SAND2017-8910C

Enabling Diverse Software Stacks on Supercomputers using High Performance Virtual Clusters

Andrew J. Younge, Kevin Pedretti, Ryan E. Grant, Brian L. Gaines, Ron Brightwell
Center for Computing Research
Sandia National Laboratories*
P.O. Box 5800, MS-1319
Albuquerque, NM 87185-1110
Email: {ajyoung, ktpedre, regrant, bgaines, rbbrigh}@sandia.gov

***Abstract*—While large-scale simulations have been the hallmark of the High Performance Computing (HPC) community for decades, Large Scale Data Analytics (LSDA) workloads are gaining attention within the scientific community not only as a processing component to large HPC simulations, but also as standalone scientific tools for knowledge discovery. With the path towards Exascale, new HPC runtime systems are also emerging in a way that differs from classical distributed computing models. However, system software for such capabilities on the latest extreme-scale DOE supercomputing needs to be enhanced to more appropriately support these types of emerging software ecosystems.**

In this paper, we propose the use of Virtual Clusters on advanced supercomputing resources to enable systems to support not only HPC workloads, but also emerging big data stacks. Specifically, we have deployed the KVM hypervisor within Cray's Compute Node Linux on a XC-series supercomputer testbed. We also use libvirt and QEMU to manage and provision VMs directly on compute nodes, leveraging Ethernet-over-Aries network emulation. To our knowledge, this is the first known use of KVM on a true MPP supercomputer. We investigate the overhead our solution using HPC benchmarks, both evaluating single-node performance as well as weak scaling of a 32-node virtual cluster. Overall, we find single node performance of our solution using KVM on a Cray is very efficient with near-native performance. However overhead increases by up to 20% as virtual cluster size increases, due to limitations of the Ethernet-over-Aries bridged network. Furthermore, we deploy Apache Spark with large data analysis workloads in a Virtual Cluster, effectively demonstrating how diverse software ecosystems can be supported by High Performance Virtual Clusters.

I. Introduction

Currently, we are at the forefront of a convergence within scientific computing between High Performance Computing (HPC) and Large Scale Data Analytics (LSDA) [1], [2]. This amalgamation of differing viewpoints in distributed systems looks to force the combination of performance characteristics of HPC's pursuit towards Exascale with data and programmer oriented concurrency models found in Big Data analytics platforms. Capitalizing upon the community's existing intellectual investments in advanced supercomputing systems and leveraging the economies of scale in hardware infrastructure could benefit more computational methods beyond what is possible as disjoint environments. Current software efforts in each area have become specialized with the gap growing rapidly, making concurrent ecosystem support within a single architectural system increasingly intractable.

Much of the convergence effort has been focused on applications and platform services. Specifically, significant work towards convergent applications has been outlined with the Big Data Ogres [3] and the corresponding HPC-ABDS model [4]. This convergence can also be seen with efforts in bringing interconnect advances from HPC to data analytics platforms, such as with InfiniBand and MapReduce [5]. However, much of the underlying hardware and OS environments between the latest MPP supercomputing resources and the cloud-enabled runtimes of big data applications are still something to be reconciled. We believe virtualization can help fill this gap.

In this, we postulate embracing this software diversity on advanced supercomputing platforms through the use of High Performance Virtual Clusters. The notion of a virtual cluster describes a self-contained entity for a cluster system that is specific to a user's software stack. They can be structured in any way that physical clusters currently can to support distributed systems, complete with a head node, compute nodes, storage servers, user gateways, or even P2P services, to name a few components. Virtual clusters are separated from the underlying computing infrastructure by running atop a virtualization layer, in our case one that is tuned for improved performance. This enables disjoint software ecosystems to provision and operate independent software stacks deployed concurrently on the same hardware.

With a proper design, current HPC system software stacks should continue to operate in the same environment as before on the same hardware, yet we also enable the ability for emerging analytics and visualization workloads such as pieces of the Apache Big Data Stack [6], among others, to deploy custom software specific to application needs rather than adapting to site-specific software. Furthermore, such virtualized clusters enable new methods for non-standard workflow composition,

*Sandia National Laboratories is a multimission laboratory managed and operated by National Technology and Engineering Solutions of Sandia, LLC., a wholly owned subsidiary of Honeywell International, Inc., for the U.S. Department of Energys National Nuclear Security Administration under contract DE-NA0003525.

such as the in-situ coupling of parallel MPI simulations with emerging data analytics and visualization tools for real-time experimental control, either across or within clusters. Virtual clusters can conceivably enable the ability to uniquely couple both simulations and analytics for enhanced computational abilities, both intra and inter cluster, such as with emerging Asynchronous Many-Task runtimes.

To build High Performance Virtual Clusters, we have chosen to utilize the KVM hypervisor for running Virtual Machines (VMs) on Cray XC-series supercomputing resources. While in this paper we use a relatively small testbed for initial prototyping and development, our selection of running VMs on a Cray resource is a specific one. Cray Inc. supercomputers currently account for 3 of the top 10 systems as of the Nov 2016 TOP500 supercomputing list [7] and, when compared to standard commodity clusters or cloud infrastructure, their architectural design is highly optimized for extreme-scale parallel workloads. If successful, we hope a virtual cluster prototype could scale up to large-scale deployments operating today, or even towards future supercomputing architectures.

Effectively, we demonstrate that KVM virtual machines can be run on a Cray supercomputer, which to our knowledge is a novel use case for KVM in a unique supercomputing environment such as a Cray. Second, we confirm what previous research [8], [9] has already noted, in that when vitualization is properly designed and configured, very low overhead and good performance can be provided to HPC applications. This is an important notion for systems that are looking to improve flexibility of supercomputing software without sacrificing significant performance or disturbing current workloads. Third, we demonstrate how to run Apache Spark in a completely unmodified format in a virtual cluster deployment on a Cray testbed. While this is possible with different storage back-ends or additional reconfiguration [10], we hope this can collectively help drive the inclusion of more large data analytics stacks in supercomputing.

This manuscript is constructed as follows. First, we evaluate current virtualization efforts in the related research section, specifically focusing on HPC and big data requirements. Next, we describe the design and implementation of virtual clusters on a Cray XC30 supercomputer, using the KVM hypervisor, and detail the technical challenges of running VMs on a Cray efficiently. Then, we use the HPCC benchmark suite and HPCG to evaluate performance, both from a single-node perspective, as well as with multi-node experiments up to 768 cores and 32 nodes. We also detail the running of Apache Spark within a new virtual cluster, and its relative ease of deployment. Finally, we discuss how our prototype can best move forward towards supporting HPC and Big Data convergence, and what technological advances are necessary for a more successful future convergent platform.

II. Related Research

Cluster computing has become one of the core tools in distributed systems for use in parallel computation [11]. Clusters have manifested themselves in many different ways, ranging from Beowulf clusters [12], which run Linux on commodity PC hardware, to some of the TOP500 supercomputing systems today. Virtual clusters represent the growing need of users to organize computational resources in an environment specific to their tasks at hand effectively, instead of sharing a common architecture across many users. With the advent of modern virtualization, virtual clusters are deployed across a set of Virtual Machines (VMs) in order to gain relative isolation and flexibility between disjoint clusters while still sharing the same underlying compute infrastructure. Virtual clusters, or a set of multiple cluster computing deployments on a single, larger physical cluster infrastructure, often have the following properties and attributes [13]:

- Resources allocation based on VM units
- Clusters built of many VMs together, or by re-provisioning physical nodes
- OS, system software, and applications dictated by users, not administrators or vendors
- Leverage local infrastructure management tools to provide a middleware solution for virtual clusters
  - Implementations could be a cloud IaaS such as OpenStack
  - Others can include a batch queue system such as PBS as found in clusters
- User experience based on virtual cluster management, not single VM management
- Consolidate server functionality on a smaller resource set using multiple VMs
- Fault tolerance through VM migration mechanisms
- Utilize dynamic scaling through the addition or deletion of VMs from the virtual cluster during runtime
- Connection to back-end storage solutions to provide virtual persistent storage

Given these abstract properties, virtual clusters can take many forms. Initial conceptions included the very notion of virtual clusters as an on-demand computing system [14], followed by using virtual clusters within the Grid computing field [15], [16] to satisfy differing Virtual Organization requirements [17]. While this methodology worked relatively well throughout High Throughput Computing (HTC) and such pleasingly parallel workloads were also migrated towards Cloud infrastructure as it became available. Today, some of the largest HTC workloads are run on large-scale cloud systems in a manner that resembles virtual clusters, including the LHC CMS and ATLAS experiments [18].

Efforts have also been underway to support more complex HPC workloads using virtual clusters. The Hobbes project [19] seeks to create an OS & Runtime framework for extreme-scale systems. In this, the Palacios VMM [20] demonstrated running HPC workloads up to thousands of nodes with minimal overhead [8], using the Catamount OS [21] specifically tuned for HPC applications. Within the FutureGrid project, initial investigation into the viability of virtualization was investigated [22], and small-scale virtual clusters were constructed focused on the OpenStack IaaS, which leveraged both NVIDIA GPUs and an InfiniBand interconnect using KVM [9] for HPC molecular dynamics simulations. The Chameleon Cloud [23], a NSF testbed project and follow-on to FutureGrid, also looks

to provide the ability for users to create virtual clusters through a catalog of appliances which can be deployed and configured for particular software stacks, focusing on leveraging more HPC-centric hardware.

A. Hypervisors and Containers

There are numerous efforts to provide virtualization on commodity x86 systems. This includes virtualization in the form of binary translation, hardware-assisted virtualization, and most recently in OS-level virtualization, also more commonly known as containers. Even within the spectrum of full virtualization, there exists multiple types of hypervisors, each with advantages and caveats. As good design requires a proper selection of the right level of abstraction for providing virtual clusters atop supercomputing resources, Figure 1 briefly details 3 major types of virtualization most often found today.

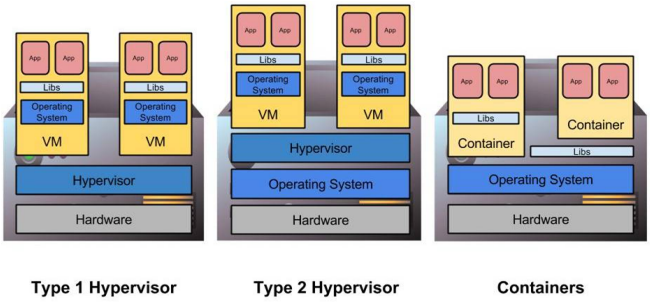


Fig. 1: Hypervisors and Containers

With a Type 1 virtualization system, the hypervisor or Virtual Machine Monitor (VMM) sits directly on the bare-metal hardware, below the OS. These native hypervisors shepherd direct control of the underlying hardware, and can be controlled and operated through the use of a single privileged VM. One example of a Type 1 hypervisor is Xen[24], which uses the Xen VMM to virtualize a privileged Linux OS, called Dom0, that then creates and manages other user DomU instances. This is in contrast to Type 2 hypervisors, which utilize a different, and sometimes more convoluted design. With a Type 2 hypervisor, there is a host OS that sits directly atop hardware as a normal OS would. However, the OS itself can abstract its own hardware, usually through extended kernel support or modules, and manages VMs as OS processes. This effectively allows for guest VMs to run atop existing OS infrastructure with just kernel modules rather than as a complete redesign of the OS system. The prime example of a Type 2 hypervisor with special relevancy to this manuscript it the Kernel Virtual Machine, or KVM hypervisor as part of Linux. Type 1 and Type 2 hypervisors are distinct from OS level virtualization, also known as containerization. With containers, there is a single OS kernel; however, instead of direct hardware abstraction, the OS kernel is used to simultaneously run multiple user-space instances in a jailed-root environment, often with separate namespaces and additional root-level restrictions on underlying resources. These environments may look and feel like a separate machine, but in fact are simply sharing a single OS kernel. The most current example is Docker containers [25]. While this provides near-native performance and ease of use, it often lacks flexibility and essentially binds resources to a specific kernel, often that kernel being Linux.

Given the levels of abstraction and various hypervisors available today, KVM, which is a Type 2 hypervisor, looked to be an appealing choice for constructing flexible virtual clusters atop extreme-scale supercomputing resources. KVM is well supported throughout industry with pervasiveness throughout current Cloud infrastructure, is well tested, and has support for multiple ISA architectures, including x86, ARM, and POWER architectures. This allows for our design, if successful, to be extended to other emerging system architectures in the future. The selection of KVM as a Type 2 hypervisor is also not a coincidence, as we find this a relatively unobtrusive addition to existing vendor HPC software stacks, as it is can exist as just a Linux kernel module and user-level support libraries, without otherwise disturbing the native software stack. This decision is also reinforced by the selection of the Palacios VMM for HPC workloads [8], which also demonstrated success in providing advanced virtualization capabilities to HPC. Furthermore, the selection of a Type 2 hypervisor in no way precludes using containerization; in fact we surmise that such a hypervisor could be used with and extend containerization options, such as Shifter [26], to include additional levels of security and resource isolation beyond what current container solutions provide.

B. Data Analytics on Cloud Infrastructure

It is expected that new LSDA and big data efforts will continue to move in a direction towards convergence with HPC in the context of software platform design [27]. We expect this to be true if virtualization can make HPC hardware that is traditionally prohibitive in such areas, such as novel ISA architectures, accelerators, and high-speed interconnects, readily available to big data platforms. This can effectively ease portability issues at the OS and runtime level. As the deployment of big data applications and platform services on virtualized infrastructure is well defined and studied [28] in the cloud computing spectrum, however we hope such efforts can be extended to provide more efficient resource utilization and direct in-situ analytics of HPC simulations. As such, research regarding virtualization can also play a part in bringing advanced hardware and performance-focused considerations to Big Data applications, effectively cross-cutting the convergence with HPC. Recent efforts have taken place utilizing collectives found in HPC applications within big data frameworks [29], leveraging high-speed, low-latency interconnects directly in Map Reduce frameworks like Hadoop [30], and investigating areas where performance-centric lessons learned can be leveraged within big data stacks [31]. These endeavors are collectively pushing forward the notion of cross-cutting convergence within analytics platform services themselves.

III. Design of High Performance Virtual Clusters

Given the expansion in emerging distributed computational paradigms to support current and future scientific computing challenges at scale, HPC resources need to expand system

software capabilities to adapt. This may be especially true for high-end supercomputing systems and MPP architectures, which have historically done well supporting MPI based workloads at high efficiency. Due to the large influx of virtualization along with Cloud infrastructure throughout industry, current CPU architectures including x86 and ARM are now capable of supporting diverse software stacks through virtual machines. However, bringing virtualization to such specialized supercomputing architectures presents a number of technical challenges.

A. Volta: a Cray XC30 testbed

Throughout this manuscript we illustrate the construction of a prototype mechanism for supporting High Performance Virtual Clusters using the Volta supercomputing testbed system. Volta is a Cray XC30 system deployed as part of the Advanced Systems Technology Testbeds project at Sandia National Laboratories through the NNSA's Advanced Simulation and Computing (ASC) program. Volta includes 56 compute nodes packaged in a single enclosure, with each node consisting of two Intel "Ivy Bridge" E5-2695v2 processors (24 cores total), 64 GB of memory, and a Cray Aries network interface. Compute nodes do not contain local storage. Shared file system support is provided by NFS I/O servers projected to compute nodes via Cray's proprietary DVS storage infrastructure. Each compute node runs an instance of Cray's Compute Node Linux OS [32] (ver. 5.2.UP04), which is based on SUSE Linux 11 with a Cray-customized 3.0.101 Linux kernel.

Volta is not heavily used and it is possible for researchers to obtain root, making it a valuable resource for system software research. This enabled more rapid prototyping of virtual clusters on Cray systems than would have been possible using a more production-oriented system. The infrastructure and techniques we have developed on Volta are also applicable to other Cray systems such as the Trinity Cray XC40 supercomputer [33], as Cray XC system software does not have hardware virtualization enabled by default.

B. KVM

As stated in the previous section, the KVM hypervisor was chosen in these experiments for a number of reasons. First, KVM is a fully supported Linux kernel module, and as Linux is the most common OS on supercomputing systems today, makes for a good fit. Furthermore, a Type 2 hypervisor is naturally a better choice for providing virtualization on such a supercomputing system, as it is minimally intrusive to the current vendor stack, and does not create considerable performance impact on host running applications. Finally, virtualization, unlike containers, has the ability to run completely separate OS and runtime systems on the guest, such as Kitten [20] or other novel OSes if desired in the future. Given the advances in reducing OS noise through the use of hybrid and lightweight co-kernels [34], [35], this could also act as a valuable research tool for future OS and Runtime research.

The default Cray Compute Node Linux kernel does not provide support for KVM, so we needed to build a custom kernel image. This was not straightforward because enabling KVM modified several Linux kernel data structure layouts, causing Cray's binary-only kernel modules to break. Through trial and error, we identified the critical data structures and modified them to place KVM-related fields at the end of structures rather than in the middle, thus preserving the field offsets assumed by Cray's binary-only modules. We are hopeful this type of workaround will not be needed in the future, but at present it is even for recent Cray software releases.

While KVM itself is a foundational piece for providing guest VMs on a Cray host compute node, it alone is not enough. The QEMU machine emulator is necessary in conjunction with KVM to provide user-space device emulation. This includes console access to the guest and various peripheral emulation including disk and I/O access. In our prototype we utilized QEMU 2.7.90, the latest available during time of development. While QEMU/KVM alone can (and will) boot VMs, the interface to do so is often lacking flexibility and customization often required by more advanced VM management services. In order to support easy deployment on a Cray, as well as provide a direct interface for larger orchestration and VM management tools such as OpenStack [36] in the future, version 3.1.0 of the Libvirt virtualization API was loaded atop QEMU/KVM. This allows for users to specify easy and sharable VM configurations expressed in XML format, as well as enable special tuning mechanisms for improved guest performance, which is detailed in the next section.

C. Guest Performance Tuning

As performance is a first class function for HPC applications, running VMs on advanced architectures also needs to be performance-focused as well. While virtualization, like any software abstraction, will introduce some overhead beyond running natively, there presents significant opportunities in tuning with KVM to minimize overhead and provide the best performance possible. An efficient virtual machine configuration is specified and described below.

The first aspect to focus on was, predictably, the virtualization of the CPU itself. While QEMU/KVM can handle virtualization of the CPU automatically towards a default set that ensures comparability on most VT-x enabled x86 CPUs, doing so has notable performance implications, including incorrect cache sizes and unimplemented CPU instructions sets. As such, matching the virtual CPU configuration to the physical CPU set becomes crucial. While one can use the `host` mode in libvirt, such functionality was not available due to the age of the KVM code attached to the 3.0 Linux kernel within Cray CNL. As such, specifying the IvyBridge architecture of Volta's CPUs, as well as a matching socket and thread count was necessary as part of the libvirt configuration to provide the necessary instruction sets for good performance, including the AVX vector unit within Intel IvyBridge CPUs.

However, it is necessary to go further than just matching CPU architecture and specifying equal core counts. This is because by default KVM will let Linux process scheduler manage which vCPUs execute on which physical cores. This can lead to inefficiencies as multiple vCPUs could execute on

a single CPU, effectively degrading performance. This could also happen with hyperthreading, whereby two logical cores are running different VCPUs yet still end up contending for a single physical core on the host. Using libvirt's <cputune> feature allows for a direct pinning of a VCPU core to a physical core to alleviate this issue by directly specifying each VCPU pinned to a cpuset of one. Furthermore, to ensure time is kept accurately within guests, the kvm-clock clock source is used. This paravirtualized driver allows for the guest to gain access to host time measurements through a memory page which is updated via a MSR, and adjust for small latencies between the read time and actual time with the guest's own constant Time Stamp Counter (TSC), yielding accurate time keeping within the guest.

```
<memoryBacking>
  <hugepages>
    <page size="2" unit="M" nodeset="0"/>
    <page size="2" unit="M" nodeset="1"/>
  </hugepages>
  <nosharepages/>
</memoryBacking>
<cpu match='exact'>
  <model>IvyBridge</model>
  <topology sockets='2' cores='12'
   threads='1'/>
  <vendor>Intel</vendor>
  <numa>
    <cell id='0' cpus='0-11' memory='30'
     unit='GiB'/>
    <cell id='1' cpus='12-23' memory='30'
     unit='GiB'/>
  </numa>
</cpu>
<numatune>
  <memory mode='strict' nodeset='0-1'/>
  <memnode cellid="0" mode="strict"
   nodeset="0"/>
  <memnode cellid="1" mode="strict"
   nodeset="1"/>
</numatune>
<vcpu>24</vcpu>
<cputune>
  <vcpupin vcpu='0' cpuset='0'/>
  <vcpupin vcpu='1' cpuset='1'/>
  ...
  <vcpupin vcpu='23' cpuset='23'/>
</cputune>
```

Another key observation is that when the guest is rendered properly to match the host, there are considerable performance improvements in memory organization to be realized. This effort is split into two aspects, guest Non Uniform Memory Access (NUMA) configuration and the use of Hugepages. Specifying a NUMA topology within the guest enables the corresponding VCPUs efficient access to local physical memory, which can help avoid unnecessary slowdowns by memory accesses across the Intel QPI between multiple CPU sockets. This specification requires two steps within Libvirt. The <numatune> section is used to create a NUMA topology of 2 memory cells (matching the dual socket IvyBridge architecture), along with the <cell> bindings in Libvirt to match memory cells with CPU sets, as well as specify each cell's size.

Backing an entire guest memory address space in hugepages can have a significant performance improvement for memory-bound applications. This is because with modern x86 virtualization, memory faults are handled predominately in hardware through a Translation Lookaside Buffer (TLB), and in a VM requires up to 6 times more memory accesses due to the design of nested page tables [37]. While this has shown to be more efficient than using shadow page tables, it is still costly for applications that require a large TLB reach. With 2MB hugepages (instead of the default 4KB paging), two things happen. First, the page miss cost decreases from 24 to 15 memory accesses when used in conjunction with Transparent Hugepages (THP) in the guest, and the total number of page faults decreases due to the additional amount of memory handled within the TLB. Hence, hugepages enable memory intensive applications to avoid spending time walking the page table entries, not only in the decreased number of TLB misses, but through the decreased cost for each miss.

Within Libvirt, providing 2MB paging is relatively straightforward as with vCPU pinning and NUMA configuration. Specifying two <hugepages> entries within <memorybacking> in Libvirt and matching the nodeset to that of the previously detailed NUMA cell will back the entire guest machine with 2MB hugepages. With modern commodity Linux host OSes, this can be done the OS itself using THP or libhugetlbfs. However, due to Cray's particularity with CNL and the older 3.0 Linux Kernel, using THP in the host CNL is not an option. Instead, we leverage Cray's hugepage application support directly with KVM. This entails setting Cray's HUGETLB_DEFAULT_PAGE_SIZE=2M and restricting usage only to the qemu-system-x86_64 application. Furthermore, it was found that pre-allocation of these hugepages by setting the minimum pool size to the total VM memory amount is critical to performance, as it appropriately balances contiguous memory allocation across both NUMA domains, again reducing QPI traversal within the VM. In a standard Linux guest, THP can be used to provide hugepages support without application modification, or standard libhugetlbfs is also available.

With both the CPU and memory configurations tuned for good performance, the focus next shifts on peripherals, which too can have a considerable impact on application performance. For disk usage, the Virtio paravirtualized driver is used with RAW disk images. While this requires paravirtualized drivers in the guest, Linux kernels greater than 2.6.25 provide this support and performance improves drastically. Previous studies have shown Virtio disk usage to be superior compared to alternatives with IDE or SATA emulation [38].

A critical piece of hardware is the network interconnect, which is often a major differentiating factor between commodity clusters and MPPs. On the Cray XC30, the interconnect is implemented by the Cray Aries network interface and router, which was not designed or intended to support virtual machines. While the Aries network provides considerable bandwidth and latency advantages over InfiniBand or Ethernet, it does not support Single Root I/O Virtualization (SR-IOV) as often found with newer Mellanox-based interconnects. Using direct PCI passthrough of the Aries NIC, like initial InfiniBand

efforts [39], is also not possible as it would deprive the Host CNL OS from any network. Furthermore, the Cray device drivers necessary are not available within a commodity guest OS.

With the current generation of the Aries interconnect, our only feasible option was to utilize the Ethernet-over-Aries interconnect emulation. The Aries interconnect is capable of L2 Ethernet emulation for TCP/IP connectivity between compute nodes, and as such the Ethernet device can be bridged to provide Ethernet connectivity within the guest VM. However, there are a few extra steps that were necessary to make the network both functional and performant. First, the assigned MAC address needs to match the last three octets of the host Ethernet device in order for the Aries network to route packets. Second, static ARP entries and static IP addresses must be manually assigned with each guest for all potential interconnected nodes within a virtual cluster. Third, the MTU should be set to 65520 bytes, the maximum possible MTU size, which matches the host Ethernet-over-Aries configuration and provides significant performance improvements over a standard MTU of 1500.

With the bridged Ethernet-over-Aries interconnect, there is variance in performance, depending on usage. When using the Ethernet network across hosts, point-to-point bandwidth measures around 30 Gbs using the iperf tool [40]. Between two VMs on the same chassis or between a VM and any compute host, performance dips to 18 Gbs. While this is a large 40% overhead in network bandwidth, it still provides an Ethernet network that is 60% faster than the current state-of-practice 10 Gbs Ethernet that found in commodity clusters and HPC-tuned clouds. This impact in performance is due to the buffering of Ethernet frames within the host OS. While Ethernet will generally cause performance degradation for large HPC applications at scale compared to native Cray Aries interconnect, as seen in Section IV, it conversely offers increased bandwidth beyond public large cloud offering for any Ethernet-based distributed frameworks with 10Gb Ethernet. It is our hope this solution will be improved upon soon with the use of SR-IOV, custom paravirtualized drivers, or other hardware-based virtualization techniques in future HPC interconnects.

IV. Experimental Results

With the methodology to create performant VMs on Volta in hand, High Performance Virtual Clusters can be created on Volta. This section describes the process of creating two virtual clusters, as well as the performance compared to native Cray performance. This includes not only from a single-node perspective, but also using weak scaling experiments up to 32 nodes and 768 cores. All experimental results were run 3 times with each data points mean calculated and reported in the following datasets. Unless specified otherwise, the coefficient of variance for all results was at or below 3%.

A. Virtual Cluster Configurations

In order to evaluate the performance of our virtual clusters, it was most appropriate to first build an HPC oriented virtual cluster. While HPC applications themselves are likely to be fastest running natively on the Cray, we can use HPC-based benchmarks as tools to effectively measure performance of our virtual cluster prototype solution. The assumption is that if HPC benchmarks perform well across our virtual machines, this will also translate to other workloads better suited for deployment on virtual clusters themselves. This is demonstrated with the use of a second virtual cluster configuration using the Apache Spark platform running the TeraSort application.

All VMs are configured with a CentOS 7 Linux image running the stock 3.11 kernel with the latest security patches. From here, the HPC images were loaded with the latest Intel 2017 parallel studio cluster suite, which includes the ICC and IFORT compilers as well as the latest MKL libraries. The MPICH 3.2 MPI library was installed for MPI communication. While OpenMPI and the Intel MPI libraries were also available for use, MPICH was used due to its close comparability with Cray's MPI libraries, effectively looking for as close to apples-to-apples comparison as possible for MPI libraries. However, the ability to allow users to deploy the latest software within each virtual cluster can have profound impacts, as seen in the performance comparison within this section.

The HPCC and HPCG benchmark workloads were built and run in the virtual cluster environment. The HPCC benchmark suite [41] provides a well-rounded perspective of the performance of HPC hardware. It includes key benchmarks such as High Performance Linpack (HPL), DGEMM, STREAM, PTRANS, RandomAccess, FFT, and PingPong, many of which are detailed herein. However, these benchmarks, with HPL in particular, are relatively synthetic and may not directly relate to real-world HPC applications. As such, we also chose to deploy the High Performance Conjugate Gradient (HPCG) benchmark [42]. Between HPL and HPCG, two "bookends" of HPC are realized to evaluate parallel application performance and scalability across our virtual cluster.

For the native environment, HPCC (with HPL) and HPCG were built using Cray's standard programming environment. Specifically, the Cray-Intel programming environment was used with Intel compiler 16.0.1. Cray's 2MB hugepage support was enabled, matching the configuration used in the guest. By default, the libSci math library is utilized for matrix calculations throughout various HPCC benchmarks. However we also put forth the effort of compiling a version of HPCC with the Intel MKL library for reasons detailed in the single-node experiments section herein.

For our Apache Spark virtual cluster, the same CentOS 7 base image from the HPC virtual cluster was used. Oracle's Java JDK 8 was subsequently installed, along with Apache Maven and other support tools. Apache Spark [43] version 2.1.0 was installed in standalone mode. Spark is one of the leading Big Data Stack platforms within the Apache Foundation, allowing for Map-Reduce based applications to run both in memory and on disk storage back-ends for increased efficiency. Spark standalone mode was selected for the virtual cluster with a head node designated to function both as an NFS server and the Spark master node and application client. Standalone mode was specifically chosen to aid comparison between a Cray deployment. However, it would be possible to

build a virtual cluster using the more common HDFS storage system instead. While such deployments in public clouds often leverage node-local storage, there are a few options for full scale deployments on a Cray, such as using Burst Buffers as investigated previously [10]. This usage model would provide increased compatibility with existing Big Data Stack services (such as Hadoop) often found on public cloud infrastructure. This potential deployment on a Cray using virtualization further illustrates the power and flexibility provided such a virtual cluster architecture.

B. Single-node Performance

Utilizing our HPC-based virtual cluster, we first look to evaluate single-node performance. Using micro-benchmarks and tuned synthetic workloads on a single node helps identify areas of overhead within a node, especially those caused by CPU configuration, memory systems, and VM entry/exit overheads. As such, we evaluate benchmarks from the HPCC benchmark suite, which is commonly utilized within HPC systems to evaluate and accept new machine deployments. All results were also run on the Volta Cray testbed in a native Cray configuration. However, a strict apples-to-apples comparison is not entirely possible, due to the differing software stacks. While efforts were made to keep various libraries as similar as possible, by default an older Intel compiler suite and Cray's LibSci were used to compile the HPCC suite, compared to the latest Intel compiler and MKL libraries within VMs.

Turning first to Figure 2, direct CPU floating point performance is measured using the DGEMM and FFT benchmarks. Here, we immediately notice that KVM's performance is very good. In fact, in some cases, KVM *outperforms* the native solution running the default Cray software stack, specifically with DGEMM and MPIFFT. Initially this confused us, as it is very unlikely that a VM will run faster than Native mode. Upon further investigation, it became apparent that the default Cray configuration using LibSci math libraries results in a significant decrease in performance compared to Intel's MKL math library. Recompiling with MKL support on Volta using an older Intel MKL library (11.3.1, the latest available on Volta), we see that native performance improves to at or better than the KVM levels in most cases. However, because within the VM we use the latest 2017 Update 1 MKL, we are able to gain even further performance benefits. This section of the code is unlikely to generate major VM entries or exits which would hamper guest VM performance. The end result is that we achieve near-native performance for KVM, with some cases slight performance improvements due to leveraging the latest system software and libraries within a VM. Single-node HPL tests reveal a similar trend with KVM, with a 9% and 2% performance increase with KVM over the default Cray and Cray+MKL, respectively. We further speculate that Intel has refined MKL recently to provide additional performance improvements. This effect illustrates the potential power of virtual clusters for those looking to evaluate dev-ops and testing, or take advantage of advances in system software sooner than than official support from the system vendor.

Next, we look at the RandomAccess benchmark within HPCC seen in Figure 3, which measures GUPS, or Giga

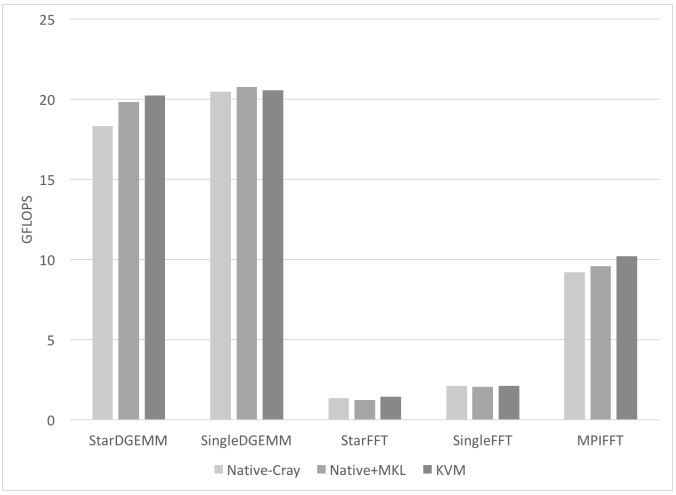


Fig. 2: Intra-node FLOPS performance

Updates per Second. Here, we observe a small but noticeable overhead with KVM across the board for RandomAccess. These results are reported using the `SANDIA_NOPT` which can give an additional boost in performance to MPIRandomAccess, both natively and within KVM. Interestingly, we do see that SingleRandomAccess incurs a more significant performance impact, potentially due to larger TLB miss costs with nested pages tables [37].

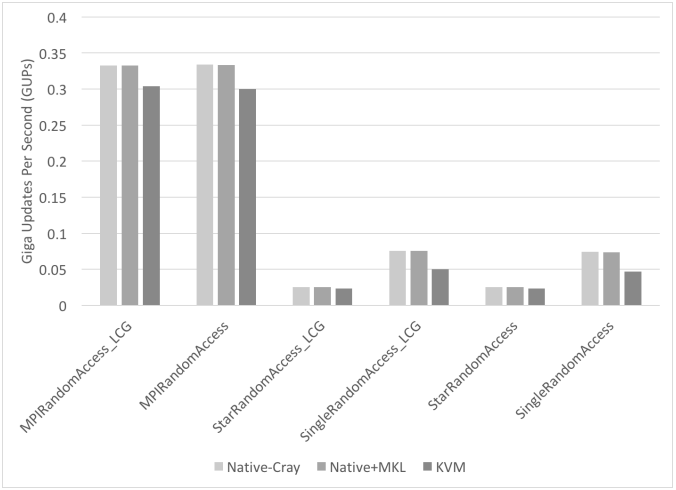


Fig. 3: Intra-node GUPS performance

Moving towards the HPCC memory benchmark STREAM in Figure 4 we again witness very good performance from the KVM implementation when compared to running natively on the Cray. Here, the benchmark reflects the added benefit of an updated and efficient guest OS. The native version is using Cray's Hugepage support with 2MB page size (to be comparable with the VM), which is based on libhugetlbfs. This is the same mechanism that is used to allocate large contiguous VM memory for the guest VM. However the VM is running an updated 3.11 Linux kernel which provides an a more advanced THP implementation, which we estimate is slightly more efficient in this case at optimizing contiguous memory patterns than libhugetlbfs. However, the potential benefit of

THP in the VM decreases as the complexity of the STREAM benchmark increases, as with Triad, which more accurately represents real-world application performance. Also, some times THP can lead to other issues, such as creating additional noise events within compute nodes as the kernel promotes or demotes THP pages, which can also lead to memory fragmentation. As such, THP usage within a virtual cluster should be carefully examined to determine its utility, however in the case of STREAM, we found little overhead.

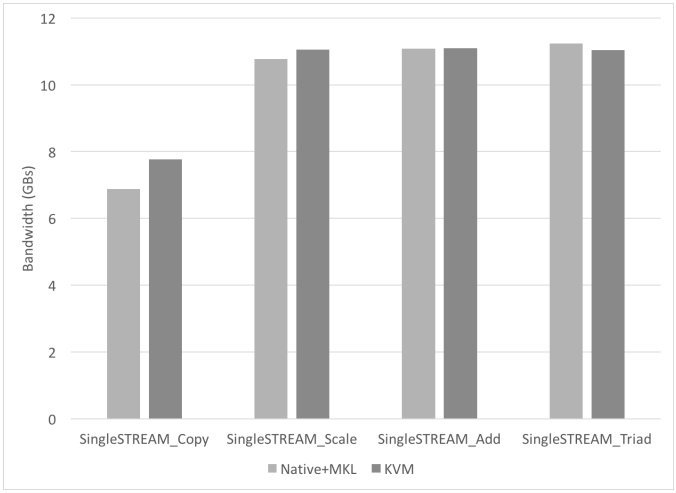
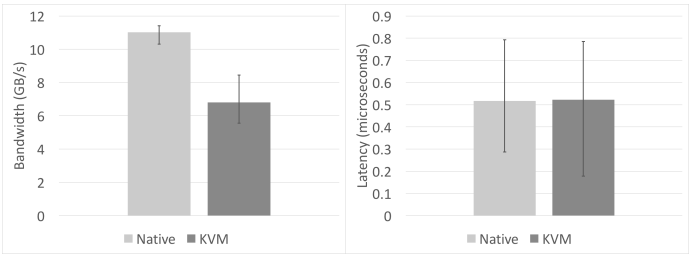


Fig. 4: Intra-node STREAM performance

Looking to the PingPong bandwidth and latency benchmarks details in Figures 5a and 5b, which in a single-node configuration measures performance of using MPI_sendrecv across a shared memory system using 8 byte messages for latency and 2MB messages. Note that the error bars in the charts represent Max and Min values for each test. Here, we see that the effective bandwidth is notably higher with the native Cray environment, with latency largely equivalent between native and KVM configurations. This is due to Cray's MPI optimization to use XPMEM for intra-node shared memory communication, whereby a single memory copy is needed. MPICH, however, performs two copies by default through a sysv shared memory buffer, leading to intra-node bandwidth degradation as observed here. This particular case illustrates why using a Cray natively may be a better option in some cases, as for MPI-based applications which are highly sensitive to intra-node communication bandwidth, running natively will yield the best performance for specific applications.

C. Mutli-node HPC Scaling

Moving on to multi-node virtual cluster configurations, we look to evaluate overall system performance using two HPC "bookends" with HPL and HPCG. As we wish to evaluate the weak scaling parameters of each application, the problem size selection based on the number of nodes becomes important, which is specified in Table I and applies to both native Cray and our virtual cluster. For HPCG, we let the default problem size adjustment take place which accompanies the application itself, which assumes roughly 25% of total system memory to avoid potential caching effects. With HPL, there exists



(a) MPI PingPong Bandwidth (b) MPI PingPong Latency

Fig. 5: HPCC MPI intra-node communication performance, with 2MB messages for bandwidth and 8 byte messages for latency

TABLE I: Problem Sizes listed in Gigabytes and N value for HPL and HPCG, respectively

Nodes	1(24)	2(48)	4(96)	8(192)	16(386)	32(768)
HPCG (GB)	19.3	38.6	77.2	154.5	308.9	617.9
HPL (N)	57920	81920	115840	163840	231680	327680

a balance between fitting the problem size to the highest possible value to perfectly fit available main memory and total benchmark runtime. As such, we chose to calculate HPL problem size (N) based on using 32GB of RAM, which is enough for producing acceptable performance and effectively using both NUMA domains, but still keeping overall runtime of each run on the order of minutes.

Experimental results for HPL are given in Figure 6, where data points represent weak scaling as node count increases. For native experimentation, we focused on the standard Cray compilation mechanisms for HPL which uses LibSci, as it is the most likely way in which the vast majority of users will utilize such a system. For a single node experiment, HPL shows a slight performance boost running in a virtual cluster, as expected based from other single-node FLOPS benchmarks previously detailed in Figure 2. Again, this is a direct result of the boost in performance by using the latest Intel and MKL libraries over the older Intel compiler and LibSci natively. However, this small performance boost quickly disappears and KVM overhead increases to about 18% as the problem size and node count increases. This overhead is due to the Ethernet-over-Aries bridged network within the virtual cluster, compared to using the Aries interconnect natively. Effectively, the HPL benchmark illustrates that the biggest challenge to providing virtual clusters on supercomputing resources is not the performance overhead of a single node, but instead with the ability to efficiently use high speed, low latency interconnects that are available. Specifically, the Ethernet bridge solution is adding significant bandwidth overhead to HPL as well as additional latency, and as such lowers overall application performance.

A similar, but less pronounced effect is found by looking at the HPCG benchmark in Figure 7. Again, the application is scaling up to 32 nodes and 768 cores. Single node performance illustrates just a 3.6% overhead when running HPCG, effectively providing near-native performance in a VM. However, this overhead grows to around 9-12% as the node

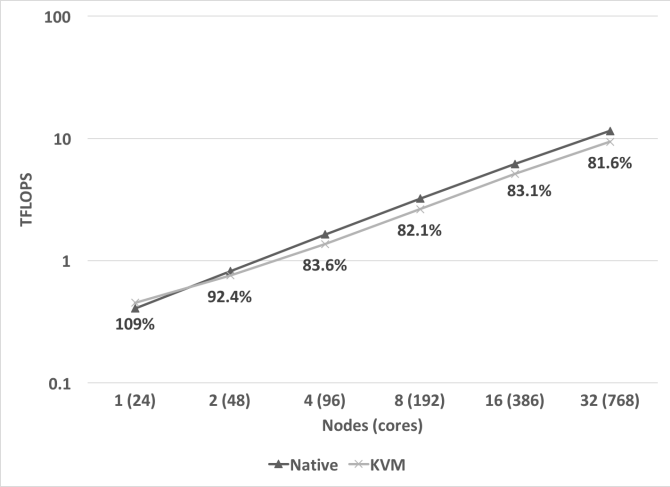


Fig. 6: Weak Scaling of the High Performance Linpack benchmark to 32 nodes

count increases. As with scaling HPL, this overhead is due to the decreased bandwidth and increased latency when using the Ethernet solution within a VM compared to native Aries. However, given the network limitations expressed, we are still generally surprised to see a benchmark focused on real-world application performance yield closely comparable performance to native on the Cray. With this result, we expect that if better network virtualization can be realized on future Cray architectures, virtual clusters could be a viable solution for medium-scale HPC applications, as well as emerging HPC runtimes.

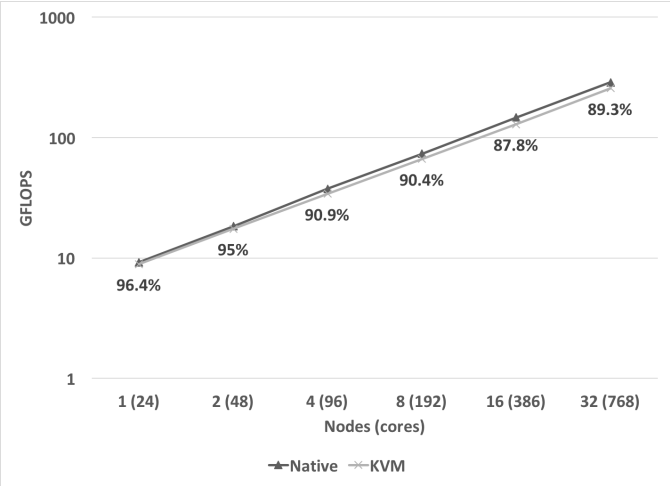


Fig. 7: Weak scaling of the HPCG benchmark up to 32 nodes

D. Apache Spark

Leveraging our Spark virtual cluster, we ran the TeraSort benchmark [44]. TeraSort is a common and simplistic benchmark, originally designed for Hadoop that was intended to sort large (1TB) data sizes, even though the benchmark developed for Spark allows for smaller problem sizes which enable data to stay in-memory. TeraSort proves to be of interest as creates an all-to-all shuffle between the Map and Reduce phases which stresses data movement significantly, with this stress growing substantially as the problem size increases. For Spark worker configurations running on the compute nodes, we specified the use of 24 cores (matching physical cores) and 32GB of RAM.

TeraSort was run on two problem sizes, 10GB and 100GB datasets generated randomly. While this is less than a full terabyte of data, it allows for the machine to still be stressed, and for a full shuffle operation to stress the memory and interconnect limits of the Spark cluster. For the 10GB problem size, we see that the shuffle easily stays within memory and our computation completes in 3 minutes 9 seconds on average. For moving to a problem size that's an order of magnitude larger, we see the total runtime increases by two orders of magnitude to over 8 hours. Here, we believe we're stressing the limits of the NFS server. With this result, we hope future endeavors will be able to leverage more node-local storage such as Burst Buffers [45] or parallel file systems such as Lustre which would expectedly improve performance drastically.

Unfortunately, a comparison could not be made between a native Spark implementation because we found the native Cray environment unsuitable for users to effectively run Spark on in any meaningful capacity. First, much of Spark's user-friendly scripts utilize ssh, however user ssh login on a compute node is not utilized natively. Of course, we can create custom job scripts to start Spark Worker processes directly on the compute nodes, however we still ran into roadblocks, specifically with the DVS storage mechanisms. Specifically, DVS, which in Volta is an overlay of NFS (not Lustre) generates numerous errors running Spark with larger files. For testing a simple 10Gb TeraSort, we were unable to complete a run. We did develop work-arounds that proved futile. This included running `rsync` to move input data to tmpfs on each node, which is very inefficient and impractical any any realistic scale. We also directly mounted the underlying NFS mount on each compute node, which too is impractical as doing so requires full root privileges on the Cray to generate, distribute, and insert NFS modules (and its dependencies) to each compute node, something normal users can not do. However, for users with full Lustre back-end systems, it is worth noting that one can use Lustre to run Spark standalone mode on a Cray XC, as seen in related research [10]. This work shows that a Lustre metadata server introduces a bottleneck, and that mounting loopback devices from Lustre can help alleviate performance impacts. Given the fact that our virtual cluster machine images can also could be hosted with loopback on Lustre, we would expect similar performance benefits if our solution was deployed on a Cray XC with Lustre.

To help further define an understanding of Spark performance on virtual clusters running atop a Cray XC supercomputing testbed, we ran the Spark test suite within spark_perf [46], [47]. This benchmark suite covers a number of common operations that are performed in a Map Reduce framework, with the ability to increase a scale factor accordingly to increase problem size, as demonstrated in data presented in Table II. Each value represents the time spent on a given benchmark problem, and is the median of 10 trials. As we can see, increasing the scale by a factor of 10 roughly doubles the

TABLE II: Benchmark results for Spark Perf running on the Volta Spark Virtual Cluster (seconds)

Scale	Throughput	Aggr-by-key	Aggr-by-key-int	Aggr-by-key-naive	Sort-by-key	Sort-by-key-int	Count	Count-filter
0.001	2.6585	0.106	0.1085	0.199	0.114	0.1125	0.034	0.0575
0.01	2.6285	0.219	0.1905	0.437	0.3065	0.3765	0.0395	0.0935
0.1	2.683	0.474	0.4135	0.9605	0.839	0.7075	0.056	0.1495
1.0	2.6975	2.24	1.886	5.19	2.976	1.797	0.162	0.2665
10.0	2.642	15.429	47.629	32.9335	5.378	3.9455	1.1095	1.1935

runtime most benchmarks for small problem sizes, which illustrates increased parallelization of the problem with increasing problem size. As the problem sizes continues to increase, the runtimes start increase more drastically. Aggregate-by-key benchmarks see roughly a 4-5x increase in runtime for a 10x larger problem. While this is still efficient, it indicates the problem size is saturating the cluster. Eventual over-saturation occurs with Aggregate-by-key with a scaling factor of 10.0, which is aggregating 4 billion records with 10 million unique values for 400 thousand unique key integers. The exception to scaling is the Scheduling Throughput benchmark, which consistently schedules 10,000 tasks in under 2.7 seconds.

V. Discussion

Given the results described in this paper, there are a number of avenues for our prototype virtual cluster on Cray supercomputing resources to be extended. First, we hope the limitations pointed out in this paper, in particular the effect of Ethernet emulation over the Aries proprietary interconnect can be addressed with future hardware designs. While achieving absolute native performance is not strictly necessary, near-native performance with small, deterministic overhead as seen with technologies such as SR-IOV [48] would represent a important step forward in hardware technology.

Second, we hope the prototype can move towards further refinement to help support other novel workloads on HPC hardware. This work could move laterally within the system software stack with additional performance capabilities, such as additional hardware like accelerators or integration with additional storage options such as Burst Buffers or parallel filesystems. This prototype virtual cluster solution could be refined for use on a much larger system, such as Trinity, or other future Cray platforms. Work could also move vertically with integration towards a larger virtual cluster orchestration efforts to enhance user experience and help lower the barrier of entry of HPC for emerging scientific computational problems. Existing container solutions could be integrated within virtual clusters, potentially creating the overlay of native Docker on a Cray supercomputer, effectively enabling the hypervisor and host OS kernel to handle security concerns and hardware control rather than just a single OS kernel layer. While this could be viewed as orthogonal to HPC-container efforts such as Shifter [26], we in fact view this effort as complimentary to virtualization where the union could collectively enhance experiences for both users and facilities system administrators.

The use of virtual clusters enables users to focus on application ecosystem composition matching desired scientific endeavors, rather than forcing development environments to adapt to HPC platforms that were never designed to support such workloads. Static batch job schedulers with vendor-specific OS and library services on most supercomputing resources are tuned primarily for MPI-based execution models, not data analytics. Effectively, we hope running Virtual Clusters at high efficiency can lower the barrier of entry to extreme-scale computing for many emerging computational tools embodied by the 4th paradigm of science [49]. Additionally, we may be able to construct a framework of scientific experiment management where Virtual Clusters and their environments can be built, shared, rerun, and archived upon demand across the greater scientific community.

Last, we envision related research that looks to cross-cut convergence between HPC and big data analytics from a software layer to use virtual clusters. By providing a software ecosystem that runs well on a supercomputing platform and is also more amenable to existing commodity analytics environments than native HPC software stacks, development enhancements to tools such as the Apache Big Data Stack can look to leverage HPC resources more quickly and effectively than porting existing efforts to an HPC-optimized system software stack or starting from scratch.

VI. Conclusion

This manuscript has described the design, implementation, and experimentation of building High Performance Virtual Clusters using a specialized Cray supercomputing testbed. These virtual clusters can extend the a supercomputing platform, in this case a XC30 testbed, to support a wide range of system software ecosystems. As an example, Apache Spark workloads were run as a custom virtual cluster, which was not possible on an XC30 natively at the time of writing, given the limitations of the HPC environment. This effort also leverages traditional HPC benchmarking tools such as HPCC and HPCG to evaluate the performance of virtual clusters against the native vendor software solution, both on single node and when scaling up to 768 cores. Overall, we find the efficiency of the virtualization mechanisms with KVM to provide reasonable performance, but the best-effort networking solution with an Ethernet-over-Aries emulation system presents challenges in scaling with application overheads ranging form 10-20% across all resources. However, it is our hope that this research will serve as motivation to drive further development in upcoming architecture designs which could alleviate much of this overhead drastically in the future. In this, we find this research opens the door for other emerging system software beyond what is capable today, perhaps with future OS designs or scalable large data frameworks to directly leverage advanced supercomputing resources.

REFERENCES

[1] D. A. Reed and J. Dongarra, "Exascale computing and big data," *Communications of the ACM*, vol. 58, no. 7, pp. 56–68, 2015.

[2] R. Leland, R. Murphy, B. Hendrickson, K. Yelick, J. Johnson, and J. Berry, "Large-Scale Data Analytics and Its Relationship to Simulation," Sandia National Laboratories, Tech. Rep., 2016.

[3] S. Jha, J. Qiu, A. Luckow, P. K. Mantha, and G. C. Fox, "A tale of two data-intensive paradigms: Applications, abstractions, and architectures," in *Proceedings of the 3rd International Congress on Big Data*, 2014.

[4] J. Qiu, S. Jha, A. Luckow, and G. C. Fox, "Towards hpc-abds: An initial high-performance big data stack," *Building Robust Big Data Ecosystem ISO/IEC JTC 1 Study Group on Big Data*, pp. 18–21, 2014.

[5] M. W. ur Rahman, N. S. Islam, X. Lu, J. Jose, H. Subramoni, H. Wang, and D. K. D. Panda, "High-performance rdma-based design of hadoop mapreduce over infiniband," in *Parallel and Distributed Processing Symposium Workshops PhD Forum (IPDPSW), 2013 IEEE 27th International*, May 2013, pp. 1908–1917.

[6] G. C. Fox, J. Qiu, S. Kamburugamuve, S. Jha, and A. Luckow, "Hpc-abds high performance computing enhanced apache big data stack," in *2015 15th IEEE/ACM International Symposium on Cluster, Cloud and Grid Computing*, May 2015, pp. 1057–1066.

[7] J. Dongarra, H. Meuer, and E. Strohmaier, "Top 500 supercomputers," website, November 2013.

[8] J. R. Lange, K. Pedretti, P. Dinda, P. G. Bridges, C. Bae, P. Soltero, and A. Merritt, "Minimal-overhead virtualization of a large scale supercomputer," in *ACM SIGPLAN Notices*, vol. 46, no. 7. ACM, 2011, pp. 169–180.

[9] A. J. Younge, J. P. Walters, S. P. Crago, and G. C. Fox, "Supporting high performance molecular dynamics in virtualized clusters using iommu, sr-iov, and gpudirect," in *ACM Virtual Execution Environments (VEE) 2015, in conjunction with ACM ASPLOS*, vol. 50, no. 7. ACM, 2015, pp. 31–38.

[10] N. Chaimov, A. Malony, S. Canon, C. Iancu, K. Z. Ibrahim, and J. Srinivasan, "Performance evaluation of apache spark on cray xc systems," in *Cray User Group 2016 (CUG)*, 2016.

[11] G. M. Amdahl, "Validity of the single processor approach to achieving large scale computing capabilities," in *Proceedings of the April 18-20, 1967, spring joint computer conference*. ACM, 1967, pp. 483–485.

[12] T. Sterling, *Beowulf cluster computing with Linux*. The MIT Press, 2001.

[13] K. Hwang, J. Dongarra, and G. C. Fox, *Distributed and cloud computing: from parallel processing to the internet of things*. Morgan Kaufmann, 2013.

[14] J. Moore, D. Irwin, L. Grit, S. Sprenkle, and J. Chase, "Managing mixed-use clusters with cluster-on-demand," *Duke University Department of Computer Science Technical Report*, 2002.

[15] J. S. Chase, D. E. Irwin, L. E. Grit, J. D. Moore, and S. E. Sprenkle, "Dynamic virtual clusters in a grid site manager," in *High Performance Distributed Computing, 2003. Proceedings. 12th IEEE International Symposium on*. IEEE, 2003, pp. 90–100.

[16] K. Keahey, I. Foster, T. Freeman, and X. Zhang, "Virtual workspaces: Achieving quality of service and quality of life in the grid," *Scientific programming*, vol. 13, no. 4, pp. 265–275, 2005.

[17] I. Foster, T. Freeman, K. Keahy, D. Scheftner, B. Sotomayer, and X. Zhang, "Virtual clusters for grid communities," in *Cluster Computing and the Grid, 2006. CCGRID 06. Sixth IEEE International Symposium on*, vol. 1. IEEE, 2006, pp. 513–520.

[18] T. Bell, B. Bompastor, S. Bukowiec, J. C. Leon, M. Denis, J. van Eldik, M. F. Lobo, L. F. Alvarez, D. F. Rodriguez, A. Marino *et al.*, "Scaling the cern openstack cloud," in *Journal of Physics: Conference Series*, vol. 664, no. 2. IOP Publishing, 2015, p. 022003.

[19] R. Brightwell, R. Oldfield, A. B. Maccabe, and D. E. Bernholdt, "Hobbes: Composition and virtualization as the foundations of an extreme-scale os/r," in *Proceedings of the 3rd International Workshop on Runtime and Operating Systems for Supercomputers*. ACM, 2013.

[20] J. Lange, K. Pedretti, T. Hudson, P. Dinda, Z. Cui, L. Xia, P. Bridges, A. Gocke, S. Jaconette, M. Levenhagen *et al.*, "Palacios and kitten: New high performance operating systems for scalable virtualized and native supercomputing," in *Parallel & Distributed Processing (IPDPS), 2010 IEEE International Symposium on*. IEEE, 2010, pp. 1–12.

[21] S. M. Kelly and R. Brightwell, "Software architecture of the light weight kernel, catamount," in *Proceedings of the 2005 Cray User Group Annual Technical Conference*. Citeseer, 2005, pp. 16–19.

[22] A. J. Younge, R. Henschel, J. T. Brown, G. von Laszewski, J. Qiu, and G. C. Fox, "Analysis of Virtualization Technologies for High Performance Computing Environments," in *Proceedings of the 4th International Conference on Cloud Computing (CLOUD 2011)*. Washington, DC: IEEE, July 2011.

[23] K. Keahey, "Chameleon: Building an experimental instrument for computer science as application of cloud computing," HEPiX at LBNL, Tech. Rep., Oct 2016.

[24] P. Barham, B. Dragovic, K. Fraser, S. Hand, T. L. Harris, A. Ho, R. Neugebauer, I. Pratt, and A. Warfield, "Xen and the art of virtualization." in *SOSP*, M. L. Scott and L. L. Peterson, Eds. ACM, 2003, pp. 164–177.

[25] D. Merkel, "Docker: lightweight linux containers for consistent development and deployment," *Linux Journal*, vol. 2014, no. 239, p. 2, 2014.

[26] D. M. Jacobsen and R. S. Canon, "Contain this, unleashing docker for hpc," *Proceedings of the Cray User Group*, 2015.

[27] S. Ekanayake, S. Kamburugamuve, and G. Fox, "Spidal: High performance data analytics with java and mpi on large multicore hpc clusters," in *Proceedings of 24th High Performance Computing Symposium (HPC 2016)*, 2016.

[28] F. Tian and K. Chen, "Towards optimal resource provisioning for running mapreduce programs in public clouds," in *Cloud Computing (CLOUD), 2011 IEEE International Conference on*. IEEE, 2011, pp. 155–162.

[29] T. Gunarathne, J. Qiu, and D. Gannon, "Towards a collective layer in the big data stack," in *Cluster, Cloud and Grid Computing (CCGrid), 2014 14th IEEE/ACM International Symposium on*. IEEE, 2014, pp. 236–245.

[30] N. S. Islam, M. Rahman, J. Jose, R. Rajachandrasekar, H. Wang, H. Subramoni, C. Murthy, and D. K. Panda, "High performance rdma-based design of hdfs over infiniband," in *Proceedings of the International Conference on High Performance Computing, Networking, Storage and Analysis*. IEEE Computer Society Press, 2012, p. 35.

[31] S. K. andKarthik Ramasamy, M. Swany, and G. Fox, "Low latency stream processing: Twitter heron with infiniband and omni-path," Indiana University Bloomington, Tech. Rep., 2017.

[32] L. Kaplan, "Cray cnl," FastOS PI Meeting & Workshop, Tech. Rep. [Online]. Available: http://www.cs.unm.edu/fastos/07meeting/CNL FASTOS.pdf

[33] K. S. Hemmert, M. W. Glass, S. D. Hammond, R. Hoekstra, M. Rajan, S. Dawson, M. Vigil, D. Grunau, J. Lujan, D. Morton *et al.*, "Trinity: Architecture and early experience," in *Cray Users Group*, 2016.

[34] B. Gerofi, M. Takagi, A. Hori, G. Nakamura, T. Shirasawa, and Y. Ishikawa, "On the scalability, performance isolation and device driver transparency of the ihk/mckernel hybrid lightweight kernel," in *Parallel and Distributed Processing Symposium, 2016 IEEE International*. IEEE, 2016, pp. 1041–1050.

[35] R. W. Wisniewski, T. Inglett, P. Keppel, R. Murty, and R. Riesen, "mos: An architecture for extreme-scale operating systems," in *Proceedings of the 4th International Workshop on Runtime and Operating Systems for Supercomputers*. ACM, 2014, p. 2.

[36] O. Sefraoui, M. Aissaoui, and M. Eleuldj, "Openstack: toward an open-source solution for cloud computing," *International Journal of Computer Applications*, vol. 55, no. 3, 2012.

[37] R. Bhargava, B. Serebrin, F. Spadini, and S. Manne, "Accelerating two-dimensional page walks for virtualized systems," in *ACM SIGARCH Computer Architecture News*, vol. 36, no. 1. ACM, 2008, pp. 26–35.

[38] R. Russell, "virtio: towards a de-facto standard for virtual i/o devices," *ACM SIGOPS Operating Systems Review*, vol. 42, no. 5, pp. 95–103, 2008.

[39] M. Musleh, V. Pai, J. P. Walters, A. J. Younge, and S. P. Crago, "Bridging the Virtualization Performance Gap for HPC using SR-IOV for InfiniBand," in *Proceedings of the 7th IEEE International Conference on Cloud Computing (CLOUD 2014)*. Anchorage, AK: IEEE, 2014.

[40] A. Tirumala, F. Qin, J. Dugan, J. Ferguson, and K. Gibbs, "Iperf: The tcp/udp bandwidth measurement tool," Webpage, 2005. [Online]. Available: http://iperf. sourceforge. net

[41] P. R. Luszczek, D. H. Bailey, J. J. Dongarra, J. Kepner, R. F. Lucas, R. Rabenseifner, and D. Takahashi, "The hpc challenge (hpcc) benchmark suite," in *Proceedings of the 2006 ACM/IEEE conference on Supercomputing*. Citeseer, 2006, p. 213.

[42] J. Dongarra, M. A. Heroux, and P. Luszczek, "Hpcg benchmark: A new metric for ranking high performance computing systems," *Knoxville, Tennessee*, 2015.

[43] M. Zaharia, M. Chowdhury, T. Das, A. Dave, J. Ma, M. McCauley, M. J. Franklin, S. Shenker, and I. Stoica, "Resilient distributed datasets: A fault-tolerant abstraction for in-memory cluster computing," in *Proceedings of the 9th USENIX conference on Networked Systems Design and Implementation*. USENIX Association, 2012, pp. 2–2.

[44] O. OMalley, "Terabyte sort on apache hadoop," *Yahoo, available online at: http://sortbenchmark. org/Yahoo-Hadoop. pdf,(May)*, pp. 1–3, 2008.
[45] K. Antypas, N. Wright, N. P. Cardo, A. Andrews, and M. Cordery, "Cori: a cray xc pre-exascale system for nersc," *Cray User Group Proceedings. Cray*, 2014.
[46] P. W. et al., "Spark-perf: Performance tests for spark," Webpage, 2016. [Online]. Available: https://github.com/databricks/spark-perf
[47] X. Meng, J. Bradley, B. Yavuz, E. Sparks, S. Venkataraman, D. Liu, J. Freeman, D. Tsai, M. Amde, S. Owen *et al.*, "Mllib: Machine learning in apache spark," *Journal of Machine Learning Research*, vol. 17, no. 34, pp. 1–7, 2016.
[48] J. Jose, M. Li, X. Lu, K. C. Kandalla, M. D. Arnold, and D. K. Panda, "Sr-iov support for virtualization on infiniband clusters: Early experience," in *Cluster, Cloud and Grid Computing (CCGrid), 2013 13th IEEE/ACM International Symposium on*. IEEE, 2013, pp. 385–392.
[49] T. Hey, S. Tansley, K. M. Tolle *et al.*, *The fourth paradigm: data-intensive scientific discovery*. Microsoft research Redmond, WA, 2009, vol. 1.