Designing a Morehouse Supercomputer

Morehouse College HPC Research Group

Zachare Lofton
Computer Science
Morehouse College
Atlanta, Georgia, USA
zachare.lofton@morehouse.edu

Trent Gaylord
Computer Engineering
Morehouse College
Atlanta, Georgia, USA
Trent.Gaylord@Morehouse.edu

Craig Edelin Jr
Computer Science
Morehouse College
Atlanta, Georgia, USA
craig.edelin@morehouse.edu

Kevin Drew
Software Engineering
Morehouse College
Atlanta, Georgia, USA
kevin.drew@morehouse.edu

Dr. Alfred Watkins
Computer Science
Morehouse College
Atlanta, Georgia, USA
alfred.watkins@morehouse.edu

ABSTRACT

Currently, Morehouse operates in a bring-your-own-device (BYOD) environment, where students are expected to download and use their personal devices to perform in Computer Science and related classes. This creates various issues with both incoming and current students. More specifically, we observed that students were having issues with limited resources for downloading and running specific software and programs. Furthermore, computer science and related programs lack computing resources for conducting research beyond the faculty devices, drastically limiting our ability to demonstrate our capacity to perform research efficiently. To mitigate this issue, we have decided to build a virtual machine that interacts with a cloud-based system, to grant students access to the tools that help them be successful which in turn should help improve our retention rate. We have performed research thus far on the tools and other resources needed to complete the work done for the implementation of the system. We have surmised that we will need a VM provisionary tool, a CPU, and a Virtualization tool to complete this mission. As a result of our research, we have succeeded in creating our server and compute node. There is still more work to be done, concerning security and migration to a new VM tool, so that we can provide a physical user interface. We would like to thank our sponsors for their contribution and support of the project: Microsoft, Title III, and Lockheed-Martin.

CCS CONCEPTS

•Networks~Network-services~Cloud-computing
•Networks~Network-services~Programmable-networks
•Networks~Network-components~End-nodes~Network servers•Networks~Network-components~Logical nodes~Network domains

KEYWORDS

High Performance Computing, Supercomputer, Linux

ACM Reference format:

Nana, T. (2020) Kubernetes tutorial for beginners [full course in 4 hours], YouTube. YouTube. Available at: https://www.youtube.com/watch?v=X48VuDVv0do (Accessed: February 16, 2023).

1 Introduction

Most schools operate in a “bring your own device environment”, and as someone may have guessed this can create its own set of problems. Generally speaking, these problems can range from issues with Operating System-to-applicational compatibility, or simply just a lack of computing resources. Furthermore, the Covid-19 epidemic has increased the need for a solution for this problem. After observing this our mentor, Dr. Alfred Watkins, proposed an idea that could help reduce the strain on the students and the institution. This idea was to build a supercomputer that would provide the students with the required environments needed to be successful in their classes.

This idea started as just building a supercomputer, but then as scholars we began to wonder how to make this more useful to the students. We then decided to evolve from just a simple supercomputer to building a cloud service system that can be used to provide students, researchers and professors with the ability to operate within the environment,

as well as provide the once in a lifetime experience, of working with a high performing computer without having to worry about having the right software/OS compatibility and the computer specifications needed to perform tasks and other scholarly duties. We hope that this can be useful to other scholars attempting to solve this issue in their own environments.

1.1 Overall Architecture

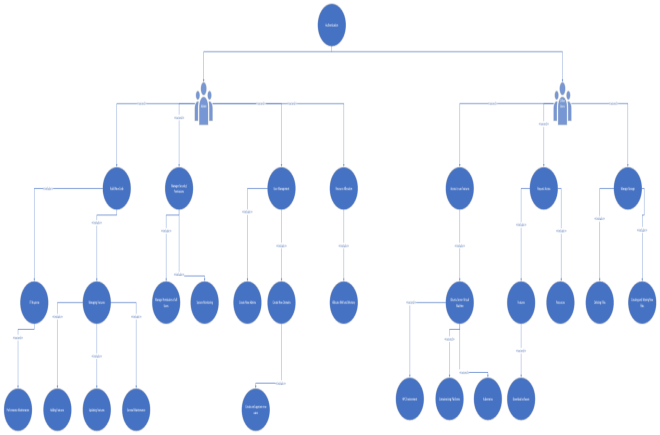

Figure 1: **This is an overall schematic of some of the key features we will have for the cloud system.**

The system will involve two sets of users: students and faculty/researchers. Super users will essentially work as researchers, as such they will have elevated privileges to monitor and maintain the environment that will be used by students. Some of the features that we are planning for the system to have include a command line interface, graphical user interface (**GUI**), and minikube. To help with students getting ready to enter industry or higher education, it is important for them to be aware of specific technologies to stay up to date and relevant. More features will be implemented at a later date, but these are the first features users should have access to.

The system will be designed using kubernetes (**K8s**), Docker, and OpenStack as the software components. Our goal is to have the user interact with the GUI and contact the server. Once on the server the user should then be allocated a standard amount of random access memory (**RAM**) and storage on a virtual machine (**VM**). If the user is a student they will have access to the tools permitted by the professor to perform the tasks needed for their work. If the user is a professor then they would be able to request resources from their portal to be distributed to their students. Furthermore, researchers will have access rights to create new features as well as update and monitor current features.

We want this system to be similar to that of current cloud computing systems such as Google Cloud, AWS, and Azure; except this software will be free for students to use without worrying about the cost.

1.2 Hardware

The hardware backbone that would provide the **RAM** for computations requested by the **VMs** students would consist of computation (**Compute**) Nodes, which would be assigned tasks from the individual ***VMs***. These Compute Nodes would require significant computing power, and the architecture of our Compute Nodes was one of the primary hurdles we had to overcome before development could proceed. The initial decision was between focusing our budget on Central Processing Units (**CPU**) or Graphics Processing Units (**GPU**).

The primary difference between CPU and GPU compute nodes is where the majority of the processing power is located and the tasks that they excel in. CPU Compute Nodes operate primarily as generalized processors. CPU Nodes focus on maximizing the number of ***instruction cycles*** within larger cores and optimizing the cache memory size. An instruction cycle is a unit representing the process where instructions are pulled from memory, decoded into processing language, and executed through the logical gates of the core. The increasing number of larger cores in individual CPU processors allows for many instruction cycles to operate simultaneously, increasing the flexibility and speed of individual CPUs. CPU Nodes focus on switching rapidly between hundreds of different operations per second. In contrast, GPU-based Compute Nodes offer similar amounts of cores & instruction cycles but offer a different approach to data processing. Instead of emphasizing cycle flexibility to manage multiple tasks, GPU acceleration emphasizes parallel processing through a comparably larger number of weaker cores. With our use-case set to provide access to a local cloud computing network to an increasing amount of users in the Department of Computer Science, the added flexibility of a CPU-based Compute Node was considered more viable than GPU-based Nodes.

Once we decided to proceed with a CPU-based Compute Node architecture, next was the decision regarding the processors' specifications within individual Nodes, precisely whether a Node should follow a singular or dual CPU configuration.

While singular CPU servers are the baseline for most server architecture, dual configurations provide several benefits for the High-Performance Computing use case. In

developing our physical architecture, two primary limiting factors were cost and physical footprint, with an additional goal of future expansion.

Our goal of hosting a local cloud service competed for space with the traditional server infrastructure already present at Morehouse College. Due to this environmental constraint, the 1:1 performance of a dual CPU server compared to a singular provided a significant boon, regardless of the increase in slots. Further, in a dual CPU configuration, the number of memory channels available on each CPU would allow for additional future expansion of RAM.

The AMD EPYC 7643 series of processors attracted our attention for its performance in HPC use cases, broad industry support, and established server infrastructure for dual CPU implementation. Based on the eight memory slots available in each EPYC processor, sixteen 32GB DDR4 3200 formed the compute node's memory for a total RAM of 512GB, with the potential for future expansion.

Despite the efficiency of the EPYC processor, dual processor configurations require additional energy compared to singular configurations. They are at risk of leaving one processor idle if work apportionment is not optimized. Latency due to non-uniform memory access (***NUMA***), where the two CPUs need to share resources in dual processor configurations, can be mitigated by partitioning workloads between the two CPUs. The downsides of a dual processor configuration can be mitigated by the software that allocates processing power via the apportionment of VMs.

1.3 Software Components

The first software utilized for the Morehouse Super computer was *Xen-tools*. ***Xen-tools*** is an open-source virtual machine **(VM)** provisioning tool that would allow the admin to allocate memory and storage to a virtual machine for users to connect and utilize its resources. Our goal is to allow students to utilize the resources on these virtual desktops remotely for researching capabilities. It was crucial that we identified a software that would allow us to create new VM's, and assign them to students and teachers at Morehouse College. On the other hand, upon testing, the implementation of *xen-tools* was not compatible with our needs. *Xen-tools* provisioned command line interface **(CLI)** VM's for users, and was not capable of setting up graphical user interface **(GUI)** VM interfaces. Since many students are not familiar with traversing a terminal this was a major issue that needed to be addressed. Furthermore, *Xen-tools* lacked documentation and was not one of the most up-to-date softwares. This posed not only a software issue, but also a security issue for users since the software was not kept updated. In addition, since the Super computer is not limited to the current, we wanted to ensure that the softwares utilized had some level of adaptability and documentation under the scenario that we may have some problem that may need to be debugged. By providing a service to students and faculty we wanted to make the user experience as easy and efficient as possible. After our hands-on experience with the *xen-tools* software we realized that it was not compatible with our goals and decided to switch to another software that would supply our needs.

After using ***Xen-tools*** we transitioned to a new provisioning tool known as ***xCAT. xCAT*** was a good tool for provisioning. While it accomplished the goals for the Super computer, during the set-up process it was not a simple set-up process. There were technical difficulties with the installation and what appeared to be compatibility issues. Since the documentation provided on the website appeared to be rather stringent, attempting to set-up the ***xCAT*** tool to be compatible with Ubuntu seemed to be very difficult. The learning curve of ***xCAT*** was very high, which prompted a switch to a new tool.

Next, we decided to transition our software to another open source software known as ***Openstack***. ***Openstack*** was a better software to utilize because it was all the services that we needed in one open-source project. ***Openstack*** allowed us access to provisioning tools, communication with Kubernetes and easy to follow documentation. Previously our struggle seemed to be identifying some form of compatibility between the operating system (Ubuntu) and the tools we were utilizing. Surprisingly enough, ***Openstack*** had its own compatible software with Ubuntu known as ***Microstack.*** What this will allow us to do so far is create virtual machines locally and then allow users to secure shell **(SSH)** into the created virtual machine. In addition, on the admin side we are provided with a management dashboard that makes organization of the virtual machines a lot more manageable. Furthermore, while currently we can only create a maximum of 10 machines per node, we plan to expand our range so that we can create a multitude of virtual machines for students to use in the future. A challenge that we currently face with the ***Microstack*** software is identifying how to allow users to **SSH** into the HPC network remotely so that students can connect from anywhere and still utilize the proper resources.

1.4 User Interface

The Morehouse HPC Server can be access through our website which has been hardcoded using HTML/CSS and Bootstrap. All code for the website is stored in a private GitHub repository only accessible to club members. Users will be able to access our documentation, information about the team, and most important the command line. Before user have full access to the site they are need to register an account for the Morehouse HPC website. After logging in, users will be able to access the CLI page of the website.

1.5 Use Cases

1. Students will be able access out CLI (Command-line Interface) after logging into the server. We have plans implement a Linux GUI (Graphical User Interface) afer the implementation of the CLI. Our website also documents how our server was developed and they will have access to this information as well. The students will have a set amount of storage and memory to work with will accessing our CLI.

2. Faculty will have access to view they're students activity status the server like the last time they logged in and how long they were connected. Faculty also have access to every feature the students are capable utilizing.

3. Admins will be able to view resource allocation and user statuses on the server. Admins will have the highest permissions within the HPC server.

ACKNOWLEDGMENTS

This work has been supported by a generous donation from Microsoft and seed grant from Title III. We would also like to give a special thanks to Shawn Brown and Ashley Ellis from Morehouse College's IT department and Sara Lambert from The university of Illinois.