Cloud Cluster Report

By: Michael DeProspo, Nathaniel Law, Thomas Seeley, and Madison Jones

April 29, 2021

1 BACKGROUND

Cloud computing, a growing field in computer science, is the process of using remote machines for your own purposes through virtualization. Given the growing prevalence of cloud computing, it may make sense for JMU to move away from using a cluster physically housed and maintained on campus, like the one that is currently in use for the needs of classes like cs470, and use a cloud computing service instead. This would serve the purpose of giving CS students more exposure to cloud computing, as well as remove the need to purchase and maintain another parallel cluster when the current one expires. In order to act as an adequate replacement, the use of a cloud computing service would need to be comparable to the use of the cluster that is currently utilized. This means it would need to be configured to run parallel and distributed programs using 1 or more nodes. For cloud computing to be a viable alternative to a cluster physically housed on campus, students would need to be able to use it without too much training, as 470 students now need to start using the cluster for labs and projects within the first few weeks of class. Our project's goal was to determine the feasibility of making the change to cloud computing services by attempting to emulate the capabilities of the current cluster using commercial cloud computing services.

2 METHODS

Our methods to set up alternatives to the JMU high-performance cluster were to create clusters using two different cloud providers, one cluster through Amazon Web Services (AWS) and one cluster through Google Cloud Platform (GCP). Our first approach to using these services to try to get a parallel cluster setup was to complete two lab activities that Dr. Lam provided us, in which we set up virtual clusters with 4 nodes on both AWS and GCP. These clusters were non-scaling but it was a good way for us to get introduced to using the technologies. After that, we started diving deeper into AWS and we did research on the many services that they offer to try to find a service they offer that would allow us to set up a cloud cluster that would be very similar in functionality to the JMU HPC, one that is auto-scaling, can use a job-scheduler like Slurm, and can run parallel and distributed programs using API's like MPI and OpenMP. We found that they offer a service called ParallelCluster that worked very well for what we were trying to accomplish. Setting up a cluster using this services incurs charges from AWS, because you have to create an EC2 instance and use a computing resource called the Elastic Computing Cloud that they provide to keep the instance running, so we configured our student accounts to use the AWS free tier in order to set up our cluster for free. Then we each individually did a lot of research and watched tutorials on how to set up a cluster using AWS ParallelCluster. During this we all documented issues and breakthroughs that we had along the set-up process, meeting up frequently to debug to get each other

unstuck and also to discuss our progress. During these meetings we would talk about what we have tried and walk-each other through on video the different solutions we found to the problems that we were having. Eventually we all were able to set up our own cloud cluster and then we created a comprehensive step-by-step tutorial with pictures that would allow people brand new to using AWS a complete guide on how to set up a parallel and distributed cloud cluster that could be used to submit jobs to run their programs.

After we created the tutorial and were all comfortable with AWS ParallelCluster, we decided to move on to the other cloud provider that we wanted to set up a cluster on, GCP. We essentially replicated the exact process that allowed us to be successful using AWS and repeated it using GCP. Like AWS, using GCP to set up a cloud cluster and then using it to run jobs will incur charges but we were able to avoid getting charged due to Dr. Lam kindly helping us out and redeeming coupons for us that we could use on our student accounts to have enough credits (a currency on GCP) that gave us ample computing time. Then we all individually worked to find ways to use GCP to set up clusters that function how the JMU HPC operates, and found two possibilities that met our needs. One method of setting up a cluster on GCP was to use a service they offer on the GCP console called Marketplace, using which you can set up an HPC cluster on the cloud by deploying a configuration of a slurm cluster they offer called SchedMD-Slurm-GCP. We created projects using the GCP console and then from that project went to the Marketplace service to deploy clusters using SchedMD-Slurm-GCP associated with our individual projects. It took about 10 minutes for the deployment to fully complete, and then once it is finished they have a button that becomes activated called “SSH TO SLURM LOGIN NODE” and it opens up a shell that has you ssh’ed to the cluster that you just created. To our amazement after that we could just start running jobs using slurm just like the JMU HPC, so it is a pretty out-of-the-box solution that has what we needed. Then we tried another way of using GCP to set up the cluster using a service on their console that they offer called Terraform. We again created projects and navigated to Terraform and then had to enable GCP’s Compute Engine API for our project. Then we used their cloud shell and ran a series of commands that clones a SchedMD slurm cluster configuration depository from Github, and then initializes it and deploys it using Terraform. It was a much more in-depth process than using Marketplace but it also provided a higher degree of customization to our clusters. After some discussion and collaborative debugging on how to set up an HPC with Terraform on GCP we made a comprehensive tutorial for creating clusters on GCP like the one we had created for AWS.

3 EXPERIMENTS

Most of our experimentation in the process of setting up HPC’s on both AWS and GCP came from the extensive amount of debugging we had to do to get our clusters functioning, and also from indulging in the amount of variety in configuration these services offer in order to optimize the set-up process we describe in our tutorials. To get the best experience and to avoid as many errors as possible in setting up our clusters using AWS ParallelCluster we found that we needed many prerequisite technologies installed such as Python 3, and upgraded versions of pip, virtualenv, and aws-parallelcluster installed. When creating a group for our clusters on AWS, we originally tried adding a wide variety of different permissions and when it was successful we

mistakenly assumed that we needed all the permissions in order to get the cluster completely functional. However we found later on that we really only needed the AdministratorAccess permission, which automatically gives many of the other permissions, and that cut down on the time it takes to set up the cluster using our tutorial by a lot. We also found for AWS ParallelCluster that when you run the "pcluster configure" command the first time, it creates a hidden directory in the folder from where you called the command named .parallelcluster, and inside that folder is a configuration file which records the values that you enter after calling pcluster configure. This provides a nice of changing your cluster after creating it because you can just quickly manually edit this file with the new values that you want like awsbatch instead of slurm for the scheduler key and then do pcluster update <name of your cluster> which will make the changes to your cluster that you reflected in the file. We also ran into problems with this however because we found that after creating a cluster some of the values are dangerous to change such as changing the value associate with the base_os key, i.e: changing from alinux2 to ubuntu1804 as the cluster had already been created with a certain operating system. Through our experimentation with this file we found that the pcluster update command first analyzes changes like this and gives you warnings if they are unsafe like the one listed above, but the user still has the option to force update the unsafe changes anyway by running pcluster update with the --force flag. Another thing we found with AWS ParallelCluster is how to upload files locally into our created clusters using the secure copy protocol. This became necessary for us to figure out because once we finally got our clusters working successfully we had to test them by submitting jobs with MPI programs and the like to make sure they actually function like the JMU HPC, but we didn't have any programs on our clusters to submit jobs with. We looked online and by trial and error we found that we could run the scp command in the shell with this format: scp -i key.pem /path/to/myfile.txt ec2-user@IPAddressOfEC2. One of the best ways for us to make our tutorials better through experimentation was to write out what we had so far, and then set up clusters by following the steps we had written down, incrementally refining wherever we thought there was any ambiguity that might prevent people new to the technology from being able to set up their own clusters.

4 RESULTS

With the AWS platform we were able to create cloud clusters using the AWS Batch and the Slurm schedulers. The platform allowed the users to submit jobs using a virtual machine on their local computer's terminal and then SSH into the login node for the cloud cluster they had created. AWS also allowed for an auto-scaling feature to dynamically deploy and destroy nodes based on the job submission requirements.

We were able to test our AWS instructions on a JMU senior Computer Science student who had not taken CS 470 or used a cloud service before. This student was able to set up a cluster following our instructions with only a couple issues, however based on his feedback we made suggested revisions.

The GCP cloud platform was also successfully able to create and deploy a cloud cluster using the Slurm scheduler. The system configuration for the cluster was able to be created using either the google marketplace setup through SchedMD-Slurm-GCP or Terraform which also relied on the SchedMD-Slurm-GCP repository via github. The access for the cloud cluster could be made by using the Google Cloud Shell to SSH into the login node and submit jobs through that. Either form of setting up the Slurm scheduler allowed for auto-scaling to create and destroy nodes for jobs.

5 DISCUSSION

The creation and deployment of the AWS cloud cluster, although a valid option for the creation of a cloud cluster, did require a more complex setup and deployment process. One of the largest problems that came out of our experimentation was the discovery of how the AWS Educate accounts that we were able to get through the CS Department and Dr. Lam did not have the necessary permissions for the deployment of a cloud cluster using AWS's Parallel Cluster tool. Once this was discovered we did have to create our own AWS accounts using their free tier of service. If JMU or the CS470 course chooses at some point to switch over to an AWS cloud cluster for their HPC this would require that either students are directly given credits to apply to a full AWS account or that the school manages a cloud cluster and students are given user permissions on a login node.

For the AWS cluster it seems very feasible for students with little to no background to be able to create and deploy their own cloud cluster, however it seems that more of a problem comes from students or users following the tutorial who have used the system before. The tutorial is written generally assuming that the user is creating a cloud deployment and using AWS for the first time after some troubleshooting we did have to include specific notes to help negate any errors that might occur such as having them delete any old pcluster deployments, updating their SSL Keys, and ensuring the newest version of the AWS Parallel Cluster is running. Despite these problems that we ran into the AWS Parallel Cluster tool does allow for a lot of flexibility and customization in how the system is configured. Parallel Cluster allows for the user to easily choose things such as compute node type, the login node's virtual OS, and the scheduler which creates a large variety of options for being able to do experiments and projects.

For the GCP deployment we found two options to create a cloud cluster using the Slurm scheduler. During the beginning of the experimentation process we initially worked on deploying a cloud cluster using Terraform as the configuration and deployment tool. While running our initial experiments we worked to create the solution but found that the deployment would fail if configured with the f1-micro (for free tier usage) compute nodes. We were unable to find why this problem was caused other than that the Slurm deployment just did not support the f1-micro nodes. The solution to this problem was just to run it with the default node type and just use our GCP Education credits for any costs that would be incurred on our accounts. Another small problem that we ran into with GCP was sometimes the Terraform platform would be unable to verify the signature of the configuration with google. We found that this problem probably came from deploying and destroying many different Terraform cloud clusters and projects, and the system was either blocking usage for a small period of time or not being able to

catch up with the rapid deployment as quickly. However, this problem could be solved by waiting for a period of time, usually just overnight or ~10-12 hours.

The initial complexity and signature problem from the Terraform deployment led to looking for a simpler solution, as the focus of our project was to allow for a student with little to no background knowledge of cloud computing to deploy their own cluster. Upon finding the SchedMD-Slurm-GCP tool in the google marketplace we found that it was significantly easier to create and deploy a cloud cluster from that than it was with Terraform. Initially we planned on removing the Terraform deployment from the instructions completely, but ended up saving it because as users become more advanced and comfortable with the cloud platform the basic.tfvars variable file used for deployment offers a significant amount of customization that might be helpful for users with specific needs for running jobs. Overall for GCP we found that using the SchedMD-Slurm-GCP tool on Google Cloud Marketplace offered by far the most feasible way for new students and users to be able to create and deploy a cloud cluster for the first time.

Throughout the entirety of both parts of our project we wanted to have the ability to create a cloud cluster that offered auto-scaling. All of the solutions proposed in our instructions are set up with the auto-scaling feature for compute nodes. The auto-scaling works similarly for all both of the platforms in that as jobs are submitted it is able to spin up the required number of nodes to run, then once the nodes return to their idle state there is a relatively short delay, generally 20-40 minutes, and the compute nodes are destroyed. When no jobs have been submitted with GCP it keeps two nodes active, the controller and login nodes. For AWS when no jobs are in the queue it will only keep the head node active.

One advantage of this is that because there are only one or two nodes that are continuously active it is able to limit the billing for the usage of each respective platform's compute nodes. However, the use of the auto-scaling did create a small problem in the functionality of usage. Jobs can be run directly using the salloc or srun commands but because of the overhead time that it takes to create new nodes as necessary we did find the most efficient way to run jobs was to submit them using an sbatch script.

6 CONCLUSIONS

Based on our research it would be reasonable to conclude an incoming CS 470 student would be able to set up a cloud cluster on either AWS or GCP, however GCP seems to be the easier route. We found GCP to be much easier to work with and the SchedMD-Slurm-GCP tool on GCP marketplace allows for a trivial setup of an auto-scaling cluster. In addition to this, a GCP cluster can be set up without a credit card, unlike AWS, many students might not have or have access to a credit card.

With all this in mind, we would recommend GCP to any educational institution or instructor who is exploring alternatives to a local high performance cluster for student usage.

7 FUTURE WORK

While we were able to set up auto-scaling clusters using both providers, we were only able to test our instructions with one student who has not taken CS 470. If someone was interested in exploring these

further it would be ideal to poll at least 10-20 more students to determine what issues arose and how many were successfully able to set up a cloud cluster as well as use it. In addition to this, we did not examine the cost or performance benchmarks between these cloud providers and JMU's HPC. A previous group did and found that a cloud provider would be cheaper to use[1]. Given that cloud services are continually getting cheaper, this is an even bigger incentive for institutions and educators to look into switching from local to cloud based HPC's. The other aspect we would suggest to explore is the control and customizability of a cloud based cluster being used by students vs a local institutional HPC. Support for students might not be able to be provided by instructors or IT staff, rather by the cloud provider support, which many educational institutions might have an issue with. We did not attempt to change configurations of the clusters we made beyond size and compute size, but there are many ways to configure them which would be worth looking into.

8 REFERENCES

1. Lee, Andrew, et al. *Cloud Services vs Local Clusters*. James Madison University, 2019.