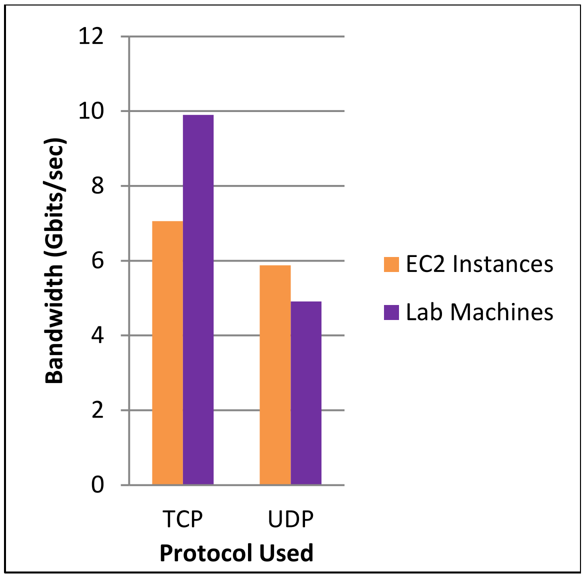


Figure 5.1: AWS HPC Cluster project EC2 Instances no Placement Groups or Enhanced Networking vs Lab Machines (MTU 9000)

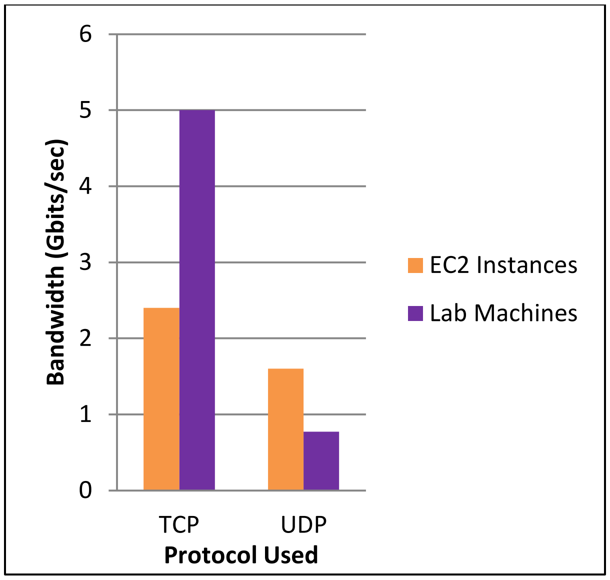
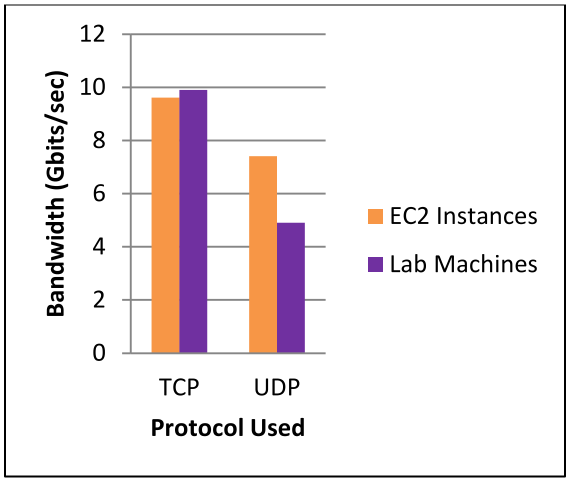


Figure 5.2: AWS HPC Cluster project EC2 Instances no Placement Groups or Enhanced Networking vs Lab Machines (MTU 1500)

Figures 5.1 and 5.2 show the output of benchmarking two standalone Dynamic AWS HPC Cluster Project c3.8xlarge type instances with a network performance level of “10GB” and that are not within a Placement Group nor have Enhanced Networking enabled against the benchmarking results of the Clemson Networking Lab machines with the MTUs of 9000 and 1500 respectively. Without the Placement Group feature, the Clemson Networking Lab Machines outperformed the AWS machines by 2.84 Gbits/sec with an MTU of 9000 and by 2.60 Gbits/sec with an MTU of 1500. However the same was not true for the UDP benchmarks, as the AWS network outperformed the local machines by 0.97 Gbits/sec with an MTU of 9000 and by 0.83 Gbits/sec with an MTU of 1500. This is along the lines of what was anticipated for TCP due to the increased overhead that virtualization puts on the network. However, for the UDP benchmarks to be higher in AWS was an unexpected surprise.



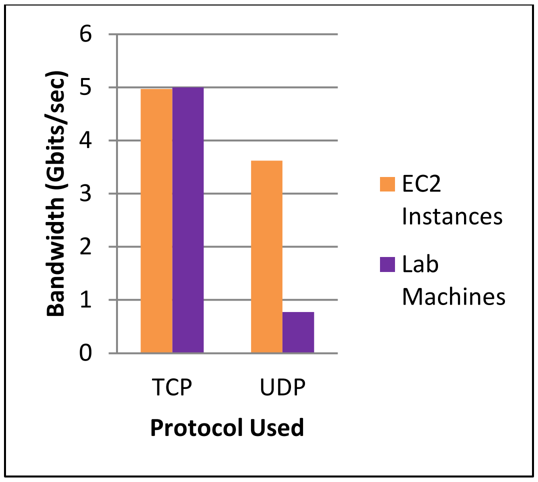


Figure 5.3: AWS HPC Cluster project EC2 Instances with Enhanced Networking and Placement Groups vs Lab Machines (MTU 9000)

Figure 5.4: AWS HPC Cluster project EC2 Instances with Enhanced Networking and inside Placement Group vs Lab Machines (MTU 1500)

Figures 5.3 and 5.4 show the output of the iPerf benchmarks on the two standalone Dynamic AWS HPC Cluster project c3.8xlarge instances within Placement Groups and with Enhanced Networking enabled versus the Clemson Networking Lab Machines with an MTU of 9000 and 1500 respectively. The AWS network performance averaged out to only be 0.29 Gbits/sec less than the Clemson Networking Lab Machines with an MTU of 9000 and a measly 0.30 Gbits/sec less than the Clemson Networking Lab Machines with an MTU of 1500. However, the AWS UDP benchmarks were far greater than the local UDP benchmarks averaging about and 2.50 Gbits/sec more bandwidth available for UDP with an MTU of 9000 and 2.85 Gbits/sec more bandwidth available for UDP with an MTU of 1500. Simply by enabling Placement Groups and Enhanced Networking by default, the network performance provided by the instances that support these features is much greater than the defaults of the other similar products that do not enable these features automatically.

HPC Challenge Network Benchmarks

The last section focused mainly on the bulk transfer of data from point to point within the AWS network which is a useful benchmark when transferring very large files over the network but it is not exactly indicative of how Message Passing Interface (MPI) and other network heavy applications will perform on the network. Since MPI and other network heavy applications utilize many simultaneous connections and many times have large numbers of packets that are destined for the same place, the performance tends to be significantly lower than the pure traffic based network performance.

The performance benchmarks that were used to determine the network performance for MPI applications was the HPC Challenge Benchmark suite. However, instead of utilizing the HPL part of the benchmark suite, this time the focus was on the Latency-Bandwidth-Benchmark. For this round of benchmarks, the HPC Challenge Benchmark suite was ran a total of ten times and the results from each run were averaged together in order to get an overall picture of the underlying network performance.. The results of these network tests were greatly different from the “10GB” network level benchmarks that were performed in the previous section. This is due to the instances used to create the cluster generated by the Dynamic AWS HPC Cluster Project had a level lower network performance, “High” versus “10GB”, than the instances used for the “10GB” benchmarks in the previous section. This was the only way to “fairly” obtain the benchmarks because any other AWS instance type that had a “10GB” level of network performance would not have met, or exceeded, the CPU and RAM requirements needed to compare to the “on premise” cluster.

The “on premise” cluster is comprised of 8 nodes where each node had a 2.0 GHz Intel Xeon E5-2620 CPU with 128GB of RAM. While the cluster generated by the Dynamic AWS HPC Cluster Project was comprised of 8 nodes and used the r3.4xlarge AWS instance type which utilizes a 2.5 GHz Intel Xeon E5-2670v2 CPU, had 122GB of RAM, and a “High” level of network performance. Since AWS only has certain configurations for their instance types it is very hard to find an instance type that is an exact match to the “on premise” system and this was the configuration that was most similar to the “on premise” cluster that was available to utilize.

The key outputs of the Latency-Bandwidth-Benchmark are the Randomly Ordered Ring Latency and the RandomRing Bandwidth. These parameters report both the available bandwidth and the latency per process that are randomly ordered in a ring. The available bandwidth per process is defined to be the total amount of message data divided by the number of processes and the maximal time needed in all processes. The latency per process is defined as the maximum time needed in all processes divided by the number of calls to the MPI_Sendrecv or MPI_Isend in each process [26]. The average results of the RandomRing Bandwidth and the Randomly Ordered Ring Latency are shown in Figures 5.5 and 5.6. The “on premise” cluster achieved an average RandomRing Bandwidth value of 0.0753 gigabits per second while the Dynamic AWS HPC Cluster Project cluster only achieved an average score of 0.0187 gigabits per second. That is a dramatic 75.1% decrease in bandwidth between the two clusters. The results of the Randomly Ordered Ring Latency benchmark are not much better; the “on premise” cluster averaged a Randomly Ordered Ring Latency value of 63.422

microseconds while the Dynamic AWS HPC Cluster Project measured an average value of 186.44 microseconds. Here having a higher number is not desirable as it means that there is a 194% increase in latency from the "on premise" cluster to the AWS cluster.

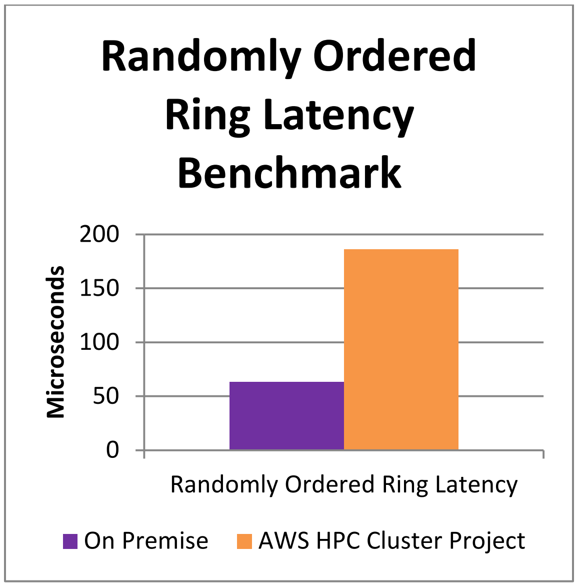


Figure 5.5: Randomly Ordered Ring Latency

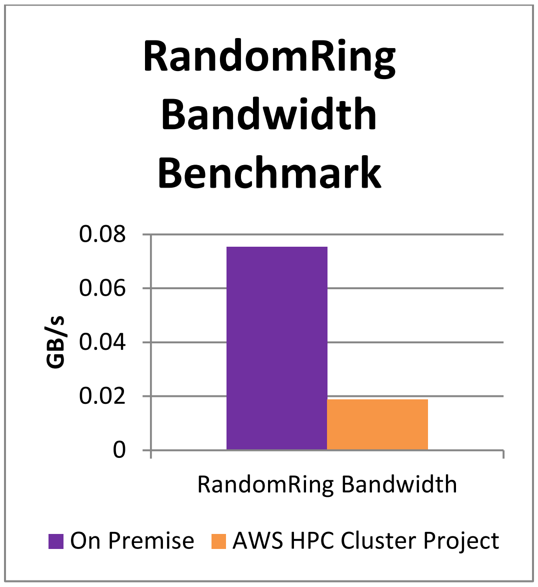


Figure 5.6: RandomRing Available Bandwidth

There were two other benchmarks within the HPC Challenge Benchmarking suite that were run that measured Fast Frontier Transform (FFT) and Random Access utilizing MPI. The FFT benchmark measures the floating point rate of execution of double precision complex one-dimensional Discrete Fourier Transform in gigaflops per second while the Random Access benchmark measures the rate of integer random updates of memory in gigaupdates (GUPs) per second [21]. These benchmarks were run in order to get an idea of the MPI performance that could be obtained by the two different clusters. The results of these benchmarks are shown in Figures 5.7 and 5.8.

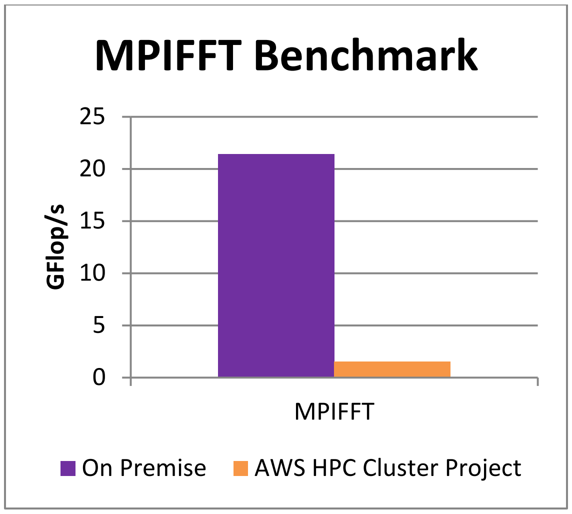


Figure 5.7: MPIFFT Benchmarks

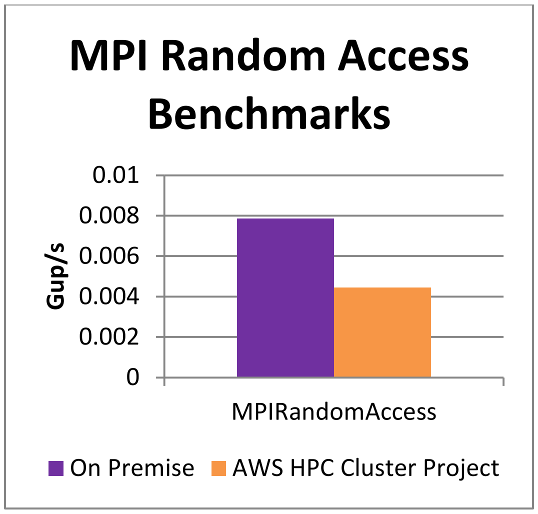


Figure 5.8: MPI Random Access Benchmarks

These numbers show that by far the "on premise" cluster outperforms the cluster that was generated by the Dynamic AWS HPC Cluster Project. However, there is a reason that this is the case that was briefly stated earlier, the AWS level of network performance. An "on premise" cluster does not have any throttling on the amount of the available network that it can utilize when performing the benchmarks, however this is not the case on AWS. As previously mentioned, AWS has certain levels of network performance that are assigned to each different AWS instance type. This means that certain instance types are limited in the amount of the underlying AWS network that they can use at any given time. This explains the drastic differences between the two cluster's performances on the network section of the benchmarks. The AWS instance type that was used in the AWS HPC cluster had a network performance level of "High" which is the second highest level of network performance available within AWS. However, the difference between "High" and the highest level of network performance "10GB" is not well documented and appears to be relatively large as the average available bandwidth calculated for the “10GB” instances in the previous section averaged right around 9.61 GB/s while the “High” instances averaged only 2.03 Gb/s. This is because since AWS instances are “multi-tenant”; meaning that there are multiple AWS instances sharing the same physical connection, the level of network performance is what specifies the “priority” of the AWS instance in terms of how much network bandwidth it is allocated. If the other AWS instances that are sharing the same connection are all of a lower network performance level then there will be more network bandwidth allocated to the AWS instance with the higher network performance level. However, the converse of that is true as well if the AWS instances that the user is using are of a lower network performance level than that of all the other AWS instances in the same “rack” than the network performance of those instances will be degraded.

All of this is in stark contrast to the “on premise” cluster which has full and complete access to its underlying network. Since the “on premise” cluster nodes are not sharing their physical connections with each other, the amount of available bandwidth for those nodes is much greater than that of an AWS instance that does not have the top network priority. Thus this explains the drastic differences between the two cluster’s HPC Challenge benchmarks and the benchmarks between the two AWS instances that utilized the “10GB” network performance level in the previous section.

Job Completion Time Benchmarks

Another major area of concern for HPC users is the time taken for a job to complete as usually the researchers are sitting around waiting for the job to complete before they can continue onto the next phase of their research. In order to show the differences between the job completion times on an “on premise” HPC cluster versus a cluster that was generated by the Dynamic AWS HPC Cluster Project, an MPI job that finds the largest prime number within a given set of numbers was ran through the HPC scheduler Torque/Maui on both clusters that utilized 8 nodes and 16 processors per node for a total of 128 processes across all the nodes on each cluster. The MPI job was ran a total of ten times on each of the clusters in order to achieve an average job completion time. All the parameters and executables used for this test were the same on each of the clusters. The only differences between the two clusters were the total number of “processes” available on the instances. The “on premise” cluster had twenty total available processes while the AWS instances had only sixteen. This is due to the differences in the processors and the rigidity of the AWS instance types.

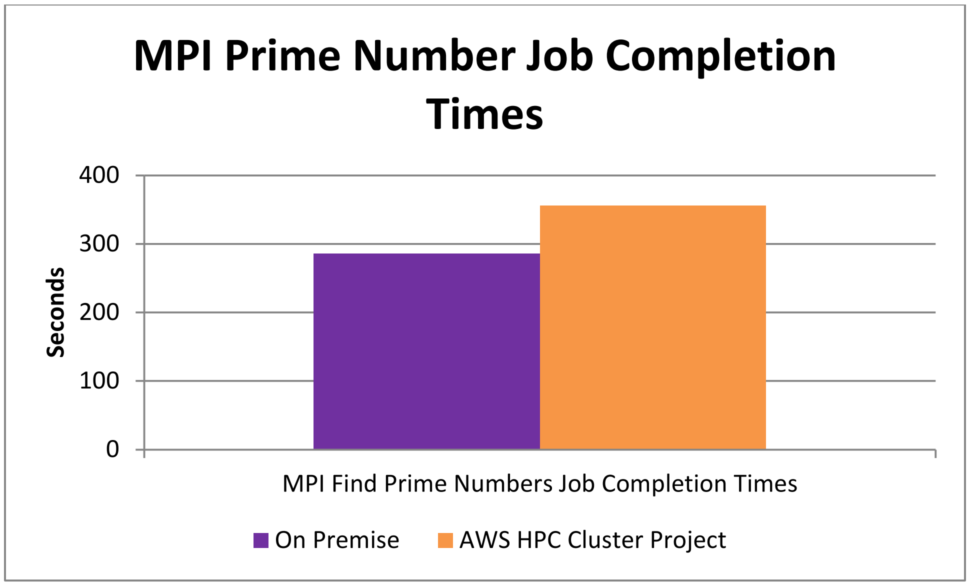


Figure 5.9: MPI Prime Number Job Completion Times

Figure 5.9 shows the average results of the ten MPI job runs. The job completion times were about seventy seconds faster on the “on premise” cluster than the job completion times on the cluster generated by the Dynamic AWS HPC Cluster Project. This is about a 24.5% increase in completion time for the job which is a significant increase in the time that it takes the job to complete. However, there is a feasible explanation for why this happens and it has to do with the MPI network benchmarks that were run on the same two clusters and discussed in the previous section.

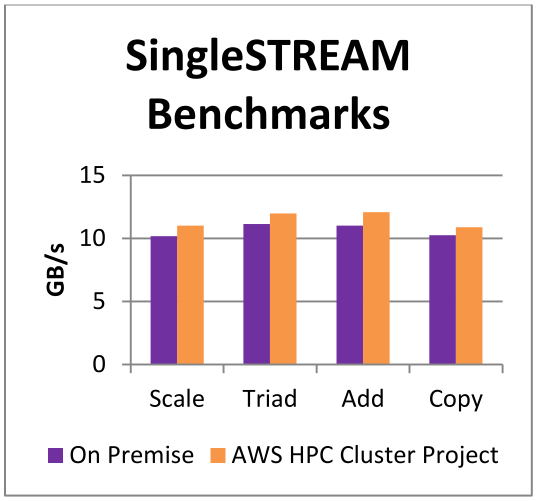
Since the job utilized MPI for doing all of its computation, the limiting factor on how fast the job can complete is again the network performance of each individual node in the cluster. This yet again points back to the differences in the network performance levels for each AWS instance type. Since there is a drastic drop off in network performance and cost for each descending level of network performance theoretically researchers could see much better performance results by paying a little more and choosing an AWS instance type that utilized the "10GB" level of network performance instead of trying to match the number of CPUs and amount of RAM that their "on premise" cluster has. However, even with choosing an instance that has the "10GB" network performance level they will probably still see some slightly slower job completion times due to the shared nature of the AWS network and the AWS instances.

Computational Performance

Computational performance is another major area that can affect a researcher's results. If the researcher's job is very computational heavy, than the computational performance of the CPU on the individual nodes is extremely important. If the computational performance of the AWS instance CPU is not good enough then it will take computational heavy jobs much longer to complete on the clusters generated by the

Dynamic AWS HPC Cluster Project then it would take the same job to complete on an "on premise" cluster.

For these benchmarks, the HPC Challenge Benchmark suite was used with the focus being on the SingleSTREAM, StarSTREAM, HPL Calculated Teraflops, and the PTRANS benchmarks. The STREAM benchmarks measure the sustained memory bandwidth to and from memory. The SingleSTREAM benchmarks test the memory bandwidth on a single processor on one of the nodes chosen at random from within the cluster and then the STREAM benchmark is performed ten times and averaged. While the StarSTREAM benchmarks performs the same benchmark for memory bandwidth but instead of just running on one processor on a random node, concurrent copies of the benchmark are run on each processor on each node in the cluster ten times and averaged [21]. The HPL Calculated Teraflops benchmark measures something completely different, the rate of execution for solving a randomly generated dense linear system of equations in double floating-point precision (IEEE 64-bit) arithmetic using MPI. While the PTRANS benchmark measures the rate at which the system can transpose a large array [21].



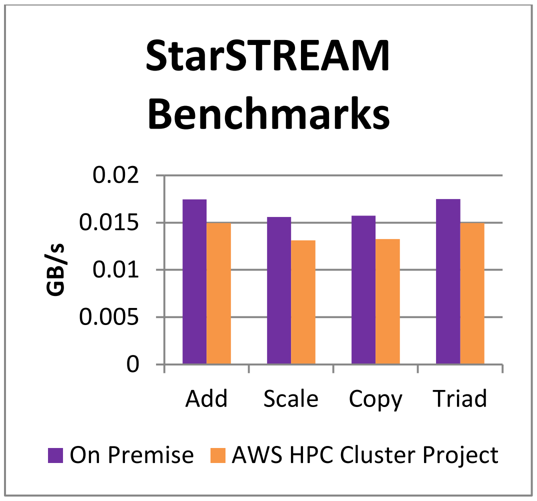


Figure 5.10: SingleSTREAM Benchmark

Figure 5.11: StarSTREAM Benchmark

Figures 5.10 and 5.11 show the results of the two STREAM benchmarks for both of the clusters. For the SingleSTREAM, the cluster generated by the Dynamic AWS HPC Cluster Project outperformed the “on premise” cluster by 7.5% on the Scale tasks, 7.0% on the Triad tasks, 8.8% on the Add tasks, and 5.9% on the Copy tasks. These are marginal increases compared to the “on premise” cluster but for a CPU intensive job, these marginal increases per CPU could add up to decrease the time needed for certain computations to complete which could speed up the job run time if used in conjunction with the “10GB” networking level of performance.

However, the StartSTREAM benchmark seems to show otherwise. These results show that the “on premise” cluster marginally outperformed the cluster generated by the Dynamic AWS HPC Cluster Project by 15.7% on the Scale tasks, 14.5% on the Triad tasks, 14.3% on Add tasks, and 15.7% on the Copy tasks when the STREAM benchmark is ran on all of the processes simultaneously. This shows that while the AWS instance’s CPUs may be marginally faster than the “on premise” cluster’s CPUs when all the CPUs are running simultaneously the AWS instance's CPUs actually run slower than the "on premise" cluster.

The hypothesis on why has to do with the shared "multi-tenant" nature of the AWS instances. Since there can be multiple AWS instances running on the same hypervisor and hardware, if all of the AWS instances on that particular hardware or hypervisor happen to all be using the CPU at the same time, there will be a slight decrease in performance to do the available hardware constraints. When AWS instances are launched by the Dynamic AWS HPC Cluster Project, they are placed within Placement Groups which helps to improve the network performance of the instances but also can place the instances on the same hypervisor or hardware. This means that when running the STREAM benchmark on all the cluster instances simultaneously the hypervisor and hardware supporting these AWS instances will be taxed more than if the benchmark was only running on a single one of the AWS instances. This taxing of the hardware can lead to the decrease in performance from running the STREAM benchmark on one instance to running it on many instances simultaneously.

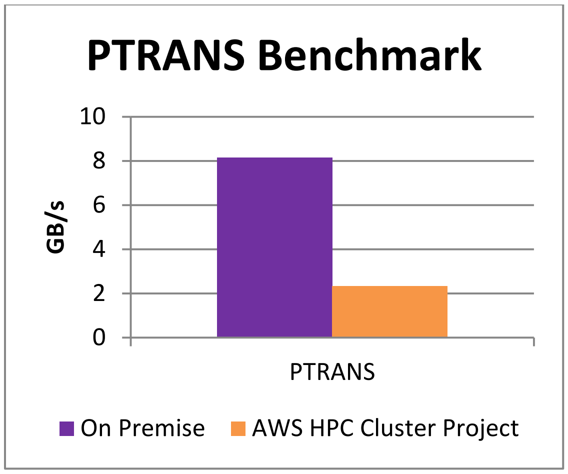


Figure 5.12: PTRANS Benchmarks

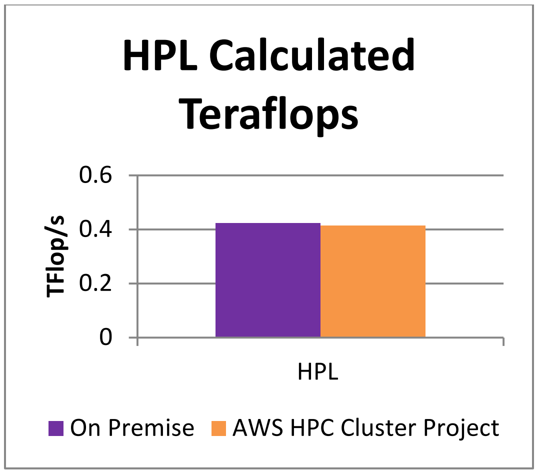


Figure 5.13: HPL Calculated Teraflops

Figures 5.12 and 5.13 show the results of the PTRANS and HPL Calculated Teraflops benchmarks respectively. The PTRANS benchmark performed on the "on premise" cluster heavily outperformed the same benchmark performed on the cluster generated by the Dynamic AWS HPC Cluster Project. The "on premise" cluster actually performed 71.4% better than the cluster generated by the Dynamic AWS HPC Cluster Project. This is due largely to the fact that the PTRANS benchmark relies heavily on pairs of processors communicating with each other simultaneously through MPI which as stated in the previous sections is known to perform better on the "on premise" cluster due to the AWS network performance limits on the AWS instance types that were used in the generation of the cluster from the Dynamic AWS HPC Cluster Project.

The HPL Calculated Teraflops benchmark told a different story though, as this benchmark showed that the two clusters were very close in their performance. Although in the end, the "on premise" cluster still outperformed the cluster generated by the Dynamic AWS HPC Cluster Project, it was only by 2.1%. This is in stark contrast to the PTRANS benchmarks and the bandwidth and latency benchmarks that have previously been discussed. This number becomes even more interesting when looking at the fact that the HPL Calculated Teraflops benchmark does utilize MPI in order to solve the problem. However, the use of MPI in the benchmark itself is much less than that of the other benchmarks that utilize MPI so there is not as much congestion or strain put on the AWS network and hence the AWS network performance level does not matter as much as it does with other benchmarks.

Cost Tradeoffs

One of the major draws to cloud based resources is the allure of the potential cost saving options that are offered by the pay-per-use model instead of having to pay to keep resources running all the time locally. This allure actually is somewhat of an illusion however and there have been studies that have studied the cost differences between the use of AWS instances and the use of local hardware. One such study *Cost-effective HPC: The Community or the Cloud?* comes from Purdue University and analyzes the cost effectiveness of AWS verse the "on premise" clusters found at Purdue University [27]. This study goes to point out that a highly utilized HPC cluster is actually cheaper to run locally instead of running it within AWS. However, the actual rate that quantifies "highly utilized" drastically varies from institution to institution and is a hard metric to successfully capture due to its many dependencies. These dependencies include the amount of IT staff available to support and maintain the cluster, the number of users, size of the cluster, and the amount of available up front capitol to invest into the cluster. All of these factors contribute to the analysis and at a certain variable point; the cost of running the cluster within the confines of AWS or other cloud providers becomes greater than the initial up front cost to create a local HPC cluster.

This also works in the other direction also, if the user does not have a large pool of available capital to invest, the IT support to maintain the cluster, or simply will not utilize the cluster enough to warrant running it continuously then they are a prime candidate to move their HPC cluster to the AWS or other cloud. By utilizing the AWS cloud, the user can leverage the ability to be able to only pay for and operate the cluster as they need it. This can save them valuable money if their budget is tight and can save them the time and hassle of having to set everything up themselves.

Another aspect to consider in terms of the cost of operating an HPC cluster within the AWS cloud is the acute awareness of exactly how much money that each job costs the user to run. With a local cluster the user has to upfront the money in order to obtain the resources needed to run their jobs and therefore the money is already spent. Hence they can run their jobs and anything else as little or as much as they want but either way the money is already spent. This is a completely different mindset when using AWS as the user is charged per hour that the AWS instances are running. This means that the user may, even subconsciously, restrict the usage of the AWS cluster because they are conscious of the fact that they are spending more money each time they start up the AWS based cluster. This can be a tough mindset to adjust to as it is programmed in most users' brains to conserve as much money as they can.

Security Tradeoffs

Security is something that is beginning to take center stage as more and more companies start to think about moving more of their critical and sensitive services into the cloud. There are many people who hesitate to migrate sensitive data into the cloud because they are no longer in physical control of the data and there is always the possibility that the cloud service provider could be hacked and their sensitive data could be compromised. This risk is not likely to go away any time soon, but AWS and other cloud providers have started to strengthen their security by working to achieve many Security Compliance Certificates in order to assure their customers that they are taking the protection of their data very seriously. This along with other cloud service provider security tools allow users to stay on top of the virtual part of securing their machines but still does not provide a way to physically ensure the safety of the machines like having the cluster on premise does. In order to explain this concept in more depth, a brief comparison of security advantages and disadvantages for both the AWS cloud based cluster and an “on premise” cluster will be given below.

AWS Security

One of the first things that people think about when they think cloud security is the fact that their sensitive data is being stored in some datacenter of which they have no control over. While this is true and is a disadvantage in the sense that the user does not have physical access to the machines or even control over who has physical access to the machines, it is also a security advantage in a way as well. This advantage comes from the fact that if there is a natural or manmade disaster at the location of the user, there is a good chance that the data center where the data is stored is fine and that the data will survive unharmed. Another area that fits into this category is the fact that AWS has 15 Certifications from different agencies such as the DoD CSM, FedRAMP, IRAP and FIPS along with compliance procedures for many privacy laws and regulations such as HIPPA [28]. This allows the users to feel more at ease that their data will be secure within the confines of the AWS network and services since AWS has undergone the rigorous testing process required for many of these certifications.

Another area that users can utilize in order to gain better control over their AWS security is the use of different AWS Services such as Security Groups and routing rules that allow the user to specify exactly what IP addresses can have access to certain ports and services on the AWS instances based off of IP CIDRs. This combined with the added security of all the AWS instances being managed by the user's own AWS account so they have full control over what resources are being created/deleted/running/paused at any given time means that the user does have options when it comes to securing virtual access to the cluster resources. AWS also provides a service called Cloud Watch that enables a detailed log to be kept about what services were used by which account user so that any suspicious activity can be tracked [29].

Local Security

Local cluster security allows the user to have control over exactly who has access to the physical machines and the data since all of the data is stored locally in the on premise datacenter. This means that all of the machines and data are managed by people that are usually within the same organization as the user and these same people are responsible for monitoring the network as well. This can be a security advantage as there is a stronger trust built if the user knows the people managing the infrastructure, but it can also be quite a disadvantage as well as if there is a security breach at the organization. In this situation, both the data and possibly even physical access could potentially be compromised at the same time. One disgruntled employee with the right credentials and a motive can take down the entire company especially with physical access to the machines.

With local machines and data storage the user also can have more control over the network and who has access into and out of the network as they have control over the networking devices and all of the configurations of these devices. The user also has the ability to obtain whichever security credentials/certificates that they deem necessary to their work flow and are not limited to just the security certificates obtained by AWS. This can be particularly useful when the user needs a more obscure security certificate as it can be painful and even impossible to get certification for AWS resources that the user wants to use.

CHAPTER VI

FUTURE WORK

Meta-Scheduler Utility Set

HPC schedulers have mainly focused on the allocation of resources across a large fixed number of dedicated HPC systems. However, these systems have certain limits such as the number of instances, types of instances, general hardware available, and the cost of operating these systems all the time whether they are being fully utilized or not. However, public clouds like AWS have the ability to bring a new dimension to these HPC schedulers by allowing for the dynamic creation of resources for which the user can choose the number of instances, type of instance, hardware, and even control the general cost of the instances as well. This means that the role of these traditional HPC schedulers needs to be re-evaluated in order to better utilize the resources that public clouds make available while at the same time staying consistent, compatible, and even interoperable with the current HPC schedulers of today. In order for this to happen a meta-scheduler utility set needs to be not only created but tightly integrated with the Dynamic AWS HPC Cluster Project in order to help users take even more control over their clusters.

This proposed meta-scheduler needs to be able to interoperate and interact with many of the current HPC schedulers such as Torque, PBS Pro, Slurm, Sun Grid Engine, and HTCondor without requiring any modification to the underlying scheduler. It will provide wrapper functionality in order to take advantage of certain AWS features that are not currently integrated into any of the previously mentioned HPC schedulers.

The proposed meta-scheduler utility set will at its core consist of three basic utilities that are fundamental to all current HPC schedulers: submit, status, and delete. These three utilities will provide the functionality that will enable a user to utilize a common interface in order to submit, monitor, or delete a job from any of the current generation HPC schedulers. But the biggest feature that the meta-scheduler utility set will add to these HPC schedulers is the ability to dynamically parse a job script and deploy the correct number and type of instances to AWS before actually launching the job. Then once the job is finished running, it will continue to monitor the launched instances and if they are no longer being used, they will be shut down. This saves the user money by only requiring them only to pay for the computing resources that they actually need.

Another area that the meta-scheduler should be able to handle is the area of data staging. This can be utilized within the current generation of HPC schedulers as well, as the ability to dynamically perform data staging before the instances are launched could save the user a good deal of money since the instances will not have to be sitting idle while the data is being transferred up to the cluster.

The last area that the meta-scheduler utility set should focus on is making the current generation of HPC schedulers easier to use. The current generation of HPC schedulers require the user to learn a completely different command set for each different HPC scheduler that they want to use. The meta-scheduler utility set would provide a common interface into all of these different HPC schedulers to where the user would just have to submit their job script for any of the HPC schedulers and if they happen to have that type of Scheduler running in their cluster, the meta-scheduler utility set would then parse the job and submit it to the correct scheduler without the user having to do anything else. This can be tightly integrated into the Dynamic AWS HPC Cluster Project and could even become the standard submission tool set used by the Dynamic AWS HPC Cluster Project.

REFERENCES

[1] "InCommon Federation." InCommon Federation. Internet2. Web. 19 Feb. 2016. <https://www.incommon.org/federation/>.

[2] "About Us." Cycle Computing. Cycle Computing. Web. 22 Feb. 2016. <http://cyclecomputing.com/about/>.

[3] "Scheduler Integration." Scheduler Integration — CycleCloud User Guide V6.0.0. Cycle Computing. Web. 22 Feb. 2016. <https://docs.cyclecomputing.com/cyclecloud/6.0.0/guide/schedulers.html>.

[4] "Customization." Customization — CycleCloud User Guide V6.0.0. Cycle Computing. Web. 22 Feb. 2016. <https://docs.cyclecomputing.com/cyclecloud/6.0.0/guide/customization.html>.

[5] "Using Spot Instances." Using Spot Instances — CycleCloud User Guide V6.0.0. Cycle Computing. Web. 22 Feb. 2016. <https://docs.cyclecomputing.com/cyclecloud/latest/guide/aws_configuration.html#amazon-placement-groups>.

[6] "Installing CycleCloud." Installing CycleCloud — CycleCloud User Guide V6.0.0. Cycle Computing. Web. 22 Feb. 2016. <https://docs.cyclecomputing.com/cyclecloud/latest/guide/server_install.html>.

[7] "CycleCloud Overview." Cycle Computing. Cycle Computing. Web. 22 Feb. 2016. <http://cyclecomputing.com/products-solutions/>.

[8] "CfnCluster." Amazon Web Services, Inc. Amazon Web Services. Web. 23 Feb. 2016. <https://aws.amazon.com/hpc/cfncluster/>.

[9] "Getting Started with CfnCluster." Getting Started with CfnCluster — CfnCluster 1.0.1. Amazon Web Services. Web. 23 Feb. 2016. <http://cfncluster.readthedocs.org/en/latest/getting_started.html>.

[10] "What Is AWS CloudFormation?" AWS CloudFormation. Amazon Web Services. Web. 23 Feb. 2016. <http://docs.aws.amazon.com/AWSCloudFormation/latest/UserGuide/Welcome.html>.

[11] "CfnCluster Auto-scaling." CfnCluster Auto-scaling — CfnCluster 1.0.1. Amazon Web Services. Web. 23 Feb. 2016. <http://cfncluster.readthedocs.org/en/latest/autoscaling.html>.

[12] "StarCluster." STAR: Cluster. MIT. Web. 23 Feb. 2016. <http://star.mit.edu/cluster/>.

[13] "What Is StarCluster?" StarCluster 0.95.6 Documentation. MIT. Web. 23 Feb. 2016. <http://star.mit.edu/cluster/docs/latest/overview.html>.

[14] "Plugin Documentation." Plugin Documentation — StarCluster 0.95.6 Documentation. MIT. Web. 23 Feb. 2016. <http://star.mit.edu/cluster/docs/latest/plugins/index.html>.

[15] "Elastic Load Balancer." Elastic Load Balancer — StarCluster 0.95.6 Documentation. MIT. Web. 23 Feb. 2016. <http://star.mit.edu/cluster/docs/latest/manual/load_balancer.html#>.

[16] "Fedora." Red Hat Enterprise Linux. Fedora. Web. 2 Mar. 2016. <https://fedoraproject.org/wiki/Red_Hat_Enterprise_Linux?rd=RHEL#History>.

[17] "Globus Connect Server." Globus Connect Server. Globus. Web. 2 Mar. 2016. <https://www.globus.org/globus-connect-server>.

[18] "Placement Groups." Amazon Elastic Compute Cloud. Amazon Web Services. Web. 2 Mar. 2016. <http://docs.aws.amazon.com/AWSEC2/latest/UserGuide/placement-groups.html>.

[19] "Amazon Simple Storage Service (S3) - Object Storage." Amazon Web Services, Inc. Amazon Web Services. Web. 2 Mar. 2016. <https://aws.amazon.com/s3/>.

[20] "Bottle: Python Web Framework." Bottle: Python Web Framework — Bottle 0.13-dev Documentation. Web. 2 Mar. 2016. <http://bottlepy.org/docs/dev/index.html>.

[21] "Dojo Toolkit Reference Guide." Dojo Toolkit Reference Guide — The Dojo Toolkit. Web. 2 Mar. 2016. <https://dojotoolkit.org/reference-guide/1.10/>.

[22] "HPCC." HPCC. Web. 19 Mar. 2016. <http://icl.cs.utk.edu/hpcc/index.html>.

[23] Roberto R. Expósito, Guillermo L. Taboada, Sabela Ramos, Juan Touriño, Ramón Doallo, Performance analysis of HPC applications in the cloud, Future Generation Computer Systems, Volume 29, Issue 1, January 2013, Pages 218-229, ISSN 0167-739X, http://dx.doi.org/10.1016/j.future.2012.06.009.

[24] "Enabling Enhanced Networking on Linux Instances in a VPC." Enabling Enhanced Networking on Linux Instances in a VPC. Web. 19 Mar. 2016. <http://docs.aws.amazon.com/AWSEC2/latest/UserGuide/enhanced-networking.html>.

[25] "IPerf - The Network Bandwidth Measurement Tool Active Measurements in TCP, UDP and SCTP." IPerf. Web. 19 Mar. 2016. <https://iperf.fr/>.

[26] Rabenseifner, Rolf, Sunil R. Tiyyagura, and Matthias Mueller. "Network bandwidth measurements and ratio analysis with the HPC challenge benchmark suite (HPCC)." Recent Advances in Parallel Virtual Machine and Message Passing Interface. Springer Berlin Heidelberg, 2005. 368-378.

[27] Carlyle, Adam G., Stephen L. Harrell, and Preston M. Smith. "Cost-effective HPC: The community or the Cloud?" Cloud Computing Technology and Science (CloudCom), 2010 IEEE Second International Conference on. IEEE, 2010.

[28] "Cloud Compliance - Amazon Web Services (AWS)." Amazon Web Services, Inc. Amazon Web Services. Web. 19 Mar. 2016. <https://aws.amazon.com/compliance/>.

[29] "Amazon CloudWatch - Cloud & Network Monitoring Services." Amazon Web Services, Inc. Amazon Web Services. Web. 19 Mar. 2016. <https://aws.amazon.com/cloudwatch/>.

[30] "EC2 Instance Types – Amazon Web Services (AWS)." *Amazon Web Services, Inc.* Amazon Web Services. Web. 3 Mar. 2016. <https://aws.amazon.com/ec2/instance-types/>.