ΕΘΝΙΚΟ ΚΑΙ ΚΑΠΟΔΙΣΤΡΙΑΚΟ ΠΑΝΕΠΙΣΤΗΜΙΟ ΑΘΗΝΩΝ

ΣΧΟΛΗ ΘΕΤΙΚΩΝ ΕΠΙΣΤΗΜΩΝ
ΤΜΗΜΑ ΦΥΣΙΚΗΣ
ΤΜΗΜΑ ΠΛΗΡΟΦΟΡΙΚΗΣ ΚΑΙ ΤΗΛΕΠΙΚΟΙΝΩΝΙΩΝ

ΔΙΑΤΜΗΜΑΤΙΚΟ ΠΡΟΓΡΑΜΜΑ ΜΕΤΑΠΤΥΧΙΑΚΩΝ ΣΠΟΥΔΩΝ ΗΛΕΚΤΡΟΝΙΚΗΣ ΑΥΤΟΜΑΤΙΣΜΟΥ

ΔΙΠΛΩΜΑΤΙΚΗ ΕΡΓΑΣΙΑ

Διαχείριση συστημάτων υψηλής απόδοσης

Abdul Hamid M. Omar

Επιβλέπων **Ιωάννης Κοτρώνης**, Καθηγητής

ΑΘΗΝΑ

ΣΕΠΤΕΜΒΡΙΟΣ 2021

ΔΙΠΛΩΜΑΤΙΚΗ ΕΡΓΑΣΙΑ

High Performance Cluster Management

Abdul Hamid M. Omar
Α.Μ.: 2016501

ΕΠΙΒΛΕΠΩΝ **Ιωάννης Κοτρώνης**, Καθηγητής

ΕΞΕΤΑΣΤΙΚΗ ΕΠΙΤΡΟΠΗ **Ρεΐσης Διονύσιος,** Καθηγητής
Πασχάλης Αντώνης, Καθηγητής

ΣΕΠΤΕΜΒΡΙΟΣ 2021

ΠΕΡΙΛΗΨΗ

Η παρούσα εργασία πάνω στο αντικείμενο των Συστημάτων Υψηλής Υπολογιστικής Απόδοσης (HPC Systems) κάνει μια εκτενή περιγραφή ολόκληρης της διαδικασίας εγκατάστασης ενός συστήματος HPC. Συγκεκριμένα, έγινε εγκατάσταση της Argo με χρήση των εργαλείων του πακέτου ανοιχτού κώδικα OpenHPC, σχηματίζοντας ένα Cluster που συνδυάζει 11 κόμβους και συνολικά 90 πυρήνες. Επιπλέον, περιγράφονται τα δομικά στοιχεία ενός Unix/Linux cluster, και η διαχείριση ενός υπολογιστικού συστήματος υψηλής απόδοσης.

Στο πλαίσιο της παρούσας εργασίας η μελέτη των HPC Συστημάτων περιέλαβε: διερεύνηση της λειτουργίας και οργάνωσής τους, διαδικασία εγκατάστασης και αναβάθμισης λογισμικού και ορθή διαχείριση και εκμετάλλευση πόρων. Συγκεκριμένα, έγινε αρχικά η αναβάθμιση των πακέτων του λογισμικού ohpc από την έκδοση 1.3.8 σε 1.3.9., έπειτα ενσωματώθηκε ο 11^{ος} κόμβος στο σύστημα και έγινε η εγκατάσταση των απαιτούμενων εργαλείων για τη μεταγλώττιση και εκτέλεση παράλληλων προγραμμάτων αξιοποιώντας δυο κάρτες γραφικών Nvidia, καθώς και ο ορισμός ουράς για την εκτέλεση των προγραμμάτων, χρησιμοποιώντας το λογισμικό OpenPBS.

Τα προηγούμενα βήματα, παρότι επαναλήφθηκαν καθ’ όλη τη διάρκεια της εργασίας, ήταν καθοριστικά στην απόκτηση γνώσεων επί διαφόρων πτυχών της διαδικασίας, βοήθησαν στην κατανόηση του μηχανισμού λειτουργίας του συστήματος, ενώ ήταν και αναγκαία για την ενημέρωση του συστήματος για αποφυγή τυχόν κινδύνων ασφαλείας, και αξιοποίηση της τελευταίας έκδοση λογισμικού που παρέχει το OpenHPC.

Το επόμενο βήμα ήταν η αναβάθμιση του συστήματος στην έκδοση OpenHPC 2 για το οποίο χρειάστηκε να γίνει η διαδικασία της εγκατάστασης εξ αρχής. Επομένως, έγινε εγκατάσταση του CentOS 8 στο server, εγκατάσταση των εργαλείων του OpenHPC και ρύθμιση του συστήματος που περιλαμβάνει: την εγκατάσταση των προγραμμάτων υποδομής, την ενσωμάτωση των κόμβων, την εγκατάσταση βασικών εργαλείων μεταγλώττισης και τη ρύθμιση εργαλείων επίβλεψης ορθής λειτουργία του συστήματος.

Παρακάτω, γίνεται αρχικά εισαγωγή στα HPC Συστήματα, όπου περιγράφονται τα βασικά στοιχειά ενός (HPC Systems), το λογισμικό και οι τεχνολογίες που συνδυάζουν για τη δημιουργία και διαχείριση HPC συστήματος. Επιπλέον, γίνεται εκτενής αναφορά στη διαδικασία εγκατάσταση της Argo, παρουσίαση προβλημάτων, τρόποι επίλυσης κτλ. Τέλος, γίνεται αναφορά στις βασικές εργασίες όπου μπορεί να χρειαστεί να επέμβει ο διαχειριστής του συστήματος, σε ξεχωριστά παραρτήματα.

ΘΕΜΑΤΙΚΗ ΠΕΡΙΟΧΗ: υπολογιστικά συστήματος υψηλής απόδοσης

ΛΕΞΕΙΣ ΚΛΕΙΔΙΑ: OpenHPC, Συστοιχίες Υπολογιστών, Παράλληλα Συστήματα,

Υπερυπολογιστής, OpenPBS

ABSTRACT

HPC Systems, support a wide range of applications. They have significant contribution in cases where parallelism can be introduced inside computation-intensive cases, allowing problem-solving and data generation in shorter times than using a single PC.

Aim of this thesis was to compile an extensive documentation related to the installation, building, management, and administration of an HPC System, through the OpenHPC open-source framework. A small cluster was formed comprised by a Master Server (Argo) and 11 computational nodes. An in-depth investigation of the system architecture, the required software and technologies used to setup and manage the cluster was performed. Emphasis was given in setting up and configuring Argo to properly handle all the nodes. Finally, some guidelines for System Administrators to address common issues are provided.

SUBJECT AREA: HPC Systems

KEYWORDS: OpenHPC, Cluster, Parallel Systems, Supercomputer, OpenPBS

ΕΥΧΑΡΙΣΤΙΕΣ

Θα ήθελα να εκφράσω τις θερμές μου ευχαριστίες στον καθηγητή μου κ. Ιωάννη Κοτρώνη, που με τις κατευθυντήριες γραμμές του και τις παραινέσεις του συνέβαλε ουσιαστικά στο παρόν έργο. Στη συνέχεια, θα ήθελα να ζητήσω ένα μεγάλο ευχαριστώ από όλο το διδακτικό προσωπικό του Τμήματος Ηλεκτρονικής Αυτοματισμού, το οποίο μου προσέφερε τις γνώσεις και τα απαραίτητα εφόδια για τη μετέπειτα επαγγελματική μου πορεία. Τέλος, ευχαριστώ το προσωπικό του τμήματος ΙΕΣΕ του ΕΚΠΑ, που στάθηκαν αρωγοί σε κάθε είδους δυσκολία που τυχόν αντιμετωπίσαμε.

ΠΕΡΙΕΧΟΜΕΝΑ

FOREWORD

Unix and now Linux, became prevalent in high-performance computing (HPC) systems, including systems at the very high end. Although these operating systems were not originally intended for scalable HPC usage, they were adapted to fulfill that role while preserving their ability to support a wide range of applications and use cases.

High Performance Computing (HPC) Systems, support a wide range of applications, in science and are used for various fields, where Big Data processing plays an important role, such as quantum mechanics, weather forecasting, climate research, oil and gas drilling and molecular modeling.

Commodity clusters provide a high-performance price ratio. Are easily scalable, with the possibility to increase the hardware components. All these advantages make it especially suitable for the university environment dedicated to run high-performance computing tasks.

Introduction

For the most part of the last decade the primary thrust of high-performance computing (HPC) development was in the direction of commodity clusters, commonly known as Beowulf clusters [13]. These clusters combine commercial off-the-shelf hardware to create systems that rival or exceed the performance of traditional supercomputers in many applications while costing as much as a factor of ten less, by benefiting from the economy of scale [1]. Not all applications are suitable for clusters, but a significant portion of interesting scientific applications can be adapted to them [1].

This thesis was motivated by the desire to design and implement a High-Performance Computing Cluster for teaching parallel computing theory and development of parallel applications, and to provide the researchers with the infrastructure to run their applications. Thus, the aim of this thesis was to setup and maintain a cluster (Argo [8]) comprised by a master server managing a number of working nodes. Thus, on the master server (argo) the OpenHPC framework was installed. OpenHPC is a community-based project providing an integrated collection of HPC-centric software components that can be used to implement a full-featured reference HPC compute resource [2]. Components that span the entire HPC software ecosystem including provisioning and system administration tools, resource management, I/O services, development tools, numerical libraries, and performance analysis tools [33].

This document is organized as follows: Section 1 deals with the overall clusters configuration, presenting the required hardware and software, the physical and logical layout of the systems, and basic operations. Section 2 presents in detail the cluster management tools, along with its administration process. Section 3 focuses on cluster implementation, presenting details related to software updating, adding new components, and the presented issues along with the adopted solutions. Section 4 presents the conclusions derived from all the work performed in this thesis. Finally, in the seven (7) appendices at the end of the present, details related to various administration and management actions are provided.

1. Cluster Architecture

A typical commodity cluster architecture requires a host (master node - SMS), a number of compute nodes (workers), an interconnecting network, and a storage unit. The master node hosts the base operating system (OS) and the required services to run and manage the clusters. The compute nodes are responsible for running the user's applications. Thus, the Argo cluster has:

- A master server (argo)
- 10 node computers (argo-cxx)
- Internal communication network between the compute nodes

resulting into a small Cluster combining 10 machines and a total of 80 cores. An additional node was added to the cluster with GPU computing capabilities rising the total nodes 11, having 90 cores and 2 GPU CUDA-enabled devices.

1.1. Node Architecture

Most of the decisions about node hardware were based on the available hardware architecture, cluster form factor, and network interface. The most important decision is the selection between single or multi-processor systems. Single processor systems have better CPU utilization due to the lack of contention for RAM, disk, and network access. Multi-processor systems reduce the communication overhead due to direct data sharing for hybrid applications. Furthermore, multi-processor systems tend to have higher performance external interfaces than single-processor systems [1]. Table 1 presents the hardware capabilities of Argo's computing nodes.

Table 1 Computing nodes hardware properties

	Property	Description
Dell Poweredge 1950 servers (10 nodes)	CPU	Two Quad-Core Intel-Xeon E5340 2.66 GHz processors
	RAM	4GB DDR2 667MHz
	GPU	
Dell Precision 7920 Rack	CPU	Intel-Xeon Silver 4114 2.2 GHz, 10Cores processor
	RAM	16GB (2X8GB) DDR4 2666MHz
	GPU	Dual Nvidia Quatro P4000, 8GB
Storage		Two identical 1TB RAID1 storage disks 120GB (X10), SSD W/R 540 MB/s - 560MB/s

1.2. Network Infrastructure

Like hardware architecture, the selection of network interfaces is a matter of optimizing and balancing between price and performance while considering the tasks the cluster is expected to perform. Performance is characterized by bandwidth and latency. For loosely coupled jobs with small input and output datasets, little bandwidth is required and 100Mbps Ethernet is an appropriate choice [1]. For tightly coupled jobs, Myrinet with its low latency and 2 Gbps + 2 Gbps bandwidth can provide an adequate solution. Other interfaces, such as upcoming InfiniBand products, provide alternatives for high-speed interfaces.

The recommended system architecture used for any HPC Cluster has several networks:

1) In-Band internal management Ethernet network. The In-Band management network is attached to all nodes in a system and available to each node’s host operating system. The purpose of the network is to provide remote access from the system’s login nodes and communication between HPC cluster system daemons.

2) Out-of-Band (OOB) power control and console Ethernet network. The OOB network is attached to all nodes in the system but is only available to the host operating system on the system’s admin/head nodes. The connection is used to communicate with BMC within each node for node power control and console access.

3) High-performance low-latency network or fabric, e.g., Mellanox InfiniBand or Intel Omni-Path for MPI and network file service. The high-performance low-latency network or fabric is attached to each computing node in the system for intra-job communication, usually via MPI.

4) A shared or optional dedicated network for parallel file system service in any HPC cluster, and NFS-based shared or high-performance parallel file system such as Lustre [9*] provides all nodes to access the shared file system [7].

The master host requires at least two Ethernet interfaces with **enp2s0** connected to the outside network and **enp0s31f6** used to provision and manage the cluster back-end. Two logical IP interfaces are expected for each compute node: the first is the standard Ethernet interface that will be used for provisioning and resource management [34]. The second interface is used to connect to each host’s baseboard management controller (BMC) and is used for power management and remote console access. (See figure: 1) Physical connectivity for these two logical IP networks is often accommodated via separate cabling and switching infrastructure; however, an alternate configuration can also be accommodated via the use of a shared NIC [2].

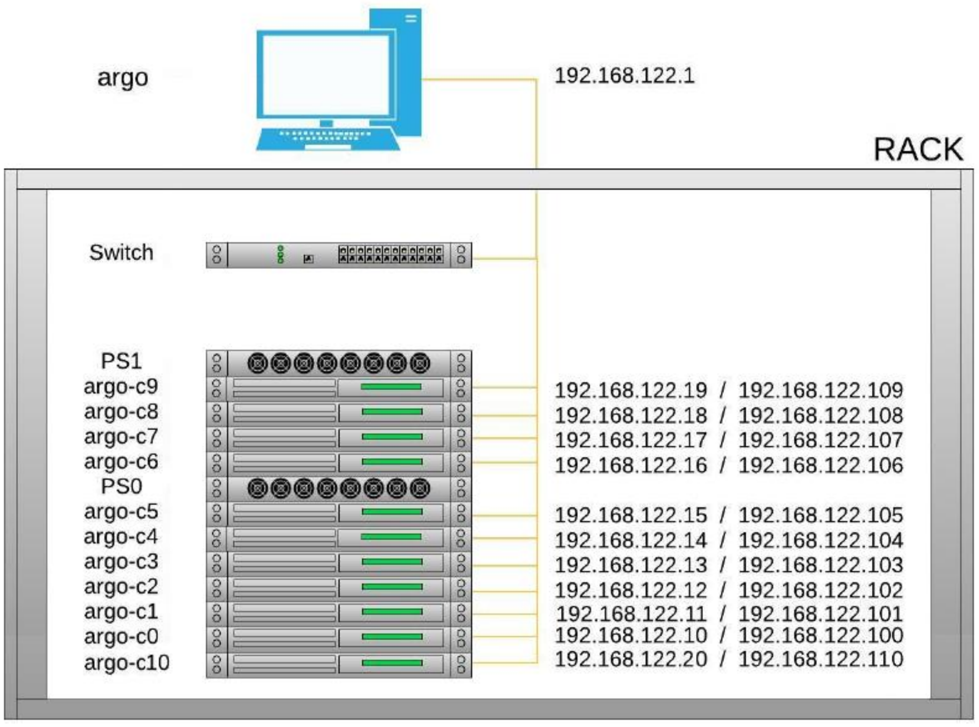


Figure 1: Network Overview [8]

For back-end communication, a TP-Link TL-SG1016 Switch is used since it represents an optimal choice for our applications [19]. TP-Link TL-SG1016 provides 16 10/100/1000Mbps RJ45 ports, with Supports MAC address self-learning and auto MDI/MDIX and Standard 19-inch rack-mountable steel case.

1.3. Core Servers and Services

In most cases, a single-core server suffices to provide all necessary core services. In fact, some clusters simply pick a node to be the nominal head of the cluster. Some large clusters provide multiple front ends, with load balancing and fail-over support to improve up-time. Core services are those services that need to be available for users to utilize the cluster. At a minimum, users need accounts and home directories. They also need a way to configure their jobs and get them to the nodes. The usual way to provide these services is to provide shared home and application directories, usually via NFS, and use a directory service such as NIS to distribute account information. Other core services a cluster architect might choose to include are batch schedulers, databases for results storage, and access to archival storage resources. The number of ways to allocate core servers to core services is practically unlimited [1].

Argo has three core servers: the data server, the user server, and the management server. All these servers are hosted on a Dell OptiPlex 7050 desktop PC which represents the cluster host. The host serves two identical 1TB RAID1 (Redundant Array of Independent Disks) connected for the system installation storage, and to users to store their applications and data.

1.4. Operating System

For OpenHPC packaged components compatibility, CentOS was chosen as the host's operating system. CentOS Linux is a community-supported distribution derived from sources freely provided to the public on Red Hat or CentOS git for Red Hat Enterprise Linux (RHEL). As such, CentOS Linux aims to be functionally compatible with RHEL. The CentOS Project mainly changes packages to remove upstream vendor branding and artwork. CentOS Linux is at no cost and is free to redistribute. Each CentOS version is maintained until the equivalent RHEL version goes out of general support. A new CentOS version is made available once a new RHEL version is rebuilt - approximately every 6-12 months for minor point releases and several years for major version bumps. The time to rebuild varies from weeks for point releases to months for major version bumps [15]. This results in a secure, low-maintenance, reliable, predictable, and reproducible Linux environment.

Since the host OS did not require special modifications, the host operating system was equipped with GNOME Desktop Environment/Applications, Development Tools, and System Administration tools to provide the best ability to manage and modify the cluster as needed [8].

1.5. OpenHPC Software

OpenHPC was launched initially in November 2015 and formalized as a collaborative Linux Foundation [12] project in June 2016. It is a community-driven project currently comprised of over 25 member organizations with representation from academia, research labs, and industry.

OpenHPC adopts a familiar repository delivery model with HPC-centric packaging in mind and provides customizable recipes for installing and configuring reference designs of computing clusters [2]. It is intended to make available current best practices and provide a framework for the delivery of future innovation in cluster computing system software. Its endeavors to adopt a repository-based delivery model similar to the underlying OS distributions commonly used as the basis for HPC Linux clusters. At present, OpenHPC is providing builds targeted against two supported OS distributions: CentOS7 and SLES12. The underlying package managers for these two distributions are yum and Zypper, respectively, and OpenHPC provides public repositories that are compatible with these RPM-based package managers.

The installation procedure outlined in current OpenHPC recipes targets bare-metal systems and assumes that one of the supported base operating systems is first installed on a chosen master host [2].

Argo uses the (Warewulf + PBS Professional) x86_64 Architecture. Installation recipe for the OpenHPC 1.3.9 version [10] and the (Warewulf + OpenPBS) x86_64 Architecture, installation recipe for the OpenHPC 2.0 version [9].

1.5.1. Software Components

HPC Linux Cluster Software Stack includes four major layers, which are the adaptive platform layer, operating system layer, system management layer, and service layer. The adaptive platform indicates the hardware platform for HPC infrastructures such as servers, networking, and storage. The operating system is mainly a Linux-based operating system with a shared NFS-based or parallel file system. The system management layer includes system deployment, system monitoring, and system operations. The service layer includes inbound and outbound networking; resource and workload management, MPI environment, compilers with Math libraries, and application frameworks.

The following software components are used in a modern HPC system:

1. *Hardware-based management software*, which is software tools used for specific hardware supports or management tools such as firmware updater and IPMI management tools.
2. *Base operating systems* like Red Hat Enterprise Linux, CentOS, and Scientific Linux-based operating systems or specific device drivers such as GPU, NIC, HBA, or OPA/InfiniBand OFED drivers.
3. *Run-time libraries and language tools*, such as C/C++/Fortran/Perl/Python compilers, or run-time libraries such as GLIBC and math libraries.
4. *Provisioning tools and configuration management system* for deploying OS and managing applications packages and their configurations, such as Warewulf and xCAT.
5. *Monitoring and reporting tools*, including system-wide In-band or Out-of-Band real-time system monitoring, event management, and reporting tools. In this case Ganglia and Nagios provide real-time web-based monitoring and flexible event management, respectively.
6. *Resource/workload manager and job scheduler* that is an integrated resource and workload management systems such as PBS Pro and SLURM
7. *Application frameworks for specific applications* such as NVIDIA CUDA, MPI (OpenMPI, MVAPICH2, Intel MPI), or Deep Learning Frameworks (Caffe, TensorFlow, or Theano).
8. *Optimized user-space applications* or industrial/vertical applications such as WRF, Quantum Espresso, VASP, or ANSYS Fluent [7].

1.6. Network Addresses and Naming Schemes

There are three basic approaches to allocating IP addresses in a cluster. For small clusters, many architects simply put all the machines on an existing network [1]. This has the advantage that no additional routing is needed for the nodes to access an external data source. The disadvantage is that it typically means the IP addresses do not correspond to physical objects, so it is hard to distinguish between machines. Additionally, not subnetting the cluster can make it too easy for inter-node communication to impact the rest of the network. The other two approaches involve placing nodes on their own subnet, either with public or private [RFC1918] addresses. Using public addresses has the advantage that with appropriate routers, cluster nodes can exchange data with arbitrary external data sources. On a subnet, IP addresses can be mnemonic to help administrators remember which machine a particular address belongs to [1]. The main disadvantage of using public addresses is that address space is becoming increasingly scarce and large allocations are difficult or expensive to obtain.

For back-end communication, the chosen IP for the host is 192.168.122.1, as for the compute nodes each was provided with the IP address 192.168.122.X (where x is equal to the number of the node + 10). The IP address for the BMC interface is 192.168.122.Y (where x is equal to the number of the node + 100) [8].

Choosing hostnames within a cluster is another issue faced by a cluster architect. The usual rules of host naming [RFC1178] apply to naming core servers. However, unless the cluster is exceedingly small and likely to remain so, a numerical naming scheme such as node00, node01, etc. is likely to be a better idea than trying to come up with a naming scheme that can handle hundreds of unique machines [1].

The cluster hostname is Argo [14], the compute nodes are named argo-cxx (where x is equal to the number of the node e.g., argo-c7) [8].

1.7. Hardware Components

Argo consists of the Master node (Dell OptiPlex 7050, Quad-Core Intel-Core i5-6500 @ 3.20GHz, 16GB DDR4 2.4GHz), 11 compute nodes in a rack for running parallel programs. The 10 Dell Poweredge 1950 servers [16] Each Dell Poweredge 1950 server provides two Quad-Core Intel-Xeon E5340 processors are the nodes argo-c[0-9]. One Dell Precision 7920 Rack (Intel Xeon Silver 4114 2.2 GHz, 10Cores with hyperthreading, 16GB (2X8GB) DDR4 2666MHz with Dual Nvidia Quatro P4000, 8GB) argo-c10.

To connect to the local network, a network card (NIC – Network Interface Controller) Broadcom NetXtreme II 5708, Gbit speed is used, as well (BMC – Baseboard Management Controller), compatible with IPMI 2.0 for remote control of the condition of each node, such as temperature, consumption, etc.

The two cards have different addresses but share a common ethernet port with packet sharing.

A high-speed network (10 Gbit) interconnects the nodes for data sharing between the nodes to boost performance.

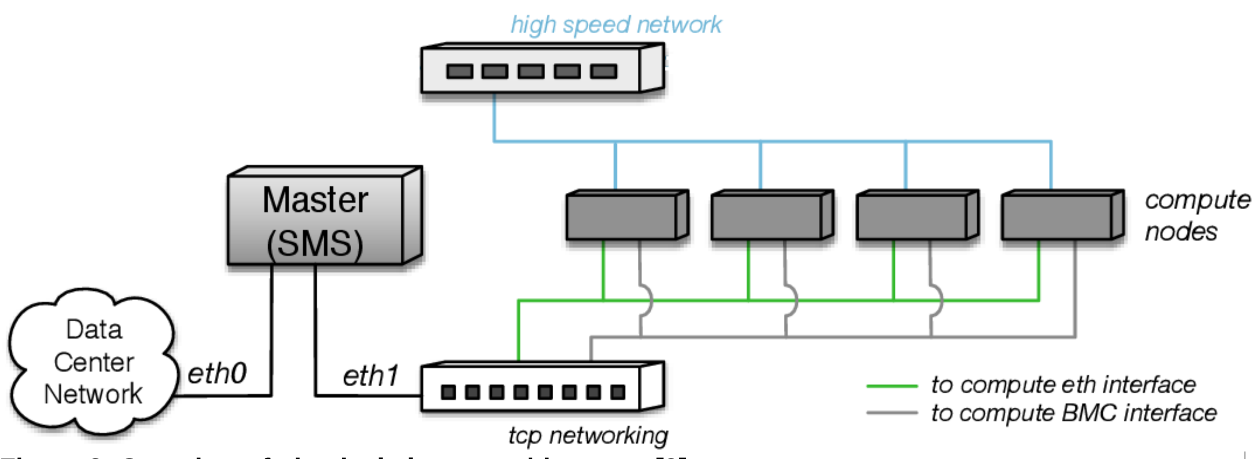


Figure 2: Overview of physical cluster architecture. [9]

2. Cluster Administration

This chapter provides information about the tools used in installing, configuring, and managing the cluster components, like nodes configuration management, job schedulers, and cluster monitoring tools.

2.1. Node Configuration Management

Since nodes generally outnumber everything else on the system, efficient configuration management is essential. Many systems install an operating system on each node and configure the node-specific portion of the installation manually. Other systems network boots the nodes using Etherboot, PXE, or LinuxBIOS. The key is a good use of centralization and automation. We have seen many clusters where the nodes are never updated without dire need because the architect made poor choices that made upgrading nodes impractical [1].

Warewulf

Warewulf is a Linux cluster solution that is scalable, flexible, and easy to use [20]. Warewulf facilitates the process of installing a cluster and long-term administration. This is achieved by changing the administration paradigm to make all the slave node file systems manageable from one point and automate the distribution of the node file system during node boot. It allows a central administration model for all slave nodes and includes the tools needed to build configuration files, monitor, and control the nodes. It is totally customizable and can be adapted to just about any type of cluster. From the software administration perspective, it does not make much difference if you are running 2 nodes or 500 nodes. The procedure is still the same, which is why Warewulf is scalable from the admin’s perspective. Also, because it uses a standard chroot-able file system for every node, it is extremely configurable and lends itself to custom environments very easily [4]. Warewulf allows for a single master-node to serve a root file system, kernel, and initial ram disk to any cluster node at boot time. It, therefore, implements the centralized cluster management scheme.

Warewulf utilizes the concept of a Virtual Node File System for the management of the node operating systems. Warewulf provides the ability to manage each node using a chroot which is a directory structure that represents a root file system. Building the VNFS image can be done on the master node or on a separate system and then imported into Warewulf master[11].

The warewulf-vnfs sub-package provides tools to create VNFS for different node operating systems. One such tool is the program wwmkchroot which uses modules to create chroots for different architectures. Modules can be developed to make chroots for just about any operating system in a simple and manageable manner [11].

By default, wwmkchroot downloads the required components for the chroot over the network, but as of Warewulf VNFS version 3.4, we can build a chroot image from the ISO/DVD images directly.

Package management and admin tasks can be performed nearly as easily as working on the host system. The tool that offers such convenience is wwmngchroot. Packages lists to remove and install can be issued at the same time, and a shell can be provided inside the chroot to perform manual tasks.

While the most configuration of provisioned images takes place in a simple chroot filesystem, such as that generated by wwmkchroot, these chroot cannot be directly provisioned by Warewulf. Instead, once you have configured the chroot to your satisfaction, you must use wwvnfs to encapsulate and compress this filesystem into a VNFS image which can be provisioned. The distinction is similar to that between the source and binary of a compiled program, where the chroot represents the source and the VNFS the binary [11].

To help improve performance and minimize consumed space in RAM, the wwvnfs tool provides a few mechanisms for excluding chroot files from the VNFS image. Excluded files remain a part of the chroot but are not included in the compressed VNFS which will be provisioned. There are several mechanisms for specifying files to be excluded.

The global vnfs.conf file allows you to specify an “excludes” line, which includes by default paths such as /opt and /usr/local. VNFS-specific configuration files can be placed in a /etc/warewulf/vnfs/[name].conf file and will be created by default if one is not present.

Optionally, the files you exclude can be hybridized. This means that rather than simply being removed from VNFS, they are replaced with a symbolic link with a prefix you specify using the –hybrid path option. For example, excluding /usr/local with a hybrid path /mnt/hybrid would replace the directory with a symbolic link to /mnt/hybrid//usr/local. This feature allows you to place some files in an alternative filesystem such as an external disk, an NFS mount, or other network filesystems. (Note, of course, that this path must be present in the fstab for your files to be accessible.) Hybridized locations can be a useful way to exclude seldom-used files from the VNFS, while still maintaining their accessibility and their presence in the chroot.

When a node is provisioned using Warewulf, at first it obtains a small bootstrap image from the master during the PXE process (transferred using TFTP). This bootstrap image includes a Linux kernel, device drivers, and a minimal set of programs to complete the provisioning process. Once the bootstrap is in place, it calls back to the master, downloads the VNFS capsule, and provisions the filesystem. It then calls init for the new filesystem to bring up the desired Linux system.

Since the same kernel is used in the bootstrap and the provisioned system, it’s important to make sure that the bootstrap kernel and device drivers match the VNFS being provisioned. Warewulf, therefore, allows you to create bootstrap images from a specific kernel, customizing them for use with a specific VNFS and/or machine. wwbootstrap is used to create these bootstrap images, bundling the device drivers, firmware, and provisioning software with the kernel. What kernel support is incorporated (kernel modules and firmware) is defined by the /etc/warewulf/bootstrap.conf.

Bootstrap images can be created using the kernels installed on the master node, or kernels present in a chroot. Using kernels from a separate chroot is especially useful if you are attempting to provision a Linux distribution different from the distro present on the master node; for example, provisioning a Debian node from a Scientific Linux master [11].

2.2. Job Scheduling

Job scheduling is potentially one of the most complex and contentious issues faced by a cluster architect. The major scheduling options are running without any scheduling, manual scheduling, batch queuing, and domain-specific scheduling. In small environments with users who have compatible goals, not having a scheduler and just letting users run what they want when they want or communicating with each other out of band to reserve resources as necessary can be a good solution. It has very little administrative overhead, and in many cases, it just works. With large clusters, some form of scheduling is usually required even if users do not have conflicting goals, it is difficult to try to figure out which nodes to run on when there are tens or hundreds available. Additionally, many clusters have multiple purposes that must be balanced. In many environments, a batch queuing system is an answer [1]. Similar setups exist, including PBS Professional, Slurm.

PBS Professional [26] is an industry-leading workload manager and job scheduler for HPC environments. PBS Professional Open-Source Project is open-source with community supports. Slurm [37] is an open-source, fault-tolerant, and highly scalable cluster management and job scheduling system for large and small Linux clusters. Slurm requires no kernel modifications for its operation and is relatively self-contained.

2.2.1. PBS Professional

PBS is a distributed workload management system that manages and monitors the computational workload on a set of one or more computers [40]. . It provides a set of daemons managing job processes. You can use PBS commands to manage PBS and to perform tasks such as submitting, querying, changing, and deleting jobs. PBS has a home directory defined in PBS_HOME environment variable, and PBS executables directory is set in PBS_EXEC. PBS can manage one or more machines. When PBS runs on several systems, normally the server (pbs_server), the scheduler (pbs_sched), and the communication daemon (pbs_comm) are installed on a “front end” system, and a MoM (pbs_mom) is installed and run on each execution host [31].

the following subsections introduce some of the basic aspects of PBS professionals.

<u>Server</u>

The server process is the central focus for PBS. It is generally referred to as the server or by the execution name pbs_server. All commands and communication with the server are via an Internet Protocol (IP) network. The server’s main function is to provide the basic batch services such as receiving/creating a batch job, modifying the job, protecting the job against system crashes, and running the job. Typically, there is one server managing a given set of resources [42].

The server contains a licensing client which communicates with the licensing server for licensing PBS jobs.

Job Executor (MoM)

The Job Executor is the component that places the job into execution. This process, pbs_mom, is informally called MoM as it is the mother of all executing jobs [28]. MoM places a job into execution when it receives a copy of the job from a server. MoM creates a new session that is as identical to a user login session as is possible. For example, if the user’s login shell is csh, then MoM creates a session in which .login is run as well as .cshrc. MoM also has the responsibility for returning the job’s output to the user when directed to do so by the server [27]. One MoM runs on each computer which will execute PBS jobs.

Scheduler

The scheduler, pbs_sched, implement the site’s policy controlling when each job is to run and on which resources [43]. The scheduler communicates with the various MoMs to query the state of system resources and with the server to learn about the availability of jobs to execute.

Communication Daemon

The communication daemon, pbs_comm, handles communication between the other PBS daemons [43].

Commands

PBS supplies both command-line programs that are POSIX 1003.2d conforming and a graphical interface. These are used to submit, monitor, modify, and delete jobs. These client commands can be installed on any system type supported by PBS and do not require the local presence of any of the other components of PBS [44].

There are three types of commands: User commands (available only to authorized users), operator commands, and manager (or administrator) commands. Operator and Manager commands require specific access privileges [31].

2.3. Nodes Monitoring Tools

The smooth operation of a cluster can be supported by the proper use of system monitoring tools. Most common monitoring tools such as Nagios and Big Sister are applicable to cluster use. The one kind of monitoring tool that does not work well with clusters is the sort that sends regular e-mail reports for each node. Even a few nodes will generate more reports than most admins have time to read. In addition to standard monitoring tools, there exist cluster-specific tools such as the Ganglia Cluster Monitor [1]. Most schedulers also contain monitoring functionality.

2.3.1. Nagios

The main purpose of system monitoring is to detect if any system is not working properly as soon as possible and notify the appropriate staff, and if possible, try to resolve the error. Such as by restarting system services if needed [45].

Nagios is an open-source tool for system monitoring. It constantly checks if other machines and the services on those machines are working properly. In addition, Nagios accepts information from other processes or machines regarding their status; for example, a web server can send information to Nagios if it is overloaded. System monitoring in Nagios is split into two categories of objects, hosts, and services:

- Hosts represent a physical or virtual device on the network (servers, routers, workstations, printers, etc.)
- Services are particular functionalities, for example, a SecureShell (SSH) server.

The simplified approach on Nagios performance checks by using only four distinct states (**Ok**, **Warning**, **Critical**, and **Unknown**) [45] is considered as a major advantage. It allows administrators to ignore monitoring values themselves and just decide on what the warning/critical limits are. This is a proven concept and is far more efficient than monitoring graphs and analyzing trends [32].

<u>**The Nagios web interface**</u>

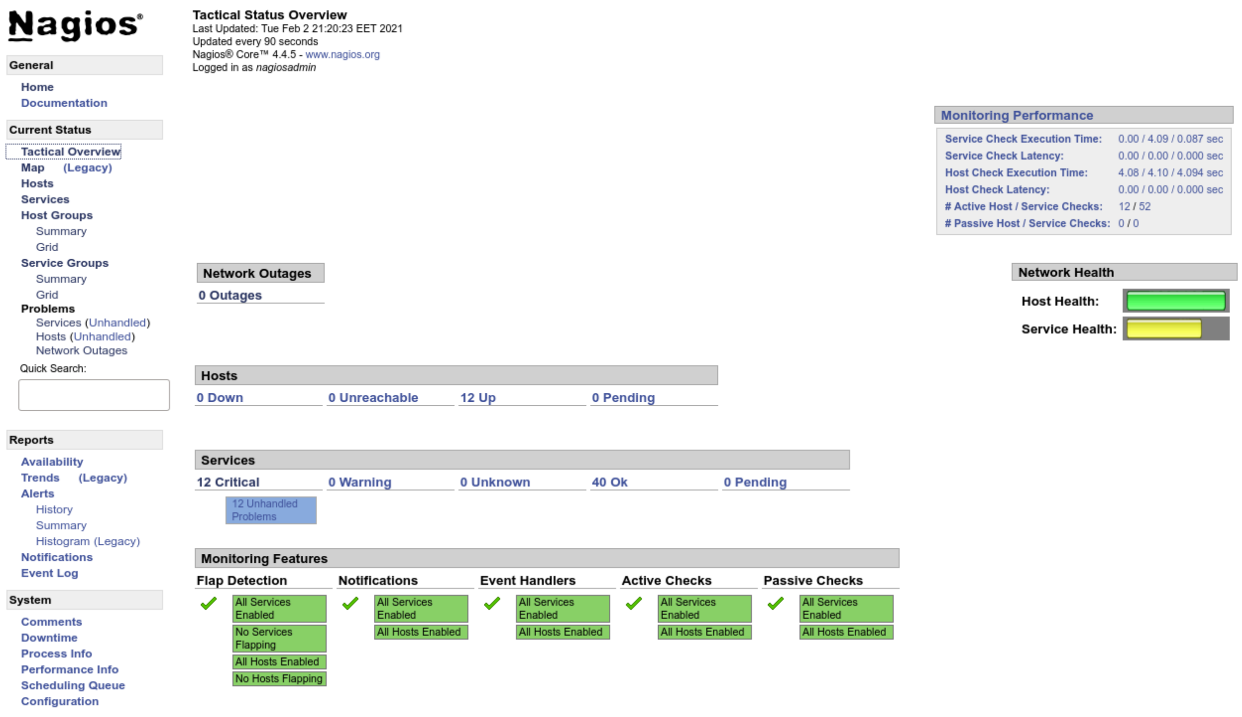
Nagios offers a web interface that can be used to view the status of all hosts and services, read logs, and generate reports. Using any browser, you can access almost any information that Nagios keeps, namely status, performance data, history, and logs. You can easily check if all your hosts and services are working correctly with just a few clicks. The interface also offers the ability to change parts of configuration on the fly.

Having the possibility to check the status of all hosts and services is a valuable functionality. Usually, a notification that something is wrong should just be a trigger to investigate the problem. Being able to see the big picture via various views of the web interface is extremely useful. Different detailed views can be used to see both what is working properly, and which hosts and/or services are in the warning or critical states [32].

<u>Tactical Overview</u>

Tactical overview presents the overall information on Nagios and monitoring. The page informs about the host and service conditions. It shows how many hosts and services are in which status. It also shows if any hosts and services have their checks, notifications, or event handlers disabled. Performance information is shown in the top right-hand corner. It shows details on checks that have been performed. It also reports latency when performing checks and the time that it takes to perform checks, on average.

Below the performance information, there is a status showing host and service health. It contains bars showing the number of hosts and services that are in an OK state. If all services are currently working properly then the bar spans to full width and is green. If some hosts or services are not working, then the color of the bar will change to yellow or red accordingly. Tactical Overview can also be used to observe hosts or services, filtered by specific criteria. Clicking on any status count text in the Network Outages, Hosts, or Services section will show a list of hosts or services with specified status; if we would click on the 16 Ok text in the Services section, it will show a list of all services with the OK status [32].



Tactical Status Overview
Last Updated: Tue Feb 2 21:20:23 EET 2021
Updated every 90 seconds
Nagios® Core™ 4.4.5 - www.nagios.org
Logged in as *nagiosadmin*

Monitoring Performance

Service Check Execution Time:	0.00 / 4.09 / 0.087 sec
Service Check Latency:	0.00 / 0.00 / 0.000 sec
Host Check Execution Time:	4.08 / 4.10 / 4.094 sec
Host Check Latency:	0.00 / 0.00 / 0.000 sec
# Active Host / Service Checks:	12 / 52
# Passive Host / Service Checks:	0 / 0

Network Outages
0 Outages

Network Health
Host Health:
Service Health:

Hosts

0 Down	0 Unreachable	12 Up	0 Pending

Services

12 Critical	0 Warning	0 Unknown	40 Ok	0 Pending
12 Unhandled Problems				

Monitoring Features

Flap Detection	Notifications	Event Handlers	Active Checks	Passive Checks
✓ All Services Enabled	✓ All Services Enabled	✓ All Services Enabled	✓ All Services Enabled	✓ All Services Enabled
No Services Flapping	All Hosts Enabled	All Hosts Enabled	All Hosts Enabled	All Hosts Enabled
All Hosts Enabled				
No Hosts Flapping				

Figure 3: Tactical Overview

status map

Nagios can show a graphical map of host parent-child relations along with the statuses. It can be accessed by clicking on the Map link in the menu on the left of the Nagios web interface. This can be used to keep track of hosts along with their statuses [32].

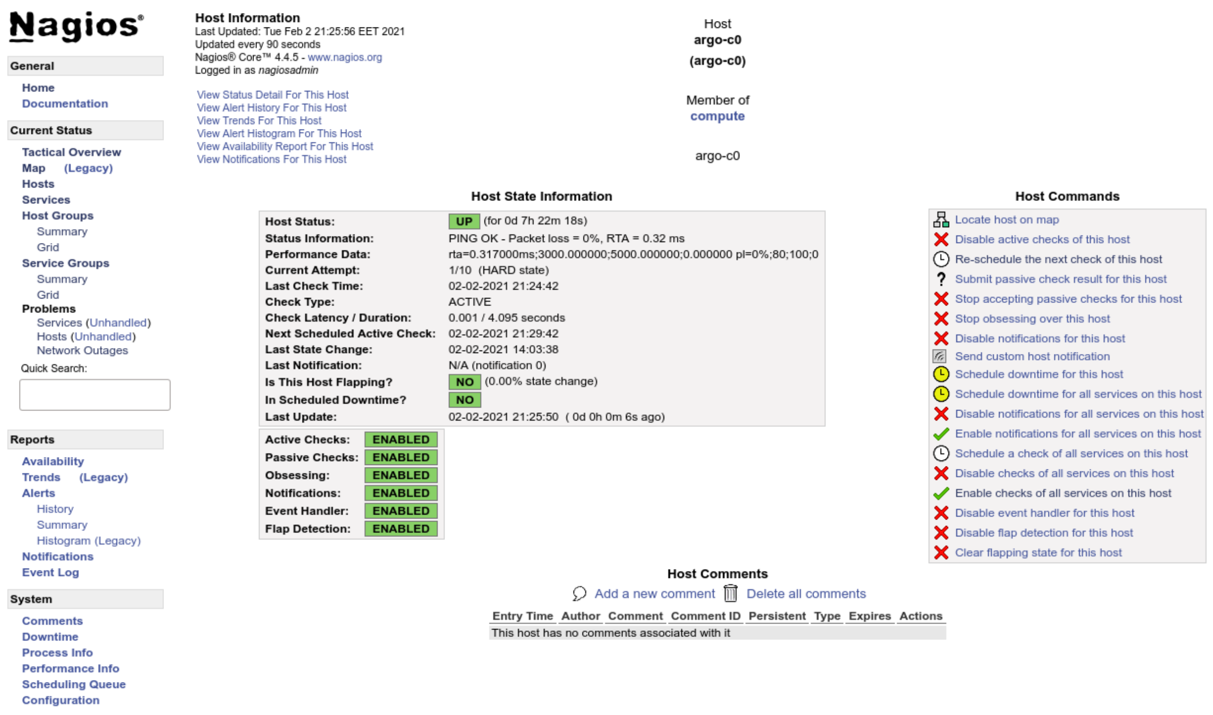
Checking status

Nagios offers a panel that shows all hosts along with their status. It can be accessed by clicking on the Hosts link in the menu on the left. Clicking on any host group description will show a list of all services on all hosts within that group. Clicking on a host group name, which is specified in brackets, will show a host group menu that allows modifying attributes for all hosts or services related to that host group. Clicking on a hostname in any host or service-related view will cause Nagios to show detailed information about the chosen host [32].

Viewing host information

Clicking on a host in any view of the web interface will take you to the Host Information page. It contains details about the current host status, a list of comments, and a commands panel that allows modifying the host configuration, scheduling checks, and sending custom notifications [32].

This page contains detailed information on the selected host. It shows the current status and host checks that have been or will be performed. It also contains information on what functionality is enabled or disabled for the specified host, whether the host is flapping along the flapping threshold value. The menu on the right can be used to perform operations related to this host. It allows toggling whether active checks should be performed. Nagios also can be configured to obsess over a host or send notifications and events. It is also possible to view options or schedule checks for all services bound to this host [32].



Host Information
Last Updated: Tue Feb 2 21:25:56 EET 2021
Updated every 90 seconds
Nagios® Core™ 4.4.5 - www.nagios.org
Logged in as *nagiosadmin*

View Status Detail For This Host
View Alert History For This Host
View Trends For This Host
View Alert Histogram For This Host
View Availability Report For This Host
View Notifications For This Host

Host
argo-c0
(argo-c0)

Member of
compute

argo-c0

Host State Information

Host Status:	UP (for 0d 7h 22m 18s)
Status Information:	PING OK - Packet loss = 0%, RTA = 0.32 ms
Performance Data:	rta=0.317000ms;3000.000000;5000.000000;0.000000 pl=0%;80;100;0
Current Attempt:	1/10 (HARD state)
Last Check Time:	02-02-2021 21:24:42
Check Type:	ACTIVE
Check Latency / Duration:	0.001 / 4.095 seconds
Next Scheduled Active Check:	02-02-2021 21:29:42
Last State Change:	02-02-2021 14:03:38
Last Notification:	N/A (notification 0)
Is This Host Flapping?	NO (0.00% state change)
In Scheduled Downtime?	NO
Last Update:	02-02-2021 21:25:50 (0d 0h 0m 6s ago)

Active Checks:	ENABLED
Passive Checks:	ENABLED
Obsessing:	ENABLED
Notifications:	ENABLED
Event Handler:	ENABLED
Flap Detection:	ENABLED

Host Commands

Locate host on map
Disable active checks of this host
Re-schedule the next check of this host
Submit passive check result for this host
Stop accepting passive checks for this host
Stop obsessing over this host
Disable notifications for this host
Send custom host notification
Schedule downtime for this host
Schedule downtime for all services on this host
Disable notifications for all services on this host
Enable notifications for all services on this host
Schedule a check of all services on this host
Disable checks of all services on this host
Enable checks of all services on this host
Disable event handler for this host
Disable flap detection for this host
Clear flapping state for this host

Host Comments

Add a new comment   Delete all comments

Entry Time	Author	Comment	Comment ID	Persistent	Type	Expires	Actions
This host has no comments associated with it							

Figure 4: host information

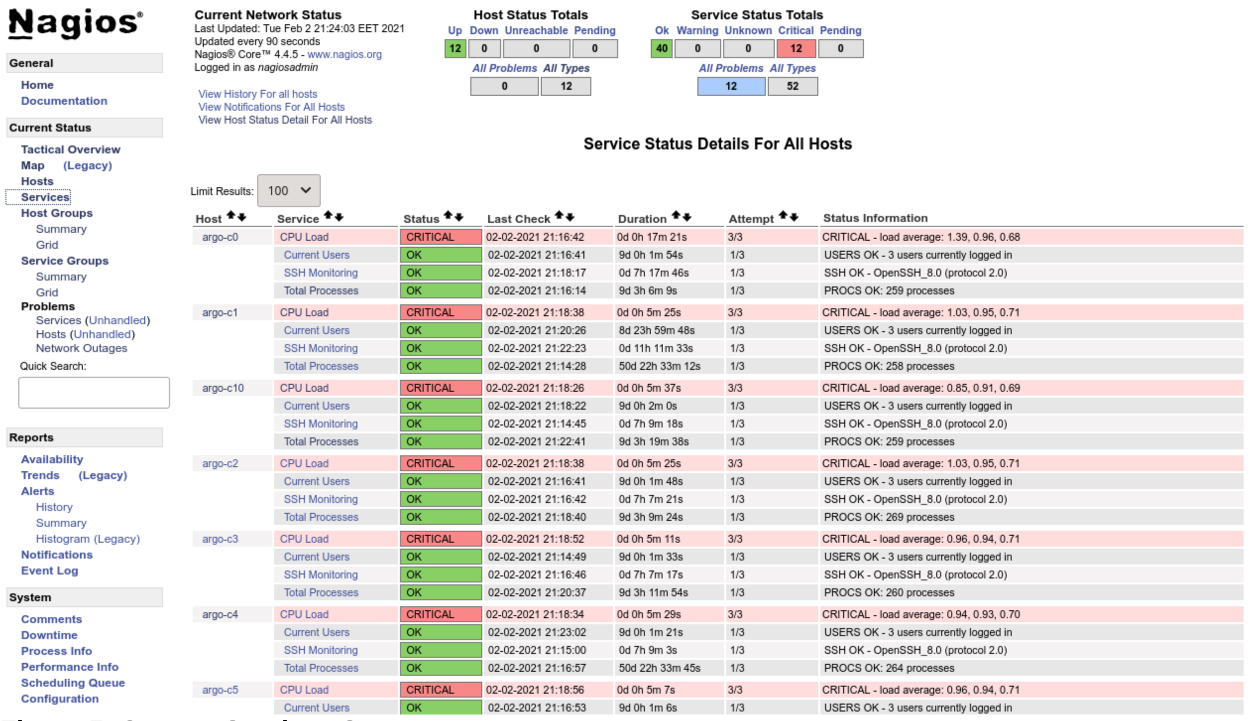
Managing services

Similar to the host-related information and operations, Nagios has similar panels for working with services. It consists of several service and service group views along with being able to view, detailed information on each service and modify its parameters.

Checking status

The Nagios web interface offers a view of all the defined services, their statuses, and basic information. It can be accessed by clicking on the Services link in the menu on the left. The main part of the page is the table showing all services along with their statuses and detailed information on the output from checks. The default order by which the table is sorted is that all services are grouped by the host they are configured for, and they are sorted by the service description.

Another interesting view is a summary of all services specified for each service group. It can be accessed by clicking on the Grid option under the Service Groups link from the menu on the left.



Current Network Status
Last Updated: Tue Feb 2 21:24:03 EET 2021
Updated every 90 seconds
Nagios® Core™ 4.4.5 - www.nagios.org
Logged in as *nagiosadmin*

View History For all hosts
View Notifications For All Hosts
View Host Status Detail For All Hosts

Host Status Totals

Up	Down	Unreachable	Pending
12	0	0	0

All Problems	All Types
0	12

Service Status Totals

Ok	Warning	Unknown	Critical	Pending
40	0	0	12	0

All Problems	All Types
12	52

Service Status Details For All Hosts

Limit Results: 100

Host	Service	Status	Last Check	Duration	Attempt	Status Information
argo-c0	CPU Load	CRITICAL	02-02-2021 21:16:42	0d 0h 17m 21s	3/3	CRITICAL - load average: 1.39, 0.96, 0.68
	Current Users	OK	02-02-2021 21:16:41	9d 0h 1m 54s	1/3	USERS OK - 3 users currently logged in
	SSH Monitoring	OK	02-02-2021 21:18:17	0d 7h 17m 46s	1/3	SSH OK - OpenSSH_8.0 (protocol 2.0)
	Total Processes	OK	02-02-2021 21:16:14	9d 3h 6m 9s	1/3	PROCS OK: 259 processes
argo-c1	CPU Load	CRITICAL	02-02-2021 21:18:38	0d 0h 5m 25s	3/3	CRITICAL - load average: 1.03, 0.95, 0.71
	Current Users	OK	02-02-2021 21:20:26	8d 23h 59m 48s	1/3	USERS OK - 3 users currently logged in
	SSH Monitoring	OK	02-02-2021 21:22:23	0d 11h 11m 33s	1/3	SSH OK - OpenSSH_8.0 (protocol 2.0)
	Total Processes	OK	02-02-2021 21:14:28	50d 22h 33m 12s	1/3	PROCS OK: 258 processes
argo-c10	CPU Load	CRITICAL	02-02-2021 21:18:26	0d 0h 5m 37s	3/3	CRITICAL - load average: 0.85, 0.91, 0.69
	Current Users	OK	02-02-2021 21:18:22	9d 0h 2m 0s	1/3	USERS OK - 3 users currently logged in
	SSH Monitoring	OK	02-02-2021 21:14:45	0d 7h 9m 18s	1/3	SSH OK - OpenSSH_8.0 (protocol 2.0)
	Total Processes	OK	02-02-2021 21:22:41	9d 3h 19m 38s	1/3	PROCS OK: 259 processes
argo-c2	CPU Load	CRITICAL	02-02-2021 21:18:38	0d 0h 5m 25s	3/3	CRITICAL - load average: 1.03, 0.95, 0.71
	Current Users	OK	02-02-2021 21:16:41	9d 0h 1m 48s	1/3	USERS OK - 3 users currently logged in
	SSH Monitoring	OK	02-02-2021 21:16:42	0d 7h 7m 21s	1/3	SSH OK - OpenSSH_8.0 (protocol 2.0)
	Total Processes	OK	02-02-2021 21:18:40	9d 3h 9m 24s	1/3	PROCS OK: 269 processes
argo-c3	CPU Load	CRITICAL	02-02-2021 21:18:52	0d 0h 5m 11s	3/3	CRITICAL - load average: 0.96, 0.94, 0.71
	Current Users	OK	02-02-2021 21:14:49	9d 0h 1m 33s	1/3	USERS OK - 3 users currently logged in
	SSH Monitoring	OK	02-02-2021 21:16:46	0d 7h 7m 17s	1/3	SSH OK - OpenSSH_8.0 (protocol 2.0)
	Total Processes	OK	02-02-2021 21:20:37	9d 3h 11m 54s	1/3	PROCS OK: 260 processes
argo-c4	CPU Load	CRITICAL	02-02-2021 21:18:34	0d 0h 5m 29s	3/3	CRITICAL - load average: 0.94, 0.93, 0.70
	Current Users	OK	02-02-2021 21:23:02	9d 0h 1m 21s	1/3	USERS OK - 3 users currently logged in
	SSH Monitoring	OK	02-02-2021 21:15:00	0d 7h 9m 3s	1/3	SSH OK - OpenSSH_8.0 (protocol 2.0)
	Total Processes	OK	02-02-2021 21:16:57	50d 22h 33m 45s	1/3	PROCS OK: 264 processes
argo-c5	CPU Load	CRITICAL	02-02-2021 21:18:56	0d 0h 5m 7s	3/3	CRITICAL - load average: 0.96, 0.94, 0.71
	Current Users	OK	02-02-2021 21:16:53	9d 0h 1m 6s	1/3	USERS OK - 3 users currently logged in

Figure 5: Current Services Status

Clicking on any status summary column will show a list of all services in that group along with detailed information. Clicking on a service group will show an overview of services split into individual hosts [32].

Managing downtimes

Nagios allows the use of the web interface to manage scheduled downtimes for hosts and services. This includes listing, adding, and deleting downtimes for both hosts and services.

The Nagios web interface lists all the scheduled downtimes. This page can be accessed by clicking on the Downtime link from the menu on the left. The page consists of two pages of all the scheduled downtimes, separately for hosts and services. Downtimes can be triggered by other downtimes. When a host downtime is scheduled, Nagios automatically adds downtimes for all child hosts.

Downtime can be scheduled from the Host Information or Service Information page by using the Schedule downtime for this host or Schedule downtime for this service option. It is also possible to use the Downtime page to schedule downtimes directly. In that case, you will need to know the hostname and service description of the service you want to disable as Nagios will not fill these in automatically.

The form consists of the host and service name, a comment, and an optional option list to choose a downtime that triggered this host/service to also be down. When specifying the time during which the downtime should be scheduled, it is possible to enter Start Time and End Time or use the Duration field. Scheduling downtime for a host is very similar; the only differences are that the Service field is not available, and the Child Hosts option list is added to specify how child hosts should be handled.

Nagios can automatically schedule downtimes for child hosts. When scheduling a host downtime, an additional option is present on whether child hosts should also be scheduled for downtime and set to be triggered by this downtime [32].

<u>Viewing process information</u>

The Nagios Process Information page shows generic information about the Nagios process. It also allows performing several actions from the Process Commands panel. This page can be accessed via the Process Info link from the menu on the left.

The page contains information on the Nagios version, process ID, status, and log rotation. It also shows whether checks, notifications, and many other functions are enabled. The menu on the right also allows us to stop and restart the Nagios daemon. It also allows us to enable and disable performing checks by sending notifications. Flap detection and performance data processing can also be turned on or off from this page [32].

<u>Generating reports</u>

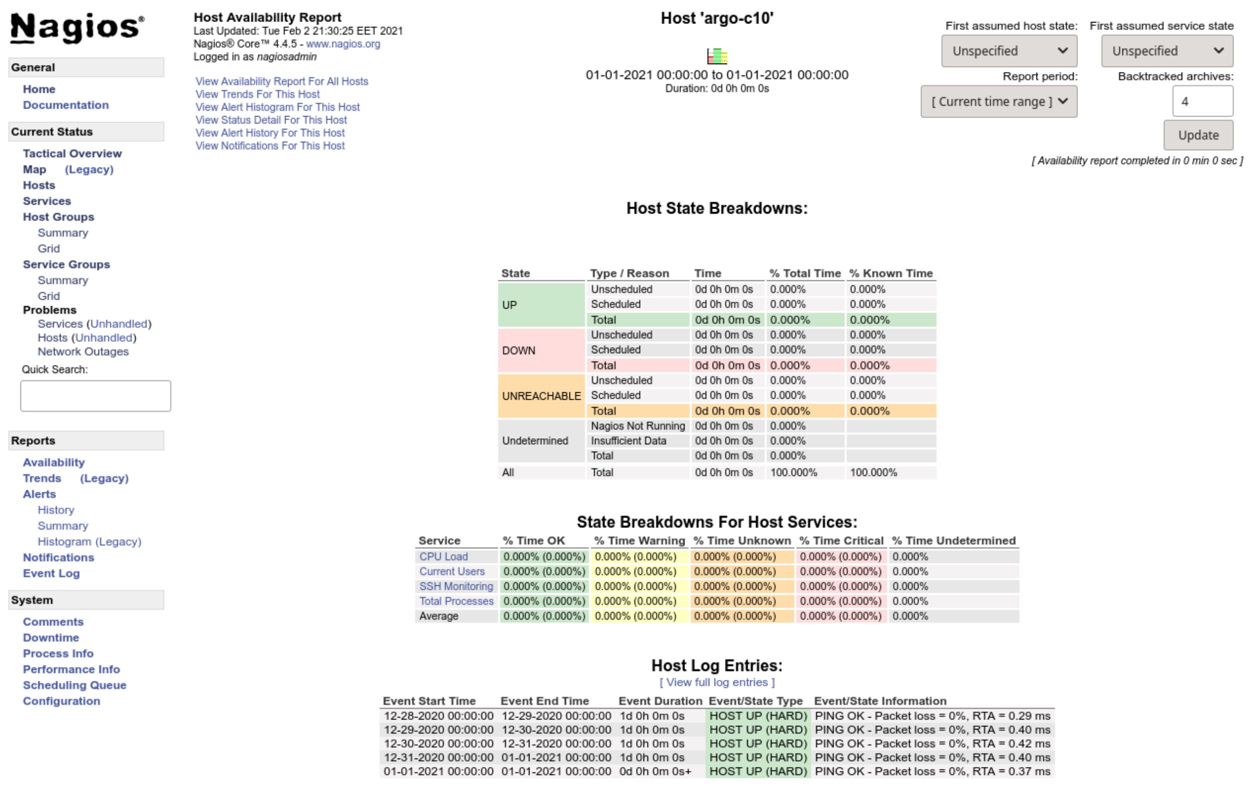
One of the most important features of the web interface is the ability to create reports. The reporting functionality can be used to browse historical notifications and alerts and see complete logs from a specified period. Nagios offers the following types of reports:

- **Trend reporting for host or service:** This shows state history changes for a single object along with status information from performed checks

- **Availability report for hosts or services:** This shows reports on how much time an object has spent in a particular status

- **Alert histogram:** This shows the number of alerts that have occurred over a period of time for a particular host or service

In addition, Nagios can report a history of alerts, notifications, or all events. This can be considered for reading Nagios logs in a more convenient way. It allows reading history either for all hosts and/or services as well as for a specific object. Generating most reports begins with choosing the report type, then the object type of the host, host group, service, or service group. Then either all or a specific object is chosen for which the report will be generated. Next, a form for specifying the period for which a report should be generated along with many additional options, which can be dependent on the type of report that should be generated. Additionally, a period can be specified so that the report only includes specific time periods such as working hours.



Host Availability Report
Last Updated: Tue Feb 2 21:30:25 EET 2021
Nagios® Core™ 4.4.5 - www.nagios.org
Logged in as *nagiosadmin*

Host 'argo-c10'

01-01-2021 00:00:00 to 01-01-2021 00:00:00
Duration: 0d 0h 0m 0s

First assumed host state: Unspecified
First assumed service state: Unspecified
Report period: [Current time range]
Backtracked archives: 4

[Availability report completed in 0 min 0 sec]

Host State Breakdowns:

State	Type / Reason	Time	% Total Time	% Known Time
UP	Unscheduled	0d 0h 0m 0s	0.000%	0.000%
	Scheduled	0d 0h 0m 0s	0.000%	0.000%
	Total	0d 0h 0m 0s	0.000%	0.000%
DOWN	Unscheduled	0d 0h 0m 0s	0.000%	0.000%
	Scheduled	0d 0h 0m 0s	0.000%	0.000%
	Total	0d 0h 0m 0s	0.000%	0.000%
UNREACHABLE	Unscheduled	0d 0h 0m 0s	0.000%	0.000%
	Scheduled	0d 0h 0m 0s	0.000%	0.000%
	Total	0d 0h 0m 0s	0.000%	0.000%
Undetermined	Nagios Not Running	0d 0h 0m 0s	0.000%	
	Insufficient Data	0d 0h 0m 0s	0.000%	
	Total	0d 0h 0m 0s	0.000%	
All	Total	0d 0h 0m 0s	100.000%	100.000%

State Breakdowns For Host Services:

Service	% Time OK	% Time Warning	% Time Unknown	% Time Critical	% Time Undetermined
CPU Load	0.000% (0.000%)	0.000% (0.000%)	0.000% (0.000%)	0.000% (0.000%)	0.000%
Current Users	0.000% (0.000%)	0.000% (0.000%)	0.000% (0.000%)	0.000% (0.000%)	0.000%
SSH Monitoring	0.000% (0.000%)	0.000% (0.000%)	0.000% (0.000%)	0.000% (0.000%)	0.000%
Total Processes	0.000% (0.000%)	0.000% (0.000%)	0.000% (0.000%)	0.000% (0.000%)	0.000%
Average	0.000% (0.000%)	0.000% (0.000%)	0.000% (0.000%)	0.000% (0.000%)	0.000%

Host Log Entries:
[View full log entries]

Event Start Time	Event End Time	Event Duration	Event/State Type	Event/State Information
12-28-2020 00:00:00	12-29-2020 00:00:00	1d 0h 0m 0s	HOST UP (HARD)	PING OK - Packet loss = 0%, RTA = 0.29 ms
12-29-2020 00:00:00	12-30-2020 00:00:00	1d 0h 0m 0s	HOST UP (HARD)	PING OK - Packet loss = 0%, RTA = 0.40 ms
12-30-2020 00:00:00	12-31-2020 00:00:00	1d 0h 0m 0s	HOST UP (HARD)	PING OK - Packet loss = 0%, RTA = 0.42 ms
12-31-2020 00:00:00	01-01-2021 00:00:00	1d 0h 0m 0s	HOST UP (HARD)	PING OK - Packet loss = 0%, RTA = 0.40 ms
01-01-2021 00:00:00	01-01-2021 00:00:00	0d 0h 0m 0s+	HOST UP (HARD)	PING OK - Packet loss = 0%, RTA = 0.37 ms

Figure 6: Host Availability Report

After specifying the parameters to the form and submitting it, the web interface will generate a report that matches your criteria. Some types of reports also allow us to export the information into the CSV format for further analysis [32].

NRPE

The NRPE addon is designed to allow the execution of Nagios plugins on remote Linux/Unix machines. The main reason for doing this is to allow Nagios to monitor "local" resources (like CPU load, memory usage, etc.) on remote machines. Since these public resources are not usually exposed to external machines, an agent like NRPE must be installed on the remote Linux/Unix machines [30].

The NRPE addon consists of two pieces:

• The check_nrpe plugin, which resides on the local monitoring machine

• The NRPE daemon, which runs on the remote Linux/Unix machine

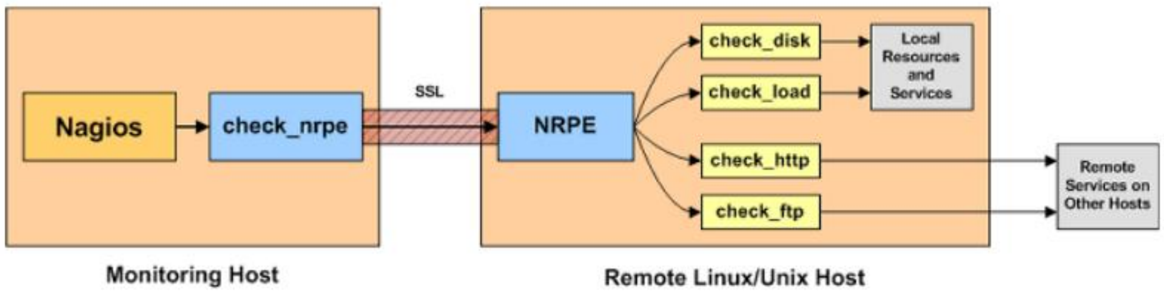


Figure 7: NRPR Design Overview [30]

To monitor a resource of service from a remote Linux/Unix machine, Nagios will execute the check_nrpe plugin and tell it what service needs to be checked. The check_nrpe plugin contacts the NRPE daemon on the remote host over an (optionally) SSL-protected connection. The NRPE daemon runs the appropriate Nagios plugin to check the service or resource. The results from the service check are passed from the NRPE daemon back to the check_nrpe plugin, which then returns the check results to the Nagios process [46].

The NRPE daemon requires that Nagios plugins be installed on the remote Linux/Unix host. Without these, the daemon would not be able to monitor anything. [30]

Nagios key system paths:

- /opt/nagios: This is where Nagios binaries and HTML files were installed
- /opt/nagios/plugins: This is where Nagios plugins are installed
- /etc/nagios: All configuration files should be placed here
- /var/nagios: This is where Nagios will keep its local state

2.4. Physical System Management

At some point in time, every system administrator finds that they need to access the console of a machine or power cycle it. The cluster is housed by the Institute of Accelerating Systems and Applications (IASA) [51], the assistance of the institute's administrators is required for manual intervention on the cluster, such as rebooting the system manually.

3. implementation

The implementation process was done in several stages. The first stage was to upgrade the cluster host OS from 1.3.8 to 1.3.9. The motivation for this update was to implement a wide system update to address the security risks and to fix minor bugs in the initial cluster. The second stage was to add a new computing node (argo-c10) to the cluster, thus in addition to the 80 cores installed previously, we added 10 cores node with 2 GPU CUDA capable devices. The third stage was to implement a major upgrade on the system the base OS was updated from CentOS 7.x to CntOS 8.x and from OpenHPC 1.3.9 to OpenHPC 2.x. The first two stages of the implementation process are described in appendices I and II, while in this chapter we focus on OpenHPC 2 set up.

The termS Host, head node master, and system management server (SMS) are used throughout the text all referring to master node hosting the OpenHPC tools and the OS.

The terms Host, head node master, and system management server (SMS) are used throughout the text all referring to master node hosting the OpenHPC tools and the OS. The first step to running OpenHPC is to run the SMS node. This is rather straightforward and simply requires creating a host with two network interfaces, one on the “public access” network and another on the “compute node” network for the personal OpenHPC cluster, loading it with the OpenHPC SMS CentOS image, and following the OpenHPC installation guide steps for SMS nodes. The steps include installing the OpenHPC packages and building the VNFS image for the compute nodes, followed by populating the WareWulf database with the virtual compute node MAC addresses and custom IP addresses. Much of the above process can be scripted using the OpenHPC installation guide [9]. The implementation chapter follows a step-by-step guide for the installation process for OpenHPC.

We installed CentOS8.2 distribution on the head node using a bootable USB ISO image and followed the distribution provided directions to install the Base Operating System (BOS) on the master host. During the installation, we set up the “public access” network1 as it's advisable to have the host connected to the internet during the installation. As for the installation destination, we combined the two identical 1TB hard drives, which through the operating system are connected to RAID1 3 technology, to provide data residencies to secure the Cluster data. After choosing the size of the partitions, we proceeded with the software selection process.

For the base environment, the option **Workstation** was chosen, in addition to the **Development Tools**, **Security Tools**, and **Performance Tools** Add-Ons for the selected Environment.

After rebooting the Host, we did update the OS packages as follow:

```
[root@argo ~]# yum check-update
[root@argo ~]# yum clean all
[root@argo ~]# yum update
```

we also installed the Extra Packages for Enterprise Linux (or EPEL) that creates, maintains, and manages a high-quality set of additional packages for Enterprise Linux, including, CentOS [47]

```
[root@argo ~]# yum install epel-release
```

The next step was to set the Bashrc[1] file with the cluster parameters to start the OpenHPC setup process. Prior to beginning the installation process of OpenHPC, we resolved the hostname and disabled SELinux, and since provisioning services rely on DHCP, TFTP, and HTTP network protocols we disabled the local firewall.

```
[root@argo ~]# echo ${sms_ip} ${sms_name} >> /etc/hosts

[root@argo ~]# selinuxenabled

[root@argo ~]# systemctl disable firewalld

[root@argo ~]# systemctl stop firewalld
```

In the following steps, we followed the OpenHPC (v2.0) Cluster Building Recipes. We kept the next subsection naming identical to the installation recipe guide and added the subsections numbering to the section's titles for easier reference.

3.1. Enable OpenHPC repository for local use (3.1)

To begin, enable the use of the OpenHPC repository by adding it to the local list of available package repositories.

```
[root@argo ~]# yum install http://repos.openhpc.community/OpenHPC/2/CentOS_8/x86_64/ohpc-release-2-1.el8.x86_64.rpm
```

The CentOS PowerTools repository is typically disabled in a standard install; however we can enable the repository as follows:

```
[root@argo ~]# yum install dnf-plugins-core

[root@argo ~]# yum config-manager --set-enabled PowerTools
```

3.2. Installation template (3.2)

The OpenHPC documentation package (docs-ohpc) includes a template script that includes a summary of all the commands used herein. This script can be used in conjunction with a simple text file its recommended to copy commands from the text file to use.

3.3. Add provisioning services on master node (3.3)

With the OpenHPC repository enabled, we begin adding the desired components onto the master server. This repository provides several aliases that are grouped into logical components to help aid in this process. To add support for provisioning services, the

[1] To avoid any error, we backed up the bashrc file from the previous set up and used it in this installation.

following commands illustrate the addition of a common base package followed by the Warewulf provisioning system.

```
[root@argo ~]# yum -y install ohpc-base

[root@argo ~]# yum -y install ohpc-warewulf
```

HPC systems rely on synchronized clocks throughout the system, and the NTP protocol can be used to facilitate this synchronization. To enable NTP services on the SMS host with a specific server ${ntp server}, and allow this server to serve as a local time server for the cluster, issue the following:

```
[root@argo ~]# systemctl enable chronyd.service

[root@argo ~]# echo "server ${ntp_server}" >> /etc/chrony.conf

[root@argo ~]# echo "allow all" >> /etc/chrony.conf

[root@argo ~]# systemctl restart chronyd
```

3.4. Add resource management services on master node (3.4)

OpenHPC provides multiple options for distributed resource management. The following command adds the OpenPBS workload manager server components to the chosen master host.

```
[root@argo ~]# yum -y install openpbs-server-ohpc
```

The client-side components will be added to the corresponding compute image in a subsequent step.

3.5. Complete basic Warewulf setup for master node (3.7)

At this point, we installed all the packages necessary to use Warewulf on the master host.

Several configuration files need to be updated to allow Warewulf to work with CentOS8.2 and to support local provisioning using a second private interface (${sms eth internal}).

```
[root@argo ~]# perl -pi -e "s/device = eth1/device = ${sms_eth_internal}/" /etc/warewulf/provision.conf

[root@argo ~]# ip link set dev ${sms_eth_internal} up

[root@argo ~]# ip address add ${sms_ip}/${internal_netmask} broadcast + dev ${sms_eth_internal}
```

After configuring the compute node's network, we must restart the relevant services to support the provisioning

```
[root@argo ~]# systemctl enable httpd.service

[root@argo ~]# systemctl restart httpd

[root@argo ~]# systemctl enable dhcpd.service

[root@argo ~]# systemctl enable tftp

[root@argo ~]# systemctl restart tftp
```

3.6. Define compute image for provisioning (3.8)

After enabling the provisioning services, the next step is to define and customize a system image that can subsequently be used to provision the compute nodes.

We imply two images to provision the compute node the first to provision the ten nodes argo-c0 to argo-c9. As for argo-c10 we used a separated provisioning image for the node due to differences in hardware and since argo-c10 requires the installation of Nvidia drivers. The following subsections highlight the process of provisioning the first ten nodes.

3.6.1. Build initial BOS image (3.8.1)

The OpenHPC build of Warewulf includes specific enhancements enabling support for CentOS8.2. The following steps illustrate the process to build a minimal, default image for use with Warewulf.

We begin by defining a directory structure on the master host that will represent the root filesystem of the compute node.[2] The default location for the image is in `/opt/ohpc/admin/images/centos8.2.`

The following command builds the initial chroot image and enables OpenHPC and EPEL repositories inside the chroot.

The wwmkchroot process used in the previous step is designed to provide a minimal CentOS8.2 configuration.

```
[root@argo ~]# wwmkchroot -v centos-8 $CHROOT

[root@argo ~]# dnf -y --installroot $CHROOT install epel-release

[root@argo ~]# cp -p /etc/yum.repos.d/OpenHPC*.repo $CHROOT/etc/yum.repos.d
```

3.6.2. Add OpenHPC components (3.8.2)

In this section, we add additional components to include resource management client services, NTP support, and other additional packages to support the default OpenHPC environment. This process augments the chroot-based install performed by wwmkchroot to modify the base provisioning image and will access the Base Operating System (BOS) and OpenHPC repositories to resolve package install requests. We begin by installing a few common base packages:

```
[root@argo ~]# yum -y --installroot=$CHROOT install ohpc-base-compute
```

To access the remote repositories by hostname (and not IP addresses), the chroot environment needs to be updated to enable DNS resolution. Assuming that the master host has a working DNS configuration in place, the chroot environment can be updated with a copy of the configuration as follows:

```
[root@argo ~]# cp -p /etc/resolv.conf $CHROOT/etc/resolv.conf
```

[2] As a precursory measure we can run (echo `$CHROOT`) if the command returns an empty line, this indicates an issue with the bashrc file. Normally the command must show the default location for the compute image.

We also included additional required components to the compute instance, including resource manager client, NTP, kernel drivers, and development environment modules support.

```
[root@argo ~]# yum -y --installroot=$CHROOT install openpbs-execution-ohpc

[root@argo ~]# perl -pi -e "s/PBS_SERVER=\S+/PBS_SERVER=${sms_name}/" $CHROOT/etc/pbs.conf

[root@argo ~]# echo "PBS_LEAF_NAME=${sms_name}" >> /etc/pbs.conf

[root@argo ~]# chroot $CHROOT opt/pbs/libexec/pbs_habitat

[sms]# perl -pi -e "s/\$clienthost \S+/\$clienthost ${sms_name}/" $CHROOT/var/spool/pbs/mom_priv/config

[root@argo ~]# echo "\$usecp *:/home /home" >> $CHROOT/var/spool/pbs/mom_priv/config

[root@argo ~]# chroot $CHROOT systemctl enable pbs

[root@argo ~]# yum -y --installroot=$CHROOT install chrony

[root@argo ~]# echo "server ${sms_ip}" >> $CHROOT/etc/chrony.conf

[root@argo ~]# yum -y --installroot=$CHROOT install kernel-`uname -r`

[root@argo ~]# yum -y --installroot=$CHROOT install lmod-ohpc
```

3.6.3. Customize system configuration (3.8.3)

Prior to assembling the image, it is advantageous to perform any additional customization within the chroot environment created for the desired compute instance. The following steps document the process to add a local SSH key created by Warewulf to support remote access and enable NFS mounting of a $HOME file system and the public OpenHPC install path (/opt/ohpc/pub) that will be hosted by the master host in this configuration.

```
[root@argo ~]# wwinit database

[root@argo ~]# wwinit ssh_keys


[root@argo ~]# echo "${sms_ip}:/home /home nfs nfsvers=3,nodev,nosuid 0 0" >>
$CHROOT/etc/fstab

[root@argo ~]# echo "${sms_ip}:/opt/ohpc/pub /opt/ohpc/pub nfs nfsvers=3,nodev 0 0" >>
$CHROOT/etc/fstab


[root@argo ~]# echo "/home *(rw,no_subtree_check,fsid=10,no_root_squash)" >>
/etc/exports

[root@argo ~]# echo "/opt/ohpc/pub *(ro,no_subtree_check,fsid=11)" >> /etc/exports

[root@argo ~]# exportfs -a

[root@argo ~]# systemctl restart nfs-server

[root@argo ~]# systemctl enable nfs-server
```

3.6.4. Enable forwarding of system logs (3.8.4.6)

To consolidate system logging information for the cluster in a central location. The following commands highlight the steps necessary to configure compute nodes to forward their logs to the SMS, and to allow the SMS to accept these log requests.

```
[root@argo ~]# echo 'module(load="imudp")' >> /etc/rsyslog.d/ohpc.conf
[root@argo ~]# echo 'input(type="imudp" port="514")' >> /etc/rsyslog.d/ohpc.conf
[root@argo ~]# systemctl restart rsyslog
[root@argo ~]# echo "*.* @${sms_ip}:514" >> $CHROOT/etc/rsyslog.conf
[root@argo ~]# echo "Target=\"${sms_ip}\" Protocol=\"udp\"" >>
$CHROOT/etc/rsyslog.conf
[root@argo ~]# perl -pi -e "s/^\*\.info/\\#\*\.info/" $CHROOT/etc/rsyslog.conf
[root@argo ~]# perl -pi -e "s/^authpriv/\\#authpriv/" $CHROOT/etc/rsyslog.conf
[root@argo ~]# perl -pi -e "s/^mail/\\#mail/" $CHROOT/etc/rsyslog.conf
[root@argo ~]# perl -pi -e "s/^cron/\\#cron/" $CHROOT/etc/rsyslog.conf
[root@argo ~]# perl -pi -e "s/^uucp/\\#uucp/" $CHROOT/etc/rsyslog.conf
```

3.6.5. Add Nagios monitoring (3.8.4.7)

The core Nagios daemon and a variety of monitoring plugins are provided by the underlying OS distribution and the following commands can be used to install and configure a Nagios server on the master node and add the facility to run tests and gather metrics from provisioned compute nodes.

Further configurations are required to run Nagios are discussed in subsection 3.6.6

```
[root@argo ~]# yum -y install --skip-broken nagios nrpe nagios-plugins-*
[root@argo ~]# yum -y --installroot=$CHROOT install nrpe nagios-plugins-ssh
[root@argo ~]# chroot $CHROOT systemctl enable nrpe
[root@argo ~]# perl -pi -e "s/^allowed_hosts=/# allowed_hosts=/"
$CHROOT/etc/nagios/nrpe.cfg
[root@argo ~]# echo "nrpe : ${sms_ip} : ALLOW" >> $CHROOT/etc/hosts.allow
[root@argo ~]# echo "nrpe : ALL : DENY"
[root@argo ~]# echo "nrpe : ALL : DENY" >> $CHROOT/etc/hosts.allow


[root@argo ~]# cp /opt/ohpc/pub/examples/nagios/compute.cfg /etc/nagios/objects
[root@argo ~]# echo "cfg_file=/etc/nagios/objects/compute.cfg" >>
/etc/nagios/nagios.cfg
[root@argo ~]# perl -pi -e "s/ \/bin\/mail/ \/usr\/bin\/mailx/g"
/etc/nagios/objects/commands.cfg
[root@argo ~]# perl -pi -e "s/nagios\@localhost/root\@${sms_name}/"
/etc/nagios/objects/contacts.cfg
[root@argo ~]# echo command[check_ssh]=/usr/lib64/nagios/plugins/check_ssh localhost
$CHROOT/etc/nagios/nrpe.cfg
[root@argo ~]# htpasswd -bc /etc/nagios/passwd nagiosadmin ${nagios_web_password}
[root@argo ~]# systemctl enable nagios
[root@argo ~]# systemctl start nagios
```

```
[root@argo ~]# chmod u+s `which ping`
```

3.6.6. Nagios Setup

After installing the core Nagios daemon, we have to configure a Nagios server on the master node and install a variety of monitoring plugins to gather metrics from the provisioned compute nodes.

OpenHPC distro provides several Nagios plugins. We installed the following:

```
[root@argo ~]# yum -y --installroot=$CHROOT install nagios-plugins-users nagios-plugins-load nagios-plugins-procs
```

the file CHROOT/etc/nagios/nrpe.cfg contain the configuration for each node by default the following section is undocumented

```
. . . . .

# The following examples use hardcoded command arguments...

# This is by far the most secure method of using NRPE


command[check_users]=/usr/lib64/nagios/plugins/check_users -w 5 -c 10

command[check_load]=/usr/lib64/nagios/plugins/check_load -r -w .15,.10,.05 -c .30,.25,.20

command[check_hda1]=/usr/lib64/nagios/plugins/check_disk -w 20% -c 10% -p /dev/hda1

command[check_zombie_procs]=/usr/lib64/nagios/plugins/check_procs -w 5 -c 10 -s Z

command[check_total_procs]=/usr/lib64/nagios/plugins/check_procs -w 150 -c 200

. . . . .
```

We can modify the parameters of each command based on user demand. In this case, we left the parameters to the default values. After setting up the nodes for monitoring we have to modify Nagios configuration on the host [30] .
We have to create a command definition in **/etc/nagios/objects/commands.cfg** object configuration files in order to use the check_nrpe plugin.

```
define command {


    command_name    check_nrpe

    command_line    $USER1$/check_nrpe -H $HOSTADDRESS$ -c $ARG1$

}
```

Next, we’ll need to create some object definitions in order to monitor the remote Linux/Unix machine. These definitions can be placed in an individual file or added to an already existing object configuration file. **(/etc/nagios/objects/compute.cfg)** First, define a new host group for the remote Linux/Unix box that references the newly created host group template.

```
define hostgroup {

    hostgroup_name compute

    alias compute nodes
    members      argo-c0,argo-c1,argo-c2,argo-c3,argo-c4,argo-c5,argo-c6,argo-
c7,argo-c8,argo-c9,argo-c10
}


define host {
    use linux-server
    host_name argo-c0
}


. . . . .


define host {
    use linux-server
    host_name argo-c10
}
```

Next, we define some services for monitoring the remote Linux/Unix box. These example service definitions will use the sample commands that have been defined in the nrpe.cfg file on the remote host. The following service will monitor the CPU load on the remote host. The "check_load" argument that is passed to the check_nrpe command definition tells the NRPE daemon to run the "check_load" command as defined in the nrpe.cfg file in the compute node.

```
define service {
    use                     generic-service
    hostgroup_name          compute
    service_description     CPU Load
    check_command           check_nrpe!check_load
}
```

Note: it's best to back up the above-mentioned files before upgrading the cluster software to restore the previous configuration upon completion.

After building the cluster, we can check Nagios for errors by using the following command:

```
[root@argo ~]# nagios -v /etc/nagios/nagios.cfg
```

Also, we can use the next command to reload/restart Nagios core services:

```
[root@argo ~]# service nagios reload
```

3.6.7. Add NHC (3.8.4.11)

Resource managers often provide for a periodic" node health check" to be performed on each compute node to verify that the node is working properly. Nodes that are determined to be" unhealthy" can be marked as down or offline to prevent jobs from being scheduled or run on them.

The NHC driver script is run periodically on each compute node by the resource manager client daemon. It loads its configuration file to determine which checks are to be run on the current node (based on its hostname). It can also be configured to mark nodes offline so that the scheduler will not assign jobs to bad nodes, reducing the risk of system-induced job failures [48].

```
[root@argo ~]# yum -y install nhc-ohpc

[root@argo ~]# yum -y --installroot=$CHROOT install nhc-ohpc
```

3.6.8. Import files (3.8.5)

The Warewulf system includes functionality to import arbitrary files from the provisioning server for distribution to managed hosts. This is one way to distribute user credentials to compute nodes. To import local file-based credentials, issue the following:

```
[root@argo ~]# wwsh file import /etc/passwd

[root@argo ~]# wwsh file import /etc/group

[root@argo ~]# wwsh file import /etc/shadow
```

3.7. Finalizing provisioning configuration (3.9)

Warewulf employs a two-stage boot process for provisioning nodes via the creation of a bootstrap image that is used to initialize the boot process, and a virtual node file system capsule containing the full system image. This section highlights the creation of the necessary provisioning images followed by the registration of desired compute nodes.

3.7.1. Assemble bootstrap image (3.9.1)

The bootstrap image includes the runtime kernel and associated modules, as well as some simple scripts to complete the provisioning process. The following commands highlight the inclusion of additional drivers and the creation of the bootstrap image based on the running kernel, and building the bootstrap image:

```
[root@argo ~]# export WW_CONF=/etc/warewulf/bootstrap.conf

[root@argo ~]# echo "drivers += updates/kernel/" >> $WW_CONF


[root@argo ~]# wwbootstrap `uname -r`
```

3.7.2. Assemble Virtual Node File System (VNFS) image (3.9.2)

With the local site customizations in place, the following step uses the `wwvnfs` command to assemble a VNFS capsule from the chroot environment defined for the compute instance.

```
[root@argo ~]# wwvnfs --chroot $CHROOT
```

3.7.3. Register nodes for provisioning (3.9.3)

In preparation for provisioning, we can now define the desired network settings for the compute nodes with the underlying provisioning system and restart the DHCP service. The compute nodes hostnames, node IPs, and MAC addresses are required by Warewulf to register the nodes. The required parameters are stored in the bashrc file.

The final step in this process associates the VNFS image assembled in previous steps with the newly defined compute nodes, utilizing the user credential files and merge key that was imported in 3.6.7.

```
[root@argo ~]# echo "GATEWAYDEV=${eth_provision}" > /tmp/network.$$

[root@argo ~]# wwsh -y file import /tmp/network.$$ --name network

[root@argo ~]# wwsh -y file set network --path /etc/sysconfig/network --mode=0644 --
uid=0


[root@argo ~]# for ((i=0; i<$num_computes; i++)) ; do

wwsh  -y  node  new  ${c_name[i]}  --ipaddr=${c_ip[i]}  --hwaddr=${c_mac[i]}  -D
${eth_provision}

done
```

Next, we define the provisioning image for the nodes, and restart the DHCP service and lastly update the PXE table:

```
[root@argo  ~]#  wwsh  -y  provision  set  "${compute_regex}"  --vnfs=centos8.2  --
bootstrap=`uname -r`  --files=dynamic_hosts,passwd,group,shadow,network


[root@argo ~]# systemctl restart dhcpd

[root@argo ~]# wwsh pxe update
```

The last step in this stage is to reboot the compute nodes to load the OS images[3].

```
[root@argo ~]# for ((i=0; i<${num_computes}; i++)) ; do

ipmitool -E -I lanplus -H ${c_bmc[$i]} -U ${bmc_username} -P ${bmc_password} chassis
power reset

sleep 10;

done
```

3.8. Development Tools (4.1)

OpenHPC-provided packages can now be added to support a flexible HPC development environment, including development tools, C/C++/FORTRAN compilers, MPI stacks, and a variety of 3rd party libraries. The following subsections highlight the additional software installation procedures. Also, OpenHPC provides recent versions of the GNU Autotools collection, the Valgrind memory debugger, EasyBuild, and Spack. These can be installed as follows:

```
[root@argo ~]# yum -y install ohpc-autotools
```

[3] A 10 seconds timer was added between each reboot to prevent any power surge.

```
[root@argo ~]# yum -y install EasyBuild-ohpc
[root@argo ~]# yum -y install hwloc-ohpc
[root@argo ~]# yum -y install spack-ohpc
[root@argo ~]# yum -y install valgrind-ohpc
```

3.9. Compilers (4.2)

OpenHPC presently packages the GNU compiler toolchain integrated with the underlying Lmod modules system in a hierarchical fashion. The modules system will conditionally present compiler-dependent software based on the toolchain currently loaded.

```
[root@argo ~]# yum -y install gnu9-compilers-ohpc
```

3.10. MPI Stacks (4.3)

For MPI development and runtime support, OpenHPC provides pre-packaged builds for a variety of MPI families and transport layers. OpenHPC 2.x introduces the use of two related transport layers for the MPICH and OpenMPI builds that support a variety of underlying fabrics.

```
[root@argo ~]# yum -y install openmpi4-gnu9-ohpc mpich-ofi-gnu9-ohpc
[root@argo ~]# yum -y install mpich-ucx-gnu9-ohpc
```

3.11. Performance Tools (4.4)

OpenHPC provides a variety of open-source tools to aid in application performance analysis. This group of tools can be installed as follows:

```
[root@argo ~]# yum -y install ohpc-gnu9-perf-tools
```

3.12. Setup default development environment (4.5)

It is convenient to have a default development environment in place so that compilation can be performed directly for parallel programs requiring MPI. This setup can be conveniently enabled via modules and the OpenHPC modules environment is pre-configured to load an ohpc module on login. The following package install provides a default environment that enables Autotools, the GNU compiler toolchain, and the OpenMPI stack.

```
[root@argo ~]# yum -y install lmod-defaults-gnu9-openmpi4-ohpc
```

3.13. 3rd Party Libraries and Tools (4.6)

OpenHPC provides pre-packaged builds for several popular open-source tools and libraries used by HPC applications and developers. The general naming convention for builds provided by OpenHPC is to append the compiler and MPI family name that the library was built against directly into the package name.

For convenience, OpenHPC provides package aliases for these 3rd party libraries and utilities that can be used to install available libraries for use with the GNU compiler family toolchain. For parallel libraries, aliases are grouped by MPI family toolchain so that administrators can choose a subset should they favor a particular MPI stack.

To install all available package offerings within OpenHPC, issue the following:

```
[root@argo ~]# yum -y install ohpc-gnu9-serial-libs
[root@argo ~]# yum -y install ohpc-gnu9-io-libs
[root@argo ~]# yum -y install ohpc-gnu9-python-libs
[root@argo ~]# yum -y install ohpc-gnu9-runtimes

[root@argo ~]# yum -y install ohpc-gnu9-mpich-parallel-libs
[root@argo ~]# yum -y install ohpc-gnu9-openmpi4-parallel-libs
```

3.14. Optional Development Tool Builds (4.7)

OpenHPC provides optional compatible builds for use with the compilers and MPI stack included in newer versions of the Intel® Parallel Studio XE software suite. These packages provide a similar hierarchical user environment experience as other compiler and MPI families present in OpenHPC.

To take advantage of the available builds, the Parallel Studio software suite must be obtained and installed separately. Once installed locally, the OpenHPC compatible packages can be installed using standard package manager semantics. [9]

3.15. Resource Manager Startup (5)

With the cluster nodes up and functional, we can now startup the resource manager services on the master host in preparation for running user jobs. The following command can be used to startup the necessary services to support resource management under OpenPBS.

```
[root@argo ~]# systemctl enable pbs
[root@argo ~]# systemctl start pbs
```

To add the compute nodes to the OpenPBS resource manager, execute the following to register each host and apply several key global configuration options, to create an execution queue and append the nodes to the newly created queue.

```
[root@argo ~]# . /etc/profile.d/pbs.sh
[root@argo ~]# qmgr -c "set server default_qsub_arguments= -V"
[root@argo ~]# qmgr -c "set server resources_default.place=excl"
[root@argo ~]# qmgr -c "set server job_history_enable=True"

[root@argo ~]# qmgr -c "create queue N10C80"
[root@argo ~]# qmgr -c "set queue N10C80 exec_queue queue_type = execution"
[root@argo ~]# qmgr -c "set queue N10C80 queue_type = execution"
[root@argo ~]# qmgr -c "set queue N10C80 enabled = True"
[root@argo ~]# qmgr -c "set queue N10C80 started = True"
[root@argo ~]# qmgr -c "set server default_queue = N10C80"

[root@argo ~]# for host in "${c_name[@]}"; do
```

```
     qmgr -c "create node $host"
     done


[root@argo ~]# for host in "${c_name[@]}"; do
     qmgr -c "set node $host queue= N10C80"
     done
```

3.16. GPU compute node

This section describes the process of adding the argo-c10 node with the stateless provisioning option to the cluster and installing CUDA development tools to the node and the host. We used a separated provisioning image for the node due to differences in hardware and since argo-c10 requires the installation of Nvidia drivers. The following subsections highlight the process of provisioning the node and installing the Nvidia drivers. For the Nvidia driver installation, we followed Nvidia instructions [23].

We replaced the cheroot variable to point to the OS image location to build the initial BOS image.

3.16.1. Build initial BOS image (3.8.1)

We begin by defining a directory structure on the master host that will represent the root filesystem of the compute node. The default location for the image is in `/opt/ohpc/admin/images/c10/centos8.2_c10`.

The following command builds the initial chroot image and enables OpenHPC and EPEL repositories inside the chroot.

The wwmkchroot process used in the previous step is designed to provide a minimal CentOS8.2 configuration.

```
[root@argo ~]# wwmkchroot -v centos-8 $CHROOT
[root@argo ~]# dnf -y --installroot $CHROOT install epel-release
[root@argo ~]# cp -p /etc/yum.repos.d/OpenHPC*.repo $CHROOT/etc/yum.repos.d
```

3.16.2. Add OpenHPC components (3.8.2)

In this section, we add additional components to include resource management client services, NTP support, and other additional packages to support the default OpenHPC environment. This process augments the chroot-based install performed by wwmkchroot to modify the base provisioning image and will access the Base Operating System (BOS) and OpenHPC repositories to resolve package install requests. We begin by installing a few common base packages:

```
[root@argo ~]# yum -y --installroot=$CHROOT install ohpc-base-compute
```

To access the remote repositories by hostname (and not IP addresses), the chroot environment needs to be updated to enable DNS resolution. Assuming that the master host has a working DNS configuration in place, the chroot environment can be updated with a copy of the configuration as follows:

```
[root@argo ~]# cp -p /etc/resolv.conf $CHROOT/etc/resolv.conf
```

We also included additional required components to the compute instance including resource manager client, NTP, kernel drivers, and development environment modules support.

```
[root@argo ~]# yum -y --installroot=$CHROOT install openpbs-execution-ohpc

[root@argo ~]# perl -pi -e "s/PBS_SERVER=\S+/PBS_SERVER=${sms_name}/" $CHROOT/etc/pbs.conf

[root@argo ~]# chroot $CHROOT opt/pbs/libexec/pbs_habitat

[sms]# perl -pi -e "s/\$clienthost \S+/\$clienthost ${sms_name}/" $CHROOT/var/spool/pbs/mom_priv/config

[root@argo ~]# echo "\$usecp *:/home /home" >> $CHROOT/var/spool/pbs/mom_priv/config

[root@argo ~]# chroot $CHROOT systemctl enable pbs

[root@argo ~]# yum -y --installroot=$CHROOT install chrony

[root@argo ~]# echo "server ${sms_ip}" >> $CHROOT/etc/chrony.conf

[root@argo ~]# yum -y --installroot=$CHROOT install kernel-`uname -r`

[root@argo ~]# yum -y --installroot=$CHROOT install lmod-ohpc
```

3.16.3. Customize system configuration (3.8.3)

Prior to assembling the image, it is advantageous to perform any additional customization within the chroot environment created for the desired compute instance. The following steps document the process to add a local ssh key created by Warewulf to support remote access and enable NFS mounting of a $HOME file system and the public OpenHPC install path (/opt/ohpc/pub) that will be hosted by the master host in this configuration.

```
[root@argo ~]# wwinit database

[root@argo ~]# wwinit ssh_keys


[root@argo ~]# echo "${sms_ip}:/home /home nfs nfsvers=3,nodev,nosuid 0 0" >>
$CHROOT/etc/fstab

[root@argo ~]# echo "${sms_ip}:/opt/ohpc/pub /opt/ohpc/pub nfs nfsvers=3,nodev 0 0" >>
$CHROOT/etc/fstab


[root@argo ~]# echo "/home *(rw,no_subtree_check,fsid=10,no_root_squash)" >>
/etc/exports

[root@argo ~]# echo "/opt/ohpc/pub *(ro,no_subtree_check,fsid=11)" >> /etc/exports

[root@argo ~]# exportfs -a

[root@argo ~]# systemctl restart nfs-server

[root@argo ~]# systemctl enable nfs-server
```

3.16.4. Enable forwarding of system logs (3.8.4.6)

To consolidate system logging information for the cluster in a central location. The following commands highlight the steps necessary to configure the compute nodes to forward their logs to the SMS.

```
[root@argo ~]# echo "*.* @${sms_ip}:514" >> $CHROOT/etc/rsyslog.conf

[root@argo ~]# echo "Target=\"${sms_ip}\" Protocol=\"udp\"" >>
$CHROOT/etc/rsyslog.conf

[root@argo ~]# perl -pi -e "s/^\*\.info/\\#\*\.info/" $CHROOT/etc/rsyslog.conf

[root@argo ~]# perl -pi -e "s/^authpriv/\\#authpriv/" $CHROOT/etc/rsyslog.conf

[root@argo ~]# perl -pi -e "s/^mail/\\#mail/" $CHROOT/etc/rsyslog.conf

[root@argo ~]# perl -pi -e "s/^cron/\\#cron/" $CHROOT/etc/rsyslog.conf

[root@argo ~]# perl -pi -e "s/^uucp/\\#uucp/" $CHROOT/etc/rsyslog.conf
```

3.16.5. Add Nagios monitoring (3.8.4.7)

The core Nagios daemon and a variety of monitoring plugins are provided by the underlying OS distro and the following commands can be used to install and configure a Nagios server on the master node and add the facility to run tests and gather metrics from provisioned compute nodes.

Further configurations are required to run Nagios.

```
[root@argo ~]# yum -y --installroot=$CHROOT install nrpe nagios-plugins-ssh

[root@argo ~]# chroot $CHROOT systemctl enable nrpe

[root@argo ~]# perl -pi -e "s/^allowed_hosts=/# allowed_hosts=/"
$CHROOT/etc/nagios/nrpe.cfg

[root@argo ~]# echo "nrpe : ${sms_ip} : ALLOW" >> $CHROOT/etc/hosts.allow

[root@argo ~]# echo "nrpe : ALL : DENY"

[root@argo ~]# echo "nrpe : ALL : DENY" >> $CHROOT/etc/hosts.allow

[root@argo ~]# systemctl enable nagios

[root@argo ~]# systemctl start nagios

[root@argo ~]# chmod u+s `which ping`
```

3.16.6. Add NHC (3.8.4.11)

Resource managers often provide for a periodic “node health check” to be performed on each compute node to verify that the node is working properly. Nodes that are determined to be” unhealthy” can be marked as down or offline to prevent jobs from being scheduled or run on them.

The NHC driver script is run periodically on each compute node by the resource manager client daemon. It loads its configuration file to determine which checks are to be run on the current node (based on its hostname). It can also be configured to mark nodes offline so that the scheduler will not assign jobs to bad nodes, reducing the risk of system-induced job failures.

```
[root@argo ~]# yum -y --installroot=$CHROOT install nhc-ohpc
```

3.16.7. Import files (3.8.5)

The Warewulf system includes functionality to import arbitrary files from the provisioning server for distribution to managed hosts. This is one way to distribute user credentials to compute nodes. To import local file-based credentials, issue the following:

```
[root@argo ~]# wwsh file import /etc/passwd
[root@argo ~]# wwsh file import /etc/group
[root@argo ~]# wwsh file import /etc/shadow
```

3.16.8. Nvidia Drivers Installation

The CUDA Driver requires that the kernel headers and development packages for the running version of the kernel to be installed at the time alongside with the driver installation, as well as whenever the driver is rebuilt. We were unable to install the kernel headers and development packages using the package manager, so we downloaded the appropriate rpm files and installed them manually.

```
[root@argo ~]# dnf -y --installroot $CHROOT install ./kernel-headers-4.18.0-193.19.1.el8_2.x86_64.rpm

[root@argo ~]# dnf -y --installroot $CHROOT install ./kernel-devel-4.18.0-193.19.1.el8_2.x86_64.rpm
```

To install the Display Driver, the Nouveau [50] drivers must be disabled first. To disable Nouveau drivers issue the following commands:

```
[root@argo ~]# echo "blacklist nouveau " >> /opt/ohpc/admin/images/c10/centos8.2_c10/etc/modprobe.d/blacklist-nouveau.conf

[root@argo ~]# echo "options nouveau modeset=0" >> /opt/ohpc/admin/images/c10/centos8.2_c10/etc/modprobe.d/blacklist-nouveau.conf
```

As for installation method, The NVIDIA CUDA Toolkit is available at https://developer.nvidia.com/cuda-downloads.[24] The site provides options to choose the platform you are using and download the NVIDIA CUDA Toolkit

The CUDA Toolkit contains the CUDA driver and tools needed to create, build and run a CUDA application as well as libraries, header files, CUDA samples source code, and other resources [23].

```
[root@argo ~]# dnf --installroot $CHROOT config-manager --add-repo https://developer.download.nvidia.com/compute/cuda/repos/rhel8/x86_64/cuda-rhel8.repo

[root@argo ~]# dnf -y --installroot $CHROOT module install nvidia-driver:latest-dkms
```

3.16.9. Finalizing provisioning configuration (3.9)

Warewulf employs a two-stage boot process for provisioning nodes via the creation of a bootstrap image that is used to initialize the boot process, and a virtual node file system capsule containing the full system image. This section highlights the creation of the necessary provisioning images followed by the registration of desired compute nodes.

3.16.10. Assemble bootstrap image (3.9.1)

The bootstrap image includes the runtime kernel and associated modules, as well as some simple scripts to complete the provisioning process. The following commands highlight the inclusion of additional drivers and the creation of the bootstrap image based on the running kernel and building the bootstrap image.

```
[root@argo ~]# export WW_CONF=/etc/warewulf/bootstrap_c10.conf

[root@argo ~]# echo "drivers += updates/kernel/" >> $WW_CONF

[root@argo ~]# echo "drivers += /usr/src/nvidia-455.45.01/" >> $WW_CONF


[root@argo ~]# wwbootstrap `uname -r` --name bootstrap_c10
```

We manually copied the kernel modules `/lib/modules/4.18.0-193.19.1.el8_2.x86_64/` to the chroot image since there was no other option to install them in the OS image.

3.16.11. Assemble Virtual Node File System (VNFS) image (3.9.2)

With the local site customization in place, the following step uses the `wwvnfs` command to assemble a VNFS capsule from the chroot environment defined for the compute instance.

```
[root@argo ~]# wwvnfs --chroot $CHROOT
```

3.16.12. Register the node for provisioning (3.9.3)

In preparation for provisioning, we can now define the desired network settings for the compute nodes with the underlying provisioning system and restart the DHCP service. The compute nodes hostnames, node IPs, and MAC addresses are acquired by Warewulf to register the nodes. The required parameters are stored in the bashrc file.

The final step in this process associates the VNFS image assembled in previous steps with the newly defined compute nodes, utilizing the user credential files and merge the keys that were imported in 3.6.7.

```
[root@argo ~]# echo "GATEWAYDEV=${eth_provision}" > /tmp/network.$$

[root@argo ~]# wwsh -y file import /tmp/network.$$ --name network

[root@argo ~]# wwsh -y file set network --path /etc/sysconfig/network --mode=0644 --
uid=0


[root@argo ~]# wwsh -y node new ${c_name[10]} --ipaddr=${c_ip[10]} --
hwaddr=${c_mac[10]} -D ${eth_provision}
```

Next, we define the provisioning image for the nodes and restart the DHCP service, and lastly update the PXE table:

```
[root@argo ~]# wwsh -y provision set "argo-c10" --vnfs=centos8.2_c10 --
bootstrap=bootstrap_c10 --files=dynamic_hosts,passwd,group,shadow,network


[root@argo ~]# systemctl restart dhcpd

[root@argo ~]# wwsh pxe update
```

The last step in this stage is to reboot the compute node to load the OS images.

```
[root@argo ~]# ipmitool -E -I lanplus -H ${c_bmc[$10]} -U ${bmc_username} -P
${bmc_password} chassis power reset
```

3.16.13. Resource Manager Setup

To add the compute node to the OpenPBS resource manager, execute the following to register each host and apply several key global configuration options, to create an execution queue and append the nodes to the newly created queue.

```
[root@argo ~]# qmgr -c "create queue GPUq"

[root@argo ~]# qmgr -c "set queue GPUq exec_queue queue_type = execution"

[root@argo ~]# qmgr -c "set queue GPUq queue_type = execution"

[root@argo ~]# qmgr -c "set queue GPUq enabled = True"

[root@argo ~]# qmgr -c "set queue GPUq started = True"


[root@argo ~]# qmgr -c "create node argo-c10"

[root@argo ~]# qmgr -c "set node argo-c10 queue= GPUq"
```

After creating the queue, we created GPU resources and equipped the node with the resources. As described in Appendix II.

```
[root@argo ~]# qmgr -c "create resource ngpus type=long, flag=nh"

[root@argo ~]# qmgr -c "set node argo-c10 resources_available.ngpus=2"
```

3.17. CUDA installation on the host

To develop and compile CUDA applications able to harness the power of the graphics processing unit (GPU), the installation of the CUDA development tools is required.

The Host does not require the Nvidia Drivers, but we installed them on the Host to be able to copy the kernels modules to the compute image as noted in (3.16.10)

```
[root@argo ~]#  dnf config-manager --add-repo
https://developer.download.nvidia.com/compute/cuda/repos/rhel8/x86_64/cuda-rhel8.repo

[root@argo ~]#  dnf clean all

[root@argo ~]#  dnf -y module install nvidia-driver:latest-dkms

[root@argo ~]#  dnf -y install cuda
```

To set the environment variables by modifying the PATH and LD_LIBRARY_PATH variables:

```
[root@argo-c10 ~]# export PATH=/usr/local/cuda-10.2/bin${PATH:+:${PATH}}

[root@argo-c10          ~]#          export          LD_LIBRARY_PATH=/usr/local/cuda-
10.2/lib64${LD_LIBRARY_PATH:+:${LD_LIBRARY_PATH}}
```

To make system-wide permanent changes to the PATH variables we added the cuda.sh script to the directory etc/profile.d/ so the path to the CUDA utilities whenever a bash shell is entered, as well as when the desktop session loads.

4. CONCLUSIONS

The process of developing and managing HPC Systems is a never-ending path. Cluster Administrators need to face multiple challenges requiring the addition, changing, updating, or removal of system components. To this end, OpenHPC proven a valuable tool easing node administration and updating.

This thesis examined the process of developing and managing an HPC System, delivering a system for users and administrators, allowing software teams to use it in computation-intensive operations. Updating OpenHPC packages from version 1.3.8 to 1.3.9. was necessary to ensure that the cluster remained up-to-date. Such updates are necessary due to the bug fixing and security issues they are addressing. Integration of a new node in Argo's infrastructure providing CUDA computation capabilities allowed to demonstrate Warewulf ability to manipulate cluster nodes individually and examine parallel computing in GPU processors.

The major step was installing the latest OpenHPC version (2.0), and CentOS8.2 on the master node, performing a clean installation, and reassembling the cluster, based on the existing scheme, this step confirms OpenHPC's flexibility and ease of use.

The entire process of setting up and managing such a cluster is straight forward enough, however requires significant Linux system administration skills and knowledge of the involved technologies, especially in cases where issues arise during the setup.

APPENDIX I

Update to OpenHPC 1.3.9

OpenHPC provides the ability to upgrade the installed packages to the latest repository versions to obtain access to bug fixes and newer component versions. This can be accomplished with the underlying package manager as OpenHPC packaging maintains a versioning state across releases. thus, allowing the system to remain equipped with up-to-date software, fixes, etc. OpenHPC provides a guide for the upgrade process.

the installation user guide [10] instructions were followed to upgrade the OpenHPC packages from version 1.3.8 to 1.3.9.

System preparation

Prior to updating the system, as a conservative measure we backed up our user's data to the /BACKUP directory, using the following command:

```
# rsync -avz /home /BACKUP
```

Hence compressing the data and ensuring that symbolic links, devices, attributes, permissions, ownerships, etc. are preserved in the transfer process.

The rsync command was chosen to schedule live incremental updates for the above-mentioned data periodically if necessary.

Also, the `/etc /opt` directories were backed up following the same instructions, to preserve the configuration files, in case a fallback was needed.

the Bash history records were collected and saved to `/home/admin/Documents/ROOT-BASH-HISTORY.txt` to obtain a record for the used command which will be useful for troubleshooting and future references and to increase our productivity.

Clearing the yum Caches

It is often useful to remove entries accumulated in the `/var/cache/yum/` directory [49]. removing a package from the cache, does not affect the copy of the software installed on your system.

The `expire-cache` options eliminate time records of the metadata and mirror-lists download for each repository. This option forces yum to re-validate the cache for each repository the next time it is used.

```
# yum clean expire-cache
```

After running the command, the list of removed metadata files will be printed.

Similarly, to clear the compute node images cached metadata we will run the same command with `CHROOT` as installation destination, as follow:

```
# yum --installroot=$CHROOT clean expire-cache
```

The command will return the number and the name of the removed metadata files.

Upgrade master (SMS) node

Using the package manager yum we can select and upgrade the OpenHPC installed packages. since package builds available from the OpenHPC repositories have “-ohpc” appended to their names so that wild cards can be used as a straightforward way to obtain updates.

```
# yum -y upgrade "*-ohpc"
```

The -*y (--assumeyes)* option, automatically answer yes for all questions. The package manager will resolve the packages dependencies at first, next it will start downloading the packages, after verifying the packages integrity the package manger will start installing the packages, and last delete the downloaded files.

Upon completion a list with the updated packages will be printed.

Upgrade packages in compute image

To upgrade the compute nodes image the package manager was invoked with the `CHROOT` parameter as an installation destination, as follow:

```
# yum -y --installroot=$CHROOT upgrade "*-ohpc"
```

Any new compute-node Base OS provided dependencies can be installed by updating the ohpc-base-compute metapackage.

```
yum -y --installroot=$CHROOT upgrade "ohpc-base-compute"
```

The above command was skipped since no packages in the compute node image were updated in our case [21].

Rebuild image(s)

While the most configuration of provisioned images takes place in the chroot filesystem, such as that generated by `wwmkchroot`, these chroots cannot be directly provisioned by Warewulf. after configuring the chroot image, the `wwvnfs` command is used to encapsulate and compress this filesystem into a VNFS image which can be provisioned. The distinction is similar to that between the source and binary of a compiled program, where the chroot represents the source and the VNFS the binary [22].

To create the VNFS image that the nodes will use to boot, the following command must be invoked:

```
# wwvnfs --chroot $CHROOT
```

The command will print the steps to create the image, and when completed the total elapsed time will be printed.

In the case where packages were upgraded within the chroot compute image, we are required to reboot the compute nodes when convenient to enable the changes.

APPENDIX II

This appendix describes the process of adding argo-c10 node with the statefull provisioning[4] option to the cluster and installing CUDA development tools to the node and the host.

GPU compute node

This section describes the addition of a new computing node (argo-c10) to the cluster, Dell Precision 7920 Rack (Intel Xeon Silver 4114 2.2 GHz, 10Cores with hyperthreading, 16GB (2X8GB) DDR4 2666MHz with Dual Nvidia Quatro P4000, 8GB)

The first step is to add the node to the Warewulf database, and provision the node, next installing the GPU drivers, and CUDA development tools.

Node Setup

This section describes the addition of the node to the Warewulf database and provision the node.

As a first step we deleted the node from the database, since it has incorrect information, after deleting the node from the database, we added the node with the correct MAC address.

```
# wwsh node delete argo-c10
```

The tool `wwsh` is a Warewulf command-line shell interface. This application allows us to interact with the Warewulf backend database and modules via a single interface.

To add a new node to the Warewulf database, the following information are required:

- The node name.
- The node IP address.
- The node MAC address.
- The Back-end Ethernet interface is used for provisioning.

```
# wwsh -y node new ${c_name[10]} --ipaddr=${c_ip[10]} --hwaddr=${c_mac[10]} -D ${eth_provision}
```

As next step we used the `provision` command for setting node provisioning attributes.

The following options are required to provision the node successfully:

- **set** Modify an existing node configuration.
- **vnfs** Define the VNFS that this node should use.
- **bootstrap** Define the bootstrap image that this node should use.
- **files** Define the files that should be provisioned to this node.

[4] The node's current set up is stateless. This appendix describes the previous set up "prior to upgrading to OpenHPC 2.x"

```
# wwsh -y provision set argo-c10 --vnfs=centos7.6 --bootstrap=`uname -r` --files=dynamic_hosts,passwd,group,shadow,network
```

In this stage, we must reboot the compute node so the node will be provisioned with the new OS image. Ipmitool was used to reboot the node as follow:

```
# ipmitool -E -I lanplus -H ${c_bmc[10]} -U ${bmc_username} chassis power reset
```

few minutes are required to reboot the node. To confirm that the node is up initially the `ping` command was used, also we were able to login to the node via `ssh`.

To further confirm the node status, we created a temporary queue on PBS to run jobs on the node, after few modifications the node was able to run jobs on the CPUs.

CUDA Installation on the Host

CUDA development tools are required to enable the users to harness the power of the graphics processing unit (GPU). CUDA is a parallel computing platform and programming model invented by NVIDIA®.

It provides a small set of extensions to standard programming languages, like C, which enable a straightforward implementation of parallel algorithms. CUDA also Supports heterogeneous computation where applications use both the CPU and GPU. Serial portions of applications are run on the CPU, and parallel portions are offloaded to the GPU. As such, CUDA can be incrementally applied to existing applications. The CPU and GPU are treated as separate devices that have their own memory spaces. This configuration also allows simultaneous computation on the CPU and GPU without contention for memory resources [23].

CUDA-capable GPUs have hundreds of cores that can collectively run thousands of computing threads. These cores have shared resources including a register file and a shared memory. The on-chip shared memory allows parallel tasks running on these cores to share data without sending it over the system memory bus.

The installation guide [23] was followed to install and check the correct operation of the CUDA development tools. Selecting the appropriate CUDA version [24] depends on the GPU version available on the node.

The CUDA Toolkit can be installed using either of two different installation mechanisms: distribution-specific packages (RPM and Deb packages), or a distribution-independent package (runfile packages). The distribution-independent package has the advantage of working across a wider set of Linux distributions but does not update the distribution's native package management system. The distribution-specific packages interface with the distribution's native package management system. It is recommended to use the distribution-specific packages, where possible.

To install the appropriate repository meta-data, first we have to determine which distribution and release number are running, by using the following command:

```
uname -m && cat /etc/*release
```

The output of the command shows the current distribution information which will be used to add the NVIDIA® repository.

```
# yum-config-manager --add-repo
http://developer.download.nvidia.com/compute/cuda/repos/rhel7/x86_64/cuda-rhel7.repo
```

After enabling the repository, we installed the CUDA development tools, the Dynamic Kernel Module Support, and CUDA drivers, as follows:

```
# yum clean all

# yum -y install nvidia-driver-latest-dkms cuda

# yum -y install cuda-drivers
```

The list of the new components and the dependencies will be printed after the command completion.

Note: since the host does not contain a CUDA-capable device the drivers aren't necessary to be installed in this section.

To use the development tools the PATH variable needs to include `/usr/local/cuda-10.2/bin` and `/usr/local/cuda-10.2/NsightCompute-<tool-version>`. `<tool-version>` refers to the version of Nsight Compute that ships with the CUDA toolkit, e.g., 2019.1.

To add this path to the PATH variable:

```
export PATH=/usr/local/cuda-10.2/bin:/usr/local/cuda-10.2/NsightCompute-2019.1${PATH:+:${PATH}}
```

To add the environment variable and to add the development tool’s location for all users permanently a file (**cuda.sh**) was created in the **/etc/profile.d** location with the following content:

```
export CUDA_PATH=/usr/local/cuda-10.2

export PATH=$PATH:/usr/local/cuda-10.2/bin

export LD_LIBRARY_PATH=$LD_LIBRARY_PATH:/usr/local/cuda-10.2/lib64/:/opt/cuda/lib64
```

Thus, permanently setting the environments variable for all users.

CUDA Installation on the Compute Node

The Nvidia® drivers are required to utilize the GPU devices on the node, we also installed CUDA development tools in case we needed to compile applications on the node itself.

To determine the type of the GPU devices on the node we can use the `lspci` [25] command to print detailed information about all PCI buses and devices in the system.

```
# lspci | grep -i nvidia
```

the command output will show all Nvidia® installed devices on the node, as follow:

```
65:00.0 VGA compatible controller: NVIDIA Corporation GP104GL [Quadro P4000] (rev a1)

65:00.1 Audio device: NVIDIA Corporation GP104 High-Definition Audio Controller (rev a1)
```

Like the previous section we need to determine the distribution and release number are running, by using the following command:

```
uname -m && cat /etc/*release
```

The output of the command shows the current distribution information which will be used to add the Nvidia® repository.

Note: since nodes are not allowed to access the internet, contrary to the previous section, we followed the distribution-independent package (run-file packages) method of installation.

The verification of the GCC version is required before the installation.

```
# gcc –version
```

The current version is 8.3.0, in most cases the version of gcc installed with a supported version of Linux will work correctly. The major concern is to not receive an error message after running the above command.

In most Linux systems the default graphics drivers are Nouveau. Nouveau [50] is a free and open-source graphics device driver for video cards.

We can determine if the Nouveau drivers are loaded by `lsmod` command. The `lsmod` command shows which loadable kernel modules are currently loaded.

If the Nouveau drivers are not loaded if the following command will print an empty line:

```
[root@argo-c10 ~]# lsmod | grep nouveau
```

Obviously, since the Nouveau drivers are currently loaded on the node, the command will print the following:

```
nouveau              1898794  0
mxm_wmi                13021  1 nouveau
video                  24538  1 nouveau
ttm                    96673  2 mgag200,nouveau
drm_kms_helper        186531  2 mgag200,nouveau
drm                   456166  5 ttm,drm_kms_helper,mgag200,nouveau
i2c_algo_bit           13413  3 igb,mgag200,nouveau
wmi                                                                21636      4
dell_smbios,dell_wmi_descriptor,mxm_wmi,nouveau
```

Since Nouveau does not support CUDA, we must disable the Nouveau drivers and load the Nvidia® drivers instead, by Creating a file at **/etc/modprobe.d/blacklist-nouveau.conf** with the following contents:

```
blacklist nouveau
options nouveau modeset=0
```

After rebooting the node, we can ensure that the Nouveau drivers has been disabled by using the above-mentioned command.

```
[root@argo-c10 ~]# lsmod | grep nouveau
```

Since the command didn't return the loaded module confirming that Nouveau has been disabled.

To install CUDA development tools, we must change the run-level to text only mode [29] by using the `init` command.

```
[root@argo-c10 ~]# init 3

[root@argo-c10 ~]# runlevel
```

```
5 3
```

The `runlevel` command shows the change from level 5 (graphical mode) to level 3 (text only mode). After taking the per-installation measures, we downloaded the appropriate CUDA development package in the host and moved the file to the node using the `scp` command to move the file over SSH. The installation package was moved to a temporary directory ~/ c10_paks.

To initiate the installer, we must run the file cuda_10.2.89_440.33.01_linux.run from the file location by:

```
[root@argo-c10 c10_paks]#sh cuda_10.2.89_440.33.01_linux.run
```

After initiation a welcome screen will be shown, the user must follow the on-screen prompts to install the tools.

At first the installation failed to complete with the following error:

```
Installation failed. See log at /var/log/cuda-installer.log for details.
```

Upon examining the cuda-installer.log we can see that the driver’s installation has failed.

......

```
[INFO]: Components to install:

[INFO]: Driver

[INFO]: 440.33.01

[INFO]: Executing NVIDIA-Linux-x86_64-440.33.01.run --ui=none --no-questions --accept-license --disable-nouveau --no-cc-version-check --install-libglvnd 2>&1

[INFO]: Finished with code: 256

[ERROR]: Install of driver component failed.

[ERROR]: Install of 440.33.01 failed, quitting
```

To see the error source, we had to examine the nvidia-installer.log

......

```
ERROR: Unable to find the development tool `cc` in your path; please make sure that you have the package 'gcc' installed.  If gcc is installed on your system, then please check that `cc` is in your PATH.

ERROR: Installation has failed.  Please see the file '/var/log/nvidia-installer.log' for details.  You may find suggestions on fixing installation problems in the README available on the Linux driver download page at www.nvidia.com.
```

To resolve this error (and the next similar errors) we created symbolic links for the CC, CXX, CPP, and LD environments.

```
[root@argo-c10 c10_paks]# export CC=/opt/ohpc/pub/compiler/gcc/8.3.0/bin/gcc
[root@argo-c10 c10_paks]# export CXX=/opt/ohpc/pub/compiler/gcc/8.3.0/bin/g++
[root@argo-c10 c10_paks]# export CPP=/opt/ohpc/pub/compiler/gcc/8.3.0/bin/cpp
[root@argo-c10 c10_paks]# export LD=/opt/ohpc/pub/compiler/gcc/8.3.0/bin/gcc

[root@argo-c10 c10_paks]# alias c++=/opt/ohpc/pub/compiler/gcc/8.3.0/bin/c++
[root@argo-c10 c10_paks]# alias g++=/opt/ohpc/pub/compiler/gcc/8.3.0/bin/g++
[root@argo-c10 c10_paks]# alias gcc=/opt/ohpc/pub/compiler/gcc/8.3.0/bin/gcc
```

```
[root@argo-c10 c10_paks]# alias cpp=/opt/ohpc/pub/compiler/gcc/8.3.0/bin/cpp
[root@argo-c10 c10_paks]# alias ld=/opt/ohpc/pub/compiler/gcc/8.3.0/bin/gcc
[root@argo-c10 c10_paks]# alias cc=/opt/ohpc/pub/compiler/gcc/8.3.0/bin/gcc
```

We were faced with a second installation error when we attempted to re-run the installer.

After examining the nvidia-installer.log file we can see that the `make` command is missing. `make` is an essential utility to execute programs and libraries from source code.

......

```
ERROR: Unable to find the development tool `make` in your path; please make sure that you have the package 'make' installed.  If make is installed on your system, then please check that `make` is in your PATH.
```

To solve this error, we downloaded the `make` rpm file (along with the required dependencies) on the host and moved them to the node using `scp`.

And installed them on the node using the yum `localinstall` option.

As follow: `yum localinstall /path/to/file.rpm`

Note: this error was expected since the node OS image is a minimal image lacking most of the development tool application.

In future upgrade this error will be avoided by installing the development tool package into the OS image.

Alongside make we had to install several tools and essential libraries to install the CUDA development tools, by the Trial-and-error method each time running the installer and refer to the log file to locate and solve the error.

The last error we faced in this stage was the following:

......

```
RROR: Unable to find the kernel source tree for the currently running kernel. Please make sure you have installed the kernel source files for your kernel and that they are properly configured; on Red Hat Linux systems, for example, be sure you have the 'kernel-source' or 'kernel-devel' RPM installed. If you know the correct kernel source files are installed, you may specify the kernel source path with the '--kernel-source-path' command line option.
```

To solve this error, we had to specify the current kernel version to the installer as following:

```
[root@argo-c10 c10_paks]# sh cuda_10.2.89_440.33.01_linux.run –kernel-source-path=/usr/src/kernels/3.10.0-1062.el7.x86_64
```

Installation summary will be printed on the screen indicating a successful installation.

After rebooting the node, to set the environment variables the PATH and LD_LIBRARY_PATH variables must be modified as follow:

```
[root@argo-c10 ~]# export PATH=/usr/local/cuda-10.2/bin${PATH:+:${PATH}}
[root@argo-c10 ~]# export LD_LIBRARY_PATH=/usr/local/cuda-10.2/lib64${LD_LIBRARY_PATH:+:${LD_LIBRARY_PATH}}
```

to make system-wide permanent changes to the PATH variables we added the cuda.sh script to the directory etc/profile.d/ so the path to the CUDA utilities whenever a bash shell is entered, as well as when the desktop session loads.

After the initial setup, the node showed a couple of errors (in Nagios & Ganglia monitoring tools), we will address this issue in a dedicated segment.

Configuring PBS for Basic GPU Scheduling

To use the GPU devices on the node we must configure PBS to run applications on the node. We used a Basic scheduling scheme to configure PBS. Basic scheduling consists of prioritizing jobs based on partition or site policies, controlling access to nodes with GPUs, ensuring that GPUs are not over-subscribed, and tracking the use of GPUs in accounting logs [26].

Configuring PBS to perform basic scheduling of GPUs is relatively simple, and only requires defining and configuring a single custom resource to represent the number of GPUs on each node.

This method allows jobs to request unspecified GPUs. Jobs should request exclusive use of the node to prevent other jobs from being scheduled on their GPUs. PBS Professional Administrator's Guide [27] was followed to configure PBS.

To modify PBS configuration at first we must stop the server and scheduler.

```
[root@argo sched_priv]# systemctl stop pbs
```

the next step is to create GPU resources on the node. The name ngpus was used, as recommend by the Administrator's Guide [27]. vnode value is Set to the number of GPUs on the node.[5]

To modify the PBS configuration, the `qmgr` command provides an administrator interface to query and configure batch system parameters.

```
Qmgr: create resource ngpus type=long, flag=nh
```

as next step we must edit PBS_HOME/sched_priv/sched_config file, to add ngpus to the list of scheduling resources as following:

```
resources: "ncpus, mem, arch, host, vnode, ngpus"


source /etc/pbs.conf ; edit $PBS_HOME/sched_priv/sched_config 1

/var/spool/pbs/sched_priv/sched_config
```

After changing the scheduler's configuration file, the scheduler must re-read it for the changes to take effect. To get a scheduler to re-read its configuration information, without stopping the scheduler, we have to HUP the scheduler:

```
Qmgr: kill -HUP <scheduler PID>
```

Note: If you set a scheduler attribute using qmgr, the change takes effect immediately and you do not need to HUP the scheduler.

[5] The ngpus resource is used exactly the way you use the ncpus resource.

To add the number of GPU devices available to each execution host in the cluster via qmgr:

```
Qmgr: set node HostA resources_available.ngpus=2
```

Specifying the node name while running the pbsnodes command shows the resources that have been added to the node:

```
[root@argo sched_priv]# pbsnodes argo-c10
```

The output shows that 2 GPU resources has been added to the node

```
argo-c10
     Mom = argo-c10.localdomain
     Port = 15002
     pbs_version = 19.1.3
     ntype = PBS
     state = free
     pcpus = 20
     resources_available.arch = linux
     resources_available.host = argo-c10
     resources_available.mem = 16000200kb
     resources_available.ncpus = 20
     resources_available.ngpus = 2
     resources_available.vnode = argo-c10
     resources_assigned.accelerator_memory = 0kb
     resources_assigned.hbmem = 0kb
     resources_assigned.mem = 0kb
     resources_assigned.naccelerators = 0
     resources_assigned.ncpus = 0
     resources_assigned.vmem = 0kb
     queue = testq
     resv_enable = True
     sharing = default_shared
     last_state_change_time = Sun May 10 22:56:21 2020
     last_used_time = Mon Apr 20 22:59:03 2020
```

After configuring the device, a test queue (GPUq) was made to run a CUDA application on the node.

The `qmgr` tool was used to create the queue:

```
Qmgr: create queue GPUq queue_type=execution
```

The newly created queue can be displayed by using the `list` option for confirmation.

```
Qmgr: list queue GPUq
```

The command output will show the queue details.

```
Queue GPUq
```

```
    queue_type = Execution

    total_jobs = 0

    state_count = Transit:0 Queued:0 Held:0 Waiting:0 Running:0 Exiting:0
Begun:0
```

The next step is to add the desirable nodes to the specific queue (in this case is argo-c10)

```
Qmgr: set node argo-c10 queue=GPUq
```

The last step is to enable and start the queue for execution.

```
Qmgr: set queue GPUq enabled = True

Qmgr: set queue GPUq started = True
```

We can verify the made changes by using the `list` option, as perilously mentioned.

```
Qmgr: list queue GPUq

Queue GPUq

    queue_type = Execution

    total_jobs = 0

    state_count = Transit:0 Queued:0 Held:0 Waiting:0 Running:0 Exiting:0
Begun:0

    hasnodes = True

    enabled = True

    started = True
```

APPENDIX III

PBS new syntax

In early versions of PBS, job submitter used “-l nodes=...” in what was called a “node specification” to specify where the job should run. The syntax for a “node specification” is **deprecated** [28].

Old versions of PBS allowed job submitter to ask for resources outside of a select statement, using “-lresource=value”, where those resources must now be requested in chunks, inside a select statement. This old-style of resource request was called a “resource specification”.
For backward compatibility, any resource specification is converted automatically to the new syntax. However, job tasks may execute in an unexpected order, because vnodes may be assigned in a different order.
To obtain a more robust system we upgraded the user instructions [8] following PBS new syntax.

Users Instructions for the HPC Cluster Argo

Step 1: Authorization for challenge requests

1. With Browser connect to Web Address https://argo-rbs.cloud.iasa.gr
2. Select (Not a member) Register
3. Enter the following details on the next screen.
   a. Full Name (English Characters)
   b. Username (used to login)
   c. Email address (for communication)
   d. Password + Confirmation
   e. CAPTCHA.
4. Make a note of the password you entered.
5. Click on Register.
6. You immediately will receive an email from Argo -Resources Booking System-Registration Confirmation (it@iasa.gr) with instructions to activate the account.

Step 2: Challenge Request

1. After activating your account, log in to the Web Page https://argo-rbs.cloud.iasa.gr (username + password from Step 1)
2. Select "subscribe to challenges.”
3. User::Info instructions will be printed on the screen.
4. Upload your public key (copy paste all contents of public key file - text) and submit.

Caution, because there is still no automatic sanitization trigger in the database, paste the public key on a continuous line, without line breaks, especially if you are using Windows OS.

5. Click “subscriptions section”.
6. Make request to Subscribe to available challenges (only one is displayed for the whole year)
7. You will receive an email when your request is approved (requires my "approval", it is not automatic)
8. After the Request Granted you are given an argoxxx account for login to the front-end machine with access to 11 nodes (90 cores).[6]

Step 3: Login to Front-End of Argo (VPN connection required)

With the provided account (username) argoxxx, log in to the front-end of the parallel machine from your computer console using the Private Key and Passphrase (without another password), as follows

I. From linux console using the command

   ssh -i [path_to_your_private_key] argoxxx@argo-rbs.cloud.iasa.gr

   Prompts you to enter Passphrase.

II. From Windows with putty, two options are available

   - In the Run command (Wind + R) run the command

     putty.exe –ssh –i [path_to_your_private_key] argoxxx@argo-rbs.cloud.iasa.gr

     a front-end console will open, with your username asking for the Passphrase.

   - Run the pageant (putty authentication agent), an icon will appear in the task bar. open it, enter the location of the private key file, at this point the program will require the passphrase. open putty (or WinSCP) and by connecting to argo-rbs.cloud.iasa.gr it will automatically login and authenticate.

In the first session, copy the MpiDemo directory to your account by running the command:

```
[user @ argo] # cp -r / etc / skel2 / MpiDemo.
```

The folder contains the following:

1. The Commands.txt file from where you can copy-paste the following commands (for your convenience). copying from this pdf is not recommended.
2. MPI program mpi_trapDemo.c and hybrid MPI + OpenMp mpiH_trapDemo.c useful as a demonstration. The program calculates the integral f (x) = x^2 in the interval 0..3 with the method of tables for 2^20 tables.

[6]Steps 1, 2 are done only once at the beginning of the semester. The remaining steps describe the standard process of compiling and executing mpi and hybrid mpi + openmp programs using the demo program.

3. The mpiPBSscript.sh script for running the compiled mpi_trapDemo.c and the mpihPBSscript.sh script for the compiled mpiH_trapDemo.c

At this point you are ready to compile (Step 4) and run (Step 5) the programs

Step 4. Compiling MPI Applications

Initially, to compile MPI programs, you need to make sure that the OpenMPI module is loaded (enabled). By using the command:

```
[user @ argo] # module list
```

you can see which modules are loaded. You can load a module by executing:

```
[user @ argo] # module load name_of_module
```

Specifically, for loading OpenMpi, run:

```
[user @ argo] # module load openmpi3
```

After loading OpenMpi, to compile an Mpi program, use the mpicc compiler, with the same syntax as gcc. In our example:

```
[user @ argo] # mpicc -O3 mpi_trapDemo.c -o mpi_trapDemo.x
```

The executable file mpi_trapDemo.x will be created.

To compile MPI programs with profiling with MpiP, first load the MpiP module:

```
[user @ argo] # module load mpiP
```

and to compile, run:

```
[user @ argo] # mpicc -O3 -g mpi_trapDemo.c -L $ MPIP_DIR /
lib -lmpiP -lbfd -lunwind -o mpi_trapDemoP.x
```

To compile OpenMP or hybrid MPI + OpenMp programs (more in 5.2), use the flag -fopenmp, for example:

```
[user @ argo] # mpicc -O3 -fopenmp mpiH_trapDemo.c -o
mpiH_trapDemo.x
```

Step 5. Run MPI Programs

To run the application, the PBS resource manager is used, which works with a queue system. You will submit the job to be executed and it will be executed when it is your turn. a request to the queue is made by a script with a .sh extension, which contains all the information and parameters for the execution. In the MpiDemo directory, there are two scripts, in which you can see some basic commands, which will be analyzed in 5.1 and 5.2.

To register a job in the queue run the following command:

```
[user @ argo] # qsub myPBSScript.sh
```

the command will return the jobID (int) or an error message in the script. To view your job status (Q-standby, R-Running):

```
[user @ argo] # qstat
```

If jobID is not in the list, the job is already done (it is quite common for small programs). In case you want to cancel the execution, delete with:

```
    [user @ argo] # qdel <jobID>
```

When the job is finished, two files will be created:

- myJob.o <id> for the output of the job [stdout]
- myJob.e <id> for errors during the execution [stderr]

Where “myJob” is defined in the script (see 5.1)

Following is an analysis of the Structure and Use of the PBS script for pure MPI programs (5.1) and hybrid MPI + OpenMp (5.2).

5.1 The Structure of PBS Script for pure MPI programs, in blue color the values that you can change are presented depending on your application parameters:

#! / bin / bash

#Max VM size, Example, 2GB per process - max 16 GB per node #

#PBS -l pvmem = 2G

# Max Wall time

#PBS -l walltime = 00: 01: 00 # Example, 1 minute

# Reserve (select) Number of Nodes and cpus / Node. Slots mpiprocs / Node,

#Example: reserve 2 nodes and 4 cpus / node running with max 8 mpi tasks per node

#PBS -l select = 2: ncpus = 4: mpiprocs = 8

#Limits (ARGO) select = 1..10; ncpus = 1..8; mpiprocs = 1 .. <any>

If you request more than 10 nodes or more than 8 ncpus the request will remain in the queue, as PBS cannot allocate the resources. Mandatory qdel <JobID>. If the job you entered remains in standby mode (Q) and there is no running job (R) it means that there is an error in the PBS script. Delete the job and check the script for correction before re-submitting.

# Only this job uses the chosen nodes (exclusive use of the node from the current # job.)

# You can also use the -l place = shared option that allows the use of resources

# By multiple jobs.

#PBS -l place = excl

# Stdout Output File, if not provided a <JobName>.o <JobID> will be created #

#PBS -o job.out

# Stderr Output File, if not provided a <JobName>.e <JobID> will be created #

#PBS -e job.err

#Queue for MPI and MPI + OpenMp hybrids #

#PBS -q workq

# Job Name - helps to distinguish jobs with mnemonic name #

#PBS -N myJob

#Change Working directory to SUBMIT directory. THIS IS MANDATORY, PBS Starts everything from $ HOME, one should change to submit directory

#cd $ PBS_O_WORKDIR

mpirun ring.x # That was compiled on front-end.

#Runs 16 processes (from select: 2 nodes X 8 mpi slots) on 4 cpus / node

Examples for execution configuration:

1. To create 32 MPI processes on 2 nodes of 8 cores (cpus) each and 16 processes / node:

# 2 nodes with 8 mpi tasks / node => 2 * 16 tasks with simple mpirun call #

#PBS -l select = 2: ncpus = 8: mpiprocs = 16

mpirun <executable>

2. To create M processes of non-multiples of 8 (M mod 8 ≠ 0), e.g., squares 25, 36, 49 etc. bind select = N (N = ⌊ M / 8 ⌋ -round-up), ncpus = 8, mpiprocs = P (P = ⌊ M / N ⌋ -round-up)

#PBS -l select = N: ncpus = 8: mpiprocs = P

# BUT we execute mpirun with the switch –np <number of processes> #

mpirun -np M <executable>

e.g., For 25 processes we bind 4 nodes (25/8 = 3.125), select N = 4 and mpiprocs = 7

#PBS -l select = 4: ncpus = 8: mpiprocs = 7 # distribution [7,7,7,4]

mpirun -np 25 <executable>]

The following will also work (but does not specify the best load balancing)

#PBS -l select = 4: ncpus = 8: mpiprocs = 8 # with automatic process allocation [8,8,8,1]

For 36 processes ⇒ 5 nodes, 36/5 = 4.5, N = 5 [8,8,8,8,4]

#PBS -l select = 5: ncpus = 8: mpiprocs = 8

mpirun -np 36 <executable>

It is recommended that you always block ncpus = 8 and share processes with mpiprocs

5.2 Customize PBS script for MPI + OpenMp Hybrid Programs

For hybrid applications the following are required: 1) to be able to create less than 8 processes in each node and 2) to create threads in each process and 3) to combine 1 + 2.

1. To create less than 8 MPI processes / nodes, let P each of N nodes, i.e., a total of P * N processes:

#PBS -l select = N: ncpus = 8: mpiprocs = P # P = 1,2,4 but we can have more 8,…

e.g., Four nodes and two processes / node with commitment of all (8) of cpus

#PBS -l select = 4: ncpus = 8: mpiprocs = 2

mpirun <executable>

2. Having defined the number of processes per node we can define in the PBS script the default number of T threads that each process creates with:

#PBS -l select = N: ncpus = 8: mpiprocs = P: ompthreads = T

T can be T ≥ 1 with a ceiling depending on the architecture, typically close to (excessive) 109. In practice we will consider T = 2, 4, 8, a maximum of 16 threads per process. If we do not set ompthreads the default value will be ompthreads = ncpus

We specify that with ompthreads we set the default number of threads that a program creates, and we can find it in the OpenMp program with

```
int omp_get_num_threads (void);
```

Of course, we can set this number in the OpenMp program with:

```
void omp_set_num_threads (int num_threads);
```

or even temporarily define it with the attribute num_threads (T1), e.g.

```
#pragma omp parallel num_threads (3)
```

which will ignore the T value you set in the script.

Therefore, ompthreads is only for our convenience, so that you have fewer parameters in your program or to avoid compilation due to changing thread numbers in the measurements.

3. To run an MPI + OpenMp hybrid program we have to combine 1) creating processes per node and 2) defining the predefined number of threads per process. To create P processes per node and T threads per process in N nodes with 8 cpus, define:

```
#PBS -l select = N: ncpus = 8: mpiprocs = P: ompthreads = T
mpirun <executable>
```

e.g., For 3 nodes, 8 cores, 2 processes / node and 4 threads per process

```
#PBS -l select = 3: ncpus = 8: mpiprocs = 2: ompthreads = 4
mpirun <executable>
```

e.g., To request two nodes, each with eight CPUs and eight MPI processes with four threads each:

```
#PBS -l select = 2: ncpus = 8: mpiprocs = 8: ompthreads = 4
```

APPENDIX IV

Instructions for Using Argo Cluster for CUDA Programs Development

ARGO consists of the front-end (Dell OptiPlex 7050, Quad-Core Intel-Core i5-6500 @ 3.20GHz, 16GB DDR4 2.4GHz) to which users are connected to utilize 11 compute nodes in a rack for running parallel programs. The 10 Dell Poweredge compute nodes runs MPI and hybrid MPI + OpenMp programs. The 11th Dell Precision 7920 Rack (Intel Xeon Silver 4114 2.2 GHz, 10Cores with hyperthreading, 16GB (2X8GB) DDR4 2666MHz with Dual Nvidia Quatro P4000, 8GB) runs CUDA and OpenMp + CUDA programs.

The cluster is available for Testing, Development, Debugging, Measurements, etc. to the students and researchers of the department, it has no restrictions, time, availability of resources, etc. Please, but no abuses, the resources are for everyone.

Log in to your front-end account and copy the CUDA-Demo folder to your account by running the command:

```
[user@argo]# cp -r /etc/skel2/CUDA-Demo .
```

The folder contains two simple applications for the purpose of presenting a compilation of CUDA programs and how to run it at the node c10 of the cluster, which has 2 GPUs. Each application is located in a separate directory (simple-add, dot-product). Together with a script (myPBSscript.sh) for its execution.

In addition, there are two applications in subdirectories (simple_add1GPUOmp and simple_add2GPUOmp) which are modifications of the original application (simple-add) using the OpenMp model to run multiple threads (simple_add1GPUOmp) and using the two GPUs in parallel (simple_add2).

Compilation of CUDA programs

The compilation of CUDA programs is made using the nvcc compiler, provided by Nvidia, as follows:

```
nvcc "flags" inputfiles -o outputfile
```

for example:

```
[user@argo]# nvcc dot-product.cu -o dot-product
```

To use the OpenMp model, it is necessary to use the -fopenmp flag in the compiler and in the linker, to compile the program. The use of the -fopenmp flag in the nvcc compiler is as follows:

```
nvcc -X -fopenmp inputfile.cu -o outputfile
```

Note: you can compile the simple-add, simple_add1GPUOmp and simple_add2GPUOmp applications using the make command, where the make tool will read the properties (compiler flags, input files) from the Makefile file. After the compilation is finished, the make tool returns the object files and the executable. You can modify the Makefile for your own applications in order to compile them with the make command, replacing the input and output file names according to your application.

Execution of CUDA programs

As with MPI programs, we use PBS, which runs on a queue system, to run CUDA programs. You will enter the job to be executed and it will be executed when it is your turn in the queue. A script with a .sh extension, is used to submit jobs which contains all the information and parameters for the desired execution.

To register a job in the queue you perform:

```
[user@argo]# qsub myPBSScript.sh
```

in this phase the shell returns the following:

```
jobID.argo
```

where jobID is a unique five-digit number that refers to the job we entered in the execution queue. The jobID changes with each entry.

After running the script, you will see two new files in the directory (myGPUJob.o [jobID] and myGPUJob.e [jobID). The file myGPUJob.o [jobID] records the program output (stdout). If a runtime error occurs, the error output (stderr) is logged in the myGPUJob.e [jobID] file. For example, after running the simple-add program, the myGPUJob.o [jobID] file will have the following content:

```
Simple vector addition example (15000000 elements)
Total GPU memory -75956224, free -167706624

Vector addition on CPU
Execution time 17.02 msecs
Bandwith 9.848318 GB/sec
Printing last 6 elements
-466735552.0 -496735552.0 -526735552.0 -556735552.0 -586735552.0 -616735488.0

Vector addition on GPU
Execution time 1.19 msecs
Bandwith 141.228363 GB/sec
Printing last 6 elements
-466735552.0 -496735552.0 -526735552.0 -556735552.0 -586735552.0 -616735488.0
~
"myGPUJob.o59154" 14L, 456C
```

The Structure of PBS Script for CUDA programs

The value that you can be changed on a case-by-case basis are highlighted:

```
#!/bin/bash
#Which Queue to use
#PBS -q GPUq

#Max Wall time, Example 1 Minute #
#PBS -l walltime=00:01:00

#How many nodes and tasks per node, Example 1 node with 1 CPU and 1 GPU#
#PBS -l select=1:ncpus=1:ngpus=1

# Only this job uses the chosen nodes
#PBS -l place=excl

#JobName #
#PBS -N myGPUJob

#Change Working directory to SUBMIT directory#
cd $PBS_O_WORKDIR

# Run executable #
./simple-add
```

Collection of information of CUDA programs with nvprof

The nvprof profiling program allows the collection and display of CUDA program execution data on the CPU and GPU, e.g., kernel execution, memory transfers, etc.

The data collection is as follows:

```
bash$ nvprof ./executable
```

To use nvprof you need to modify myPBSscript.sh to run the application using nvprof. As an example, for collecting information about the simple-add program we need to change the last line in the executable of the script:

```
# Run executable #
nvprof ./simple-add
```

After running the program, the file myGPUJob.e [jobID] contains data recorded by nvprof, for example simple-add the output will be similar to:

```
==92147== NVPROF is profiling process 92147, command: ./simple_add
==92147== Profiling application: ./simple_add
==92147== Profiling result:
         Type  Time(%)      Time     Calls       Avg       Min       Max  Name
 GPU activities:   67.81%  16.992ms         2  8.4962ms  8.2164ms  8.7759ms  [CUDA memcpy HtoD]
                   27.60%  6.9176ms         1  6.9176ms  6.9176ms  6.9176ms  [CUDA memcpy DtoH]
                    3.49%  874.04us         1  874.04us  874.04us  874.04us  kadd(float*, float*, float*, int)
                    1.10%  275.30us         1  275.30us  275.30us  275.30us  [CUDA memset]
      API calls:   85.51%  192.32ms         1  192.32ms  192.32ms  192.32ms  cudaMemGetInfo
                   10.83%  24.349ms         3  8.1163ms  7.0949ms  8.8986ms  cudaMemcpy
                    2.15%  4.8280ms         3  1.6093ms  253.12us  2.2961ms  cudaFree
                    0.50%  1.1275ms         1  1.1275ms  1.1275ms  1.1275ms  cudaThreadSynchronize
                    0.33%  751.59us         3  250.53us  187.04us  358.72us  cudaMalloc
                    0.33%  744.14us         2  372.07us  369.39us  374.75us  cuDeviceTotalMem
                    0.28%  619.08us       194  3.1910us     271ns  128.65us  cuDeviceGetAttribute
                    0.03%  59.868us         2  29.934us  25.861us  34.007us  cuDeviceGetName
                    0.03%  57.410us         1  57.410us  57.410us  57.410us  cudaMemset
                    0.01%  20.936us         1  20.936us  20.936us  20.936us  cudaLaunchKernel
                    0.01%  17.997us         2  8.9980us  2.3010us  15.696us  cuDeviceGetPCIBusId
                    0.00%  3.4700us         3  1.1560us     411ns  2.3260us  cuDeviceGetCount
                    0.00%  1.8730us         4     468ns     334ns     669ns  cuDeviceGet
                    0.00%  1.0470us         2     523ns     441ns     606ns  cuDeviceGetUuid
                    0.00%     248ns         1     248ns     248ns     248ns  cudaGetLastError
```

Customize PBS script for OpenMp + CUDA Hybrid Programs (1 or 2 GPUs)

1. OpenMp single GPU

In the simple_add1GPUOmp application, CPU operations are performed on multiple threads. The GPUq queue allows the use of up to ncpus = 20 cores (there are 10 cores, but due to hyperthreading they look 20). The control of the number of threads is done by modifying the script, in particular it is controlled by the value of the variable ompthreads = xx or the variable ncpus = xx in case the variable ompthreads is not defined. For example, the following script allows the application to utilize 20 threads in 10 cores.

```
#!/bin/bash
# Which Queue to use, DO NOT CHANGE #
#PBS -q GPUq

# Max Wall time, Example 1 Minute #
```

```
#PBS -l walltime=00:01:00

# How many nodes and tasks per node,  1 node with 10 cpus 2 tasks(threads) per cpu#
#PBS -lselect=1:ncpus=10:ompthreads=20:ngpus=1

# Only this job uses the chosen nodes
#PBS -lplace=excl

# JobName #
#PBS -N myGPUJob

#Change Working directory to SUBMIT directory
cd $PBS_O_WORKDIR

# Run executable #
./simple_add
```

the output of the program shows the execution time in one thread and 8 threads respectively:

```
Simple vector addition example (15000000 elements)
Total GPU memory -75956224, free -194969600

Vector addition on CPU
Execution time 16.76 msecs

Vector addition on CPU (multithreads)
Execution time 7.86 msecs

Vector addition on GPU
Execution time 1.15 msecs
```

2. OpenMp Multiple GPU

The simple_add2GPUOmp application utilizes the two graphics cards in the node to run the CUDA kernel. Using the OpenMp model the program is branched into two threads, and each thread calls a GPU to process a portion of the data.

To view the data for the two graphics cards from nvprof we use the option -print-summary-per-gpu. As presented in the following script.

```
#!/bin/bash

# Which Queue to use, DO NOT CHANGE #
#PBS -q GPUq

# Max Wall time, Example 1 Minute #
#PBS -l walltime=00:01:00

# How many nodes and tasks per node, 1 node with 20 tasks/threads 2 GPU
#PBS -lselect=1:ncpus=20:ompthreads=20:ngpus=2 -lplace=shared

# Only this job uses the chosen nodes
#PBS -lplace=excl

# JobName #
#PBS -N myGPUJob

#Change Working directory to SUBMIT directory
cd $PBS_O_WORKDIR
```

```
# Run executable #
nvprof --print-summary-per-gpu ./simple_add
```

After the execution of the program in the file myGPUJob.e [jobID] the data of the two GPUs are recorded as in the following example:

```
==162941== NVPROF is profiling process 162941, command: ./simple_add
==162941== Profiling application: ./simple_add
==162941== Profiling result:

==162941== Device "Quadro P4000 (0)"
            Type  Time(%)     Time     Calls       Avg       Min       Max  Name
 GPU activities:   56.09%  10.631ms        2  5.3156ms  5.1780ms  5.4531ms  [CUDA memcpy HtoD]
                   40.87%  7.7475ms        1  7.7475ms  7.7475ms  7.7475ms  [CUDA memcpy DtoH]
                    2.31%  437.56us        1  437.56us  437.56us  437.56us  kadd(float*, float*, float*, int)
                    0.73%  139.19us        1  139.19us  139.19us  139.19us  [CUDA memset]

==162941== Device "Quadro P4000 (1)"
            Type  Time(%)     Time     Calls       Avg       Min       Max  Name
 GPU activities:   65.24%  9.6868ms        2  4.8434ms  4.8413ms  4.8455ms  [CUDA memcpy HtoD]
                   30.88%  4.5855ms        1  4.5855ms  4.5855ms  4.5855ms  [CUDA memcpy DtoH]
                    2.94%  436.28us        1  436.28us  436.28us  436.28us  kadd(float*, float*, float*, int)
                    0.94%  139.03us        1  139.03us  139.03us  139.03us  [CUDA memset]
```

The output of the program is presented the execution time in 2 threads in the processor and the two graphics cards:

```
Simple vector addition example (15000000 elements)
Total GPU memory -75956224, free -194969600

Vector addition on CPU
Execution time 16.52 msecs

Vector addition on GPU
CPU(0)|GPU(0):Execution time 0.58 msecs
```

Note: The kernel execution time at program output differs from nvprof data because it is calculated by the processor "includes calling the kernel and returning control".

APPENDIX V

Kernel update

This appendix describes the steps performed for a kernel update on the cluster. The kernel is the most important component of any Linux operating system. A Linux kernel works as the interpreter or mediator between computer hardware and software processes.

Each Linux distribution is shipped with a stable version of Linux Kernel and the supported software and drivers. But the shipped Kernel may not be the latest one.

One needs to upgrade the whole operating system to move to another major version of Linux Kernel.

Host Kernel update

The current Kernel version can be checked using the following command:

```
[root@argo ~]# uname -r
4.18.0-193.19.1.el8_2.x86_64
```

To check for available update, including the availability of the latest Kernel update the yum command can be used.[7]

```
[root@argo ~]# yum check-update
Last metadata expiration check: 1:56:51 ago on Tue 12 Jan 2021 05:09:01 PM EET.

......

kernel.x86_64                     4.18.0-240.1.1.el8_3                          BaseOS
kernel-core.x86_64                4.18.0-240.1.1.el8_3                          BaseOS
kernel-devel.x86_64               4.18.0-240.1.1.el8_3                          BaseOS
kernel-headers.x86_64             4.18.0-240.1.1.el8_3                          BaseOS
kernel-modules.x86_64             4.18.0-240.1.1.el8_3                          BaseOS
kernel-tools.x86_64               4.18.0-240.1.1.el8_3                          BaseOS
kernel-tools-libs.x86_64          4.18.0-240.1.1.el8_3                          BaseOS
......
```

To Install the kernel package by using the following command. You can also install any other kernel-* package according to the system requirement, or to install them separately as following:

```
[root@argo ~]# yum update kernel
[root@argo ~]# yum update kernel-devel
[root@argo ~]# yum update kernel-headers
[root@argo ~]# yum update kernel-tools kernel-tools-libs
```

Next step is to load the latest Kernel version as default during boot time using the `grubby` command to list the available Kernels and change the default kernel.

```
[root@argo ~]# grubby --info=ALL | grep title
title="CentOS Linux (4.18.0-240.1.1.el8_3.x86_64) 8"
title="CentOS Linux (4.18.0-193.19.1.el8_2.x86_64) 8 (Core)"
title="CentOS Linux (4.18.0-193.el8.x86_64) 8 (Core)"
title="CentOS Linux (0-rescue-1694ddb46dce458baa75a96468630089) 8 (Core)"

[root@argo ~]# grubby --info ALL | grep id
```

[7] We can update all the available packages using yum update, but we choose to update he Kernel only to avoid any dependency mismatch.

```
args="ro       crashkernel=auto       resume=UUID=80ef1b3b-92de-42f1-a549-95791639ebd7
rd.md.uuid=2db9eef3:c3ee4fef:f141f853:62cf3afb
rd.md.uuid=d1ea18d9:80cc314b:4bcb0d00:3613284f  rhgb  quiet  rd.driver.blacklist=nouveau
$tuned_params"
id="1694ddb46dce458baa75a96468630089-4.18.0-240.1.1.el8_3.x86_64"
args="ro       crashkernel=auto       resume=UUID=80ef1b3b-92de-42f1-a549-95791639ebd7
rd.md.uuid=2db9eef3:c3ee4fef:f141f853:62cf3afb
rd.md.uuid=d1ea18d9:80cc314b:4bcb0d00:3613284f  rhgb  quiet  rd.driver.blacklist=nouveau
$tuned_params"
id="1694ddb46dce458baa75a96468630089-4.18.0-193.19.1.el8_2.x86_64"
args="ro       crashkernel=auto       resume=UUID=80ef1b3b-92de-42f1-a549-95791639ebd7
rd.md.uuid=2db9eef3:c3ee4fef:f141f853:62cf3afb
rd.md.uuid=d1ea18d9:80cc314b:4bcb0d00:3613284f  rhgb  quiet  rd.driver.blacklist=nouveau
$tuned_params"
id="1694ddb46dce458baa75a96468630089-4.18.0-193.el8.x86_64"
args="ro       crashkernel=auto       resume=UUID=80ef1b3b-92de-42f1-a549-95791639ebd7
rd.md.uuid=2db9eef3:c3ee4fef:f141f853:62cf3afb
rd.md.uuid=d1ea18d9:80cc314b:4bcb0d00:3613284f rhgb quiet rd.driver.blacklist=nouveau"
id="1694ddb46dce458baa75a96468630089-0-rescue"
```

The vmlinuz files are located in the boot directory. vmlinuz is the name of the Linux kernel executable is a compressed Linux kernel, and it loads the OS into memory.

```
[root@argo ~]# ls /boot/
config-4.18.0-193.19.1.el8_2.x86_64                     lost+found/
config-4.18.0-193.el8.x86_64                                         System.map-4.18.0-
193.19.1.el8_2.x86_64
config-4.18.0-240.1.1.el8_3.x86_64                                   System.map-4.18.0-
193.el8.x86_64
efi/                                                                 System.map-4.18.0-
240.1.1.el8_3.x86_64
grub2/                                                                vmlinuz-0-rescue-
1694ddb46dce458baa75a96468630089
initramfs-0-rescue-1694ddb46dce458baa75a96468630089.img                 vmlinuz-4.18.0-
193.19.1.el8_2.x86_64
initramfs-4.18.0-193.19.1.el8_2.x86_64.img                             .vmlinuz-4.18.0-
193.19.1.el8_2.x86_64.hmac
initramfs-4.18.0-193.19.1.el8_2.x86_64kdump.img         vmlinuz-4.18.0-193.el8.x86_64
initramfs-4.18.0-193.el8.x86_64.img                                    .vmlinuz-4.18.0-
193.el8.x86_64.hmac
initramfs-4.18.0-193.el8.x86_64kdump.img                                vmlinuz-4.18.0-
240.1.1.el8_3.x86_64
initramfs-4.18.0-240.1.1.el8_3.x86_64.img                              .vmlinuz-4.18.0-
240.1.1.el8_3.x86_64.hmac
loader/
```

After setting the default kernel to the desirable version we need to reboot the server to load the Kernel.

```
[root@argo ~]# grubby --set-default /boot/vmlinuz-4.18.0-240.1.1.el8_3.x86_64
The    default    is    /boot/loader/entries/1694ddb46dce458baa75a96468630089-4.18.0-
240.1.1.el8_3.x86_64.conf    with    index    0    and    kernel    /boot/vmlinuz-4.18.0-
240.1.1.el8_3.x86_64
```

Compute Nodes Kernel update

We create a backup to the current node images. The images are located in: (`/opt/ohpc/admin/images/`). The next steps are similar to previous steps. First, we Install the kernel package using the following command:

```
[root@argo ~]# yum --installroot=$CHROOT update kernel-devel kernel-headers kernel
```

After installing latest Kernel, the command wwvnfs is used to encapsulate and compress this filesystem into a VNFS image which can be provisioned.

```
[root@argo ~]# wwvnfs --chroot $CHROOT
```

Once we finished compressing the filesystem, we need to reboot the node to load the new Kernel.

Since we use a separate image for the node argo-c10 we need to update the $CHROOT value and install the Kernel and compress the image. Also, we need to update the latest Nvidia packages for the node.

APPENDIX VI

High Speed Network Setup

This appendix describes the process/instructions to set up the 10Gbs network for internal communication between the compute nodes, this network is crucial to boost the cluster efficiency, so making sure that the network function properly is an important task. We choose to set up the network with static IP since the network connects 10 nodes only, so the set-up process is straightforward.

The IP address setup for each node is chosen as follows:

IP = 192.168.240.X (where X corresponded to the cluster numeric postfix)
For example, the node argo-c0 will have the IP 192.168.240.0

The network setup for each node is done as following:

The first setup is to create a cfg file with the network properties in the location `/etc/sysconfig/network-config/name.cfg`
The file name corresponds to the interface name[8].
The content of the file is the following:

```
DEVICE=eth2

BOOTPROTO=static

ONBOOT=yes

IPADDR=192.168.1.240

NETMASK=255.255.255.0
```

The variable DEVICE corresponds to the network interface, and IPADDR represent the interface IP address.
After setting up the network configuration file we use the **ifup** and **ifdwon** tools to turn ON or OFF the network.

```
[root@argo-c0 ~]# ifup eth2
Connection successfully activated (D-Bus active path:
/org/freedesktop/NetworkManager/ActiveConnection/24899)
```

The **ifconfig** tool can be used to verify if the network if up

```
[root@argo-c0 ~]# ifconfig
. . . .
eth2: flags=4163<UP,BROADCAST,RUNNING,MULTICAST>  mtu 1500
        inet 192.168.1.240  netmask 255.255.255.0  broadcast 192.168.1.255
        inet6 fe80::6d4:c4ff:fe62:f73a  prefixlen 64  scopeid 0x20<link>
        ether 04:d4:c4:62:f7:3a  txqueuelen 1000  (Ethernet)
        RX packets 1119207  bytes 365964363 (349.0 MiB)
        RX errors 0  dropped 0  overruns 0  frame 0
        TX packets 373330  bytes 60899833 (58.0 MiB)
        TX errors 0  dropped 0 overruns 0  carrier 0  collisions 0
. . . .
```

[8] the name my change after provisioning the node, use the MAC address

The next table shows the interface name the MAC address and the chosen IP address for each node.

Node	Interfaces	MAC address	IP address
argo-c0	eth2	04:d4:c4:62:f7:3a	192.168.1.240
argo-c1	eth2	04:d4:c4:62:f3:48	192.168.1.241
argo-c2	eth2	04:d9:f5:11:66:3c	192.168.1.242
argo-c3	eth2	04:d9:f5:13:8f:b9	192.168.1.243
argo-c4	eth2	04:d9:f5:13:8e:9a	192.168.1.244
argo-c5	eth2	04:d9:f5:13:8c:e4	192.168.1.245
argo-c6	eth2	04:d9:f5:13:8f:a4	192.168.1.246
argo-c7	eth2	04:d9:f5:13:8f:b4	192.168.1.247
argo-c8	eth4	04:d9:f5:13:8f:a2	192.168.1.248
argo-c9	eth2	04:d9:f5:11:68:70	192.168.1.249

APPENDIX VII

Integration Test Suite

This appendix details the installation and basic use of the integration test suite used to support OpenHPC releases. The test is used to confirm component builds are functional and inter-operable within the modular OpenHPC environment.
To install the test suite, provide by a standalone RPM:

```
[root@argo ~]# yum -y install test-suite-ohpc
```

The RPM installation creates a user named ohpc-test to house the test suite and provide an isolated environment for execution.
Most components can be tested individually, but a default configuration is setup to enable collective testing. To test an isolated component, use the configure option to disable all tests, then re-enable the desired test to run.
Example output is shown below

```
[root@argo ~]# su - ohpc-test
[ohpc-test@argo ~]$ cd tests
[ohpc-test@argo tests]$ ./configure --disable-all --enable-tbb --enable-ntp --enable-
admin --enable-cmake –enable-compilers
```

After configuring the test, the command **make check** is used to run the test suite, the command will print the test output, example output is shown below

```
[ohpc-test@argo tests]$ make check

Making check in time
make  check-TESTS
PASS: ntp
============================================================================
Testsuite summary for test-suite 2.0.0
============================================================================
# TOTAL: 1
# PASS:  1
# SKIP:  0
# XFAIL: 0
# FAIL:  0
# XPASS: 0
# ERROR: 0
============================================================================
make --no-print-directory check-TESTS
PASS: compilers/ohpc-tests/test_compiler_families
PASS: dev-tools/cmake/run
PASS: dev-tools/tbb/ohpc-tests/test_compiler_families
============================================================================
Testsuite summary for test-suite 2.0.0
============================================================================
# TOTAL: 3
# PASS:  3
# SKIP:  0
# XFAIL: 0
# FAIL:  0
# XPASS: 0
# ERROR: 0
============================================================================
```

APPENDIX VIII

Administrator Guide

This appendix provides a guide to setup the Argo cluster, it can be used as a guideline to maintain and upgrade the cluster when necessary. This Appendix contains general guidelines to assist the system administrator in the above-mentioned processes.

- **Configuration Backup**
  To assure a smooth transition it is advised to backup several files from the current working cluster as s reference.
  The following list indicates some of the files that can be useful to the administrator during the setup process.
  - **Bashrc,** the root bashrc file contains the recipe details for installing a cluster starting from bare-metal, such as the defined IP addresses and the hardware MAC addresses in order to support a controlled provisioning process. These values are necessarily unique to the hardware being used (location: /root/.bashrc).
  - **Hosts,** Prior to beginning the installation process the hosts file contains the host and the nodes IP and name, to ensure that the SMS hostname is resolvable locally (location: /etc/hosts).
  - **resolv.conf,** is used to access the remote repositories by hostname (and not IP addresses) to configure the system's Domain Name System (DNS) resolver (location: /etc/resolv.conf).
  - **objects,** all the elements that are involved in the monitoring process are defined in the object directory, hence backing up the directory can be useful as a point of reference when re-configuring Nagios (location: /etc/nagios/objects/).

- **Base OS Setup**
  CentOS8.2 distribution is installed on the head node using a bootable USB ISO image, following the distribution provided directions to install the Base Operating System (BOS) on the master host. It's advisable to set up the "public access" network1, to have the host connected to the internet during the installation. As for the installation destination, we combined the two identical 1TB hard drives, which through the operating system are connected to RAID1 3 technology, to provide data residencies to secure the Cluster data. After choosing the size of the partitions, we proceeded with the software selection process.

  The Hard disk storage is partitioned as follows:
  - root partition 256Gb
  - var partition 64Gb
  - boot partition 2Gb
  - data partition 256Gb
  - swap partition 16Gb

  For the base environment, the option **Workstation** was chosen, in addition to the **Development Tools**, **Security Tools**, and **Performance Tools** Add-Ons for the selected Environment.

- **Host Network Setup**

  The Host (SMS) requires at least two Ethernet interfaces with enp2s0 connected to the local data center network and enp0s31f6 used to provision and manage the cluster back-end (note that these interface names may vary depending on OS naming conventions). The first interface (enp2s0) will be configured during the OpenHPC installation process. The second (enp0s31f6) interface must be configured during the OS installation process as stated in the previous steps.

Interfaces	MAC address	IP address
enp0s31f6	50:9a:4c:21:d6:21	192.168.122.1
enp2s0	00:40:f4:eb:17:ea	88.197.43.249

The file ifcfg-enp2s0, contains the network configuration, this information is required to setup the network.

```
[root@argo ~]# cat /etc/sysconfig/network-scripts/ifcfg-enp2s0
TYPE="Ethernet"
PROXY_METHOD="none"
BROWSER_ONLY="no"
BOOTPROTO="static"
DEFROUTE="yes"
IPV4_FAILURE_FATAL="no"
IPADDR="88.197.43.249"
GATEWAY=88.197.43.129
IPV6INIT="no"
IPV6_AUTOCONF="yes"
IPV6_DEFROUTE="no"
IPV6_FAILURE_FATAL="no"
IPV6_ADDR_GEN_MODE="stable-privacy"
NAME="enp2s0"
UUID="a77fa9b8-bc8d-4884-8082-cce3e4f64991"
DEVICE="enp2s0"
ONBOOT="yes"
PREFIX="25"
DNS1="88.197.43.131"
IPV6_PRIVACY="no"
```

- **Cluster's Variables Preparation**
  The installation recipe details for installing a cluster use predefined bash variables, such as the defined IP addresses and the hardware MAC addresses in order to support a controlled provisioning process. These values are necessarily unique to the hardware being used. These variables are stored in the ROOT bashrc file.
  Note that the file contains two CHROOT variables since the node argo-c10 has a separate installation destination from the other nodes.

- **CUDA Software Installation (Host)**
  CUDA installation guide [23] provides the instructions to install and check the correct operation of the CUDA development tools. Selecting the appropriate CUDA version [24] depends on the GPU version available on the node.
  The CUDA Toolkit can be installed using the distribution-specific packages (RPM and Deb packages), It is recommended to use the distribution-specific packages, where possible.
  To install the appropriate repository meta-data, we must determine which distribution and release number are running. After enabling the repository, The CUDA development tools, the Dynamic Kernel Module Support, and CUDA drivers can be installed using the package manager (yum). After installing CUDA development tools we must add the environment variable and add the development tool's location for all users permanently, by creating a (**cuda.sh**) file in **/etc/profile.d** , Thus, permanently setting the environments variable for all users.

- **Add BeeGFS**
  To add optional support for mounting BeeGFS file systems, an additional external yum repository provided by the BeeGFS project must be configured. The ${sysmgmtd host} should point to the server running the BeeGFS Management Service. Starting the client service triggers a build of a kernel module, hence the kernel module development packages must be installed first.

- **Intel® Parallel Studio XE Software Installation**
  The Intel Parallel Studio software suite must be obtained and installed separately. Once installed locally, the OpenHPC compatible packages can be installed using the standard package manager.

- **Install OpenHPC Components**
  Chapter 3 in this document provides detailed instructions for installing the OpenHPC components and provisioning the compute nodes, please note that the provisioning of the nodes argo-c0 to argo-c9 is done separately from the node argo-c10, the latest is installed on a different location since we need to install Kernel Module Support, and CUDA drivers on the specific node.

- **Install OpenHPC Development Components**
  OpenHPC packages can be added to support a flexible HPC development environment including development tools, C/C++/FORTRAN compilers, MPI stacks, and a variety of 3rd party libraries. Sections 3.8 of this document provide instructions on the installed components.

- **Resource Manager Setup**
  After setting up the compute nodes, the Resource Manager setup is required to utilize the nodes. At first, we have to start up the necessary services to support resource management under OpenPBS. The next step is to add the compute nodes to the OpenPBS resource manager, to register each host and apply several key global configuration options, and to create two executions queue (argo-c10 will be added to a separate execution queue) lastly, we have to append the nodes to the newly created queues respectively.

References

1. Building a High-performance Computing Cluster Using FreeBSD, Brooks Davis, Michael AuYeung, Gary Green, Craig Lee, The Aerospace Corporation, El Segundo, CA
2. Cluster Computing with OpenHPC, Schulz, Karl W.; Baird, C. Reese; Brayford, David; Georgiou, Yiannis; Kurtzer, Gregory M.; Simmel, Derek; Sterling, Thomas; Sundararajan, Nirmala; Van Hensbergen, Eric
3. Best Practices for the Deployment and Management of Production HPC clusters
4. SOFTICE: Facilitating both Adoption of Linux Undergraduate Operating Systems Laboratories and Students' Immersion in Kernel Code
5. Slicing and Dicing OpenHPC Infrastructure: Virtual Clusters in OpenStack
6. Low-Cost High-Performance Computing Via Consumer GPUs
7. Customized HPC Cluster Software Stack on QCT Developer Cloud
8. HPC Cluster Argo: Συναρμολόγηση, Εγκατάσταση, Συντήρηση, Λειτουργία με το OpenHPC Ανδρέας Ν. Χαραλάμπους
9. OpenHPC(v2.0) Cluster Building Recipes, CentOS8.2 Base OS Warewulf/OpenPBS Edition for Linux* (x86 64) https://github.com/openhpc/ohpc/releases/download/v2.0.GA/Install_guide-CentOS8-Warewulf-OpenPBS-2.0-x86_64.pdf
10. OpenHPC (v1.3.9) Cluster Building Recipes CentOS7.7 Base OS Warewulf/PBS Professional Edition for Linux* (x86 64) https://github.com/openhpc/ohpc/releases/download/v1.3.9.GA/Install_guide-CentOS7-Warewulf-PBSPro-1.3.9-x86_64.pdf
11. Virtual Node File System Management https://warewulf.lbl.gov/subprojects_components_plugins/vnfs.html
12. Linux Foundation. https://www.linuxfoundation.org.
13. Beowulf: A Parallel Workstation for Scientific Computation
14. *Argo https://en.wikipedia.org/wiki/Argo*
15. *CentOS Wiki https://wiki.centos.org/*
16. PowerEdge 1950 Server Product Details https://www.dell.com/en-us/work/shop/povw/poweredge-1950
17. https://github.com/tacc/lmod. Lmod: An Environment Module System based on Lua.
18. OpenHPC Community. OpenHPC Project. https://openhpc.community/, 2019.
19. 16-Port Gigabit Rackmount Switch https://www.tp-link.com/us/business-networking/unmanaged-switch/tl-sg1016/
20. *General Warewulf Overview https://warewulf.lbl.gov/*
21. ohpc releases https://github.com/openhpc/ohpc/releases
22. Virtual Node File System Management https://warewulf.lbl.gov/subprojects_components_plugins/vnfs.html
23. CUDA Toolkit Documentation: NVIDIA CUDA, Installation Guide for Linux https://docs.nvidia.com/cuda/cuda-installation-guide-linux/index.html
24. CUDA Toolkit 11.4 Update 2 Downloads https://developer.nvidia.com/cuda-downloads?target_os=Linux&target_arch=ppc64le&target_distro=RHEL&target_version=7&target_type=rpmnetwork
25. lspci(8) - Linux man page https://linux.die.net/man/8/lspci
26. PBS Professional Documentation https://www.altair.com/pbs-works-documentation/
27. PBS Administrator's Guide https://www.altair.com/pdfs/pbsworks/PBSAdminGuide2020.1.pdf
28. PBS User Guide https://www.altair.com/pdfs/pbsworks/PBSUserGuide2020.1.pdf
29. Chapter 20. System Initialization 20.5. Run Levels. Linux Standard Base Core Specification 4.1. 2011. Retrieved 2011-04-21.
30. NRPE Documentation Copyright (c) 1999-2017 Ethan Galstad https://assets.nagios.com/downloads/nagioscore/docs/nrpe/NRPE.pdf
31. PBS Professional 19.2.3 Big Book Guide, Altair
32. Learning Nagios, Third Edition, Wojciech Kocjan Piotr Beltowski
33. Kunpeng BoostKit for HPC Installation Guide, HUAWEI TECHNOLOGIES CO., LTD
34. HPC, fakecineaste ,2018, http://fakecineaste.blogspot.com/2018/02/hpc.html
35. Beowulf Cluster Computing with Linux, Thomas Sterling
36. Monitoring with Ganglia, Matt Massie, Bernard Li, Brad Nicholes, and Vladimir Vuksan
37. Slurm Workload Manager - Quick Start User Guide, https://slurm.schedmd.com/quickstart.html
38. Operating Systems for Supercomputers and High-Performance Computing
39. High Performance Computing Modern Systems and Practices, Thomas Sterling, Matthew Anderson, Maciej Brodowicz,
40. A method of evaluation of high-performance computing batch schedulers, Jeremy Stephen Futral, University of North Florida.
41. High-performance computing using a reconfigurable accelerator (add this for future expantion :-) FPGA)
42. Essential Cluster OS Commands, Computer Engineering Group, http://gec.di.uminho.pt/

43. Chapter 17. Cluster Computing with Linux, http://etutorials.org/Linux+systems/cluster+computing+with+linux/
44. PBS Professional® 13.0 Installation and Upgrade Guide
45. Introduction to Nagios, Wojciech Kocjan, Piotr Beltowski, 2016
46. nrpe(8) [centos man page], https://www.unix.com/man-page/centos/8/nrpe/
47. How to install and enable EPEL repository on a CentOS/RHEL 7, https://www.cyberciti.biz/faq/installing-rhel-epel-repo-on-centos-redhat-7-x/
48. Linux Clusters Institute: Node Health Check (NHC), Jeffrey Lang, Sr. Systems Administrator, University Of Wyoming
49. 8.4.7. Working with Yum Cache, https://access.redhat.com/documentation/en-us/red_hat_enterprise_linux/6/html/deployment_guide/sec-working_with_yum_cache
50. Nouveau: Accelerated Open Source driver for nVidia cards, https://nouveau.freedesktop.org/
51. Institute of Accelerating Systems & Applications – IASA, https://www.iasa.gr/index.php