System-Level Support for Composition of Applications

Brian Kocoloski
John Lange
Dept. of Computer Science
University of Pittsburgh
Pittsburgh, PA 15260 USA
{briankoco,jacklange}
@cs.pitt.edu

Hasan Abbasi
David E. Bernholdt
Terry R. Jones
Oak Ridge Nat'l Laboratory
P.O.B. 2008
Oak Ridge, TN 37831 USA
{habbasi,bernholdtde,trjones}
@ornl.gov

Jai Dayal
Georgia Institute of
Technology
College of Computing
Atlanta, GA 30332 USA
jdayal3@gatech.edu

Noah Evans
Michael Lang
Los Alamos National
Laboratory
Los Alamos, NM 87544 USA
noah.evans@gmail.com,
mlang@lanl.gov

Jay Lofstead
Kevin Pedretti
Sandia National Laboratories
P.O. Box 5800
Albuquerque, NM 87123 USA
{gflofst,ktpedre}
@sandia.gov

Patrick G. Bridges
Dept. of Computer Science
University of New Mexico
Albuquerque, NM 87131 USA
bridges@cs.unm.edu

ABSTRACT

Current HPC system software lacks support for emerging application deployment scenarios that combine one or more simulations with in situ analytics, sometimes called multi-component or multi-enclave applications. This paper presents an initial design study, implementation, and evaluation of mechanisms supporting composite multi-enclave applications in the Hobbes exascale operating system. These mechanisms include virtualization techniques isolating application custom enclaves while using the vendor-supplied host operating system and high-performance inter-VM communication mechanisms. Our initial single-node performance evaluation of these mechanisms on multi-enclave science applications, both real and proxy, demonstrate the ability to support multi-enclave HPC job composition with minimal performance overhead.

Categories and Subject Descriptors

D.4.7 [**Operating Sytems**]: Organization and Design; C.5.1 [**Computer System Implementation**]: Super (very large) computers; C.1.2 [**Multiprocessors**]: Parallel Processors

1. INTRODUCTION

Emerging HPC applications are increasingly composed of multiple communicating, cooperating components. This includes coupled simulation + analytics workflows, coupled multiphysics simulations and scalable performance analysis and debugging systems [35, 37, 27, 25, 30, 32, 7]. This emergence is a result of the changing compute/memory/IO balance of high-end computer systems, the push toward increasing physical fidelity and realism in simulations motivating increasing use of coupled multiphysics simulations, and the generally increasing size and richness of data generated. Additionally, a compositional approach has the potential to help HPC application developers address a wider range of scientific problems; quickly prototype and develop new simulations; customize application behavior to underlying system communication, I/O and memory characteristics; and more easily handle the myriad challenges of extreme-scale scientific computing.

System software support is both essential to enable application composition and lacking in current HPC system software stacks. For example, current HPC systems are optimized for allocating a disjoint subset of system nodes to a single application. Most HPC OSes are not optimized to support or enable co-location and communication between cooperating executables. This has forced HPC application designers to combine multiple components into a single executable using hard-to-maintain library tricks that result in multi-gigabyte executables or to split components spatially between nodes, with the latter approach increasing data movement, power, and general resource usage compared to systems supporting co-location [38].

This paper presents a preliminary design study and initial evaluation of an operating system/runtime (OS/R) environment, Hobbes, with explicit support for composing HPC applications from multiple cooperating components. The design is based on our previously presented vision [9] and makes systematic use of both virtualization and lightweight operating systems techniques to support multiple communicating application enclaves per node. In addition, it also includes efficient inter-enclave communication tools to enable application composition. Furthermore, we show that our Hobbes OS/R supports the composition of applications across multiple isolated enclaves with little to no performance overhead.

ROSS '15, June 16, 2015, Portland, Oregon, USA

http://dx.doi.org/10.1145/2768405.2768412

The main contributions of this work are:

- An analysis of application requirements for composition in next-generation HPC systems;
- The design of an integrated HPC operating system and runtime (OS/R) environment that meets those requirements;
- A prototype implementation of the key components of this design; and
- An initial single node evaluation of the performance of this implementation on multi-enclave HPC benchmark tests.

The remainder of this paper is organized as follows. Section 2 presents an overview of HPC application composition use-cases and ways such applications can benefit from the environment provided by the Hobbes OS/R. Section 3 describes the high-level design aspects of the Hobbes OS/R to support composition, and presents a detailed description of the key software components comprising the OS/R environment. Section 4 follows with a description of three example applications from the simulation + analytics category and demonstrates their implementation in the current Hobbes OS/R prototype, while Section 5 presents a preliminary evaluation of two of these applications in a set of multi-enclave environments. Section 6 describes related work and Section 7 discusses the results presented as well as future work. Section 8 provides a final summary.

2. APPLICATION COMPOSITION

While computational science and engineering applications embody a great deal of diversity, we have identified a number of composite application generic use case categories. These include:

- **Simulation + analytics** involves a simulation component communicating with one providing data analytics, visualization, or similar services. In most cases, data is presumed to flow from the simulation to the analytics component one way. However computational steering could introduce a reciprocal data flow, shared data, and/or a control interaction from the analytics to the simulation. While dynamic compositions are possible, we expect most applications of this type to involve static compositions where the components and their deployments are defined in advance and stay fixed throughout the execution of the application. Three examples are described in this paper.
- **Coupled multiphysics simulations** involve multiple simulation components, each modeling different physics, working cooperatively. Data may flow one-way, two-ways, or may be shared between components. There may also be control interactions between components. Coupled multiphysics simulations may be either static or dynamic. For example, there are an increasing number of examples in the literature of complex simulations in which new computational tasks (components, in this context) are instantiated on demand during a simulation and have a lifetime much shorter than the overall simulation. Examples of this category include [25, 30, 32, 7].
- **Application introspection** is a novel way of thinking about activities that need deep access to an application such as performance monitoring or debugging. Whereas today such applications usually involve complex combinations of simulation and "tool" processes, which must "attach" to them, in the Hobbes environment, such applications could be cast as co-located enclaves for which the communications can be defined as appropriate.

Composite applications can benefit from the Hobbes environment in several ways. Components may be designed with very different requirements from the underlying operating system. For example, a simulation component with few OS requirements might run well in a lightweight kernel (LWK) environment with minimal local performance overheads and inter-node jitter while an analytics component might require the richer services provided by a full Linux OS. Or two components in a coupled simulation might require different, incompatible runtime systems when used within the same executable or might not share resources well. Composition also allows increased component deployment flexibility in ways that can both accelerate computation and reduce resource requirements. For example, simulation and analytics components that require different runtime environments can be consolidated on the same node allowing communication to occur at memory speed rather than over a network.

3. THE HOBBES OS/R

The design of the Hobbes OS/R has been guided by three main considerations:

- The OS/R should allow dynamic (re-)configuration at runtime of hardware resource assignments and system software environments.
- The OS/R should provide a single, unified user space runtime API inside each enclave to support configuration flexibility and minimize application development effort.
- The OS/R should provide configurable isolation levels for each enclave in the system to meet the performance isolation requirements of each application.

We have designed the Hobbes OS/R architecture as a configurable runtime environment that allows dynamic configuration of isolated enclaves consisting of partitioned hardware resources and specialized system software stacks. The Hobbes environment provides a number of features to support composite applications including APIs for communication and control abstractions as well as languages describing the interactions, compositions, and deployment configurations of the target system environment.

In order to make our approach accessible to existing application code bases, we have sought to ensure that composition can be achieved with minimal developer effort. The design of our composable architecture is focused on the high level goal of allowing exascale users to seamlessly compose application components at runtime as part of a job definition. Application components consist of individual workloads capable of functioning independently inside dedicated OS/R instances. Each component is coupled to others using a common inter-enclave communication framework that is portable across each OS/R environment. This approach allows unmodified application binaries to be deployed into arbitrary enclave topologies without modification. Moreover, our system leverages existing APIs to provide wide portability (namely the XPMEM and ADIOS library interfaces) for existing code bases without modification.

The high-level overview of our design is illustrated in Figure 1. As the foundation for our architecture, we have based our work on the Kitten lightweight kernel [22] and the Pisces

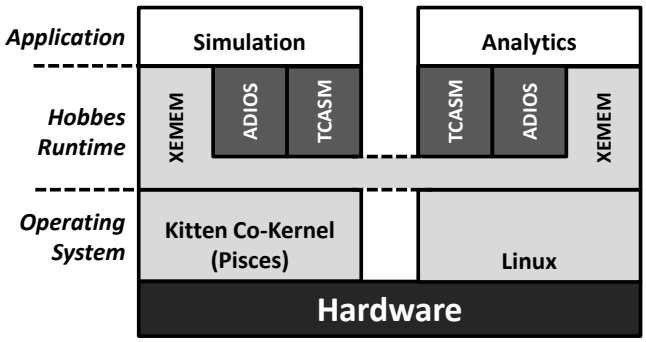

Figure 1: The Hobbes OS/R Supporting an Application Composed in Two Enclaves

lightweight co-kernel architecture [28] along with the Palacios VMM [22, 21] to support Virtual Machine based enclaves. Our approach provides the runtime provisioning of isolated enclave instances that can be customized to support each application component. Additionally, our approach allows application composition through the use of cross enclave shared memory segments through the XEMEM shared memory system [20], whose application interface can be accessed using existing I/O mechanisms such as ADIOS [24] or TCASM [5].

3.1 Operating System Components

Kitten and Palacios. Kitten [22] is a special-purpose OS kernel designed to provide a simple, lightweight environment for executing massively parallel HPC applications. Like previous lightweight kernel OSes, such as Catamount [18] and CNK [14], Kitten uses simple resource management policies (e.g., physically contiguous memory layouts) and provides direct user-level access to network hardware (OS bypass). A key design goal of Kitten is to execute the target workload – highly-scalable parallel applications with non-trivial communication and synchronization requirements – with higher performance and more repeatable performance than is possible with general purpose operating systems. Kitten also supports virtualization capabilities through its integration with Palacios.

Palacios [22] is an open source VMM designed to be embeddable into diverse host OSes and currently fully supports integration with Linux and Kitten host environments. When integrated with Kitten co-kernel hosts, Kitten and Palacios act as a lightweight hypervisor providing full system virtualization and isolation for unmodified guest OSes. The combination of Kitten and Palacios has been demonstrated to provide near native performance for large-scale HPC applications using lightweight VMs running atop a Kitten host environment [21].

Pisces Co-Kernel Architecture. Pisces [28] is a co-kernel architecture designed to allow multiple specialized OS/R instances to execute concurrently on the same local node. Pisces enables the decomposition of a node's hardware resources (CPU cores, memory blocks, and I/O devices) into partitions that are fully managed by independent system software stacks, including OS kernels, device drivers, and I/O management layers. Using Pisces, a local compute node can initialize multiple Kitten OS instances as co-kernel enclaves executing alongside an unmodified Linux host OS. Furthermore, by leveraging Palacios support, virtual machine instances can be created on top of these co-kernels as well. Pisces supports the dynamic assignment and revocation of resources between enclaves. Full co-kernel instances may be created and destroyed in response to workload requirements (e.g., application launch and termination), or individual resources may be revoked from or added to running instances. Specific details of these operations are presented elsewhere [28].

3.2 Runtime Components

XEMEM: Cross Enclave Memory. The XEMEM shared memory architecture [20] supports application-level shared memory communication across enclaves (co-kernels and/or Palacios VMs). XEMEM exports a user-level API that is backwards compatible with the API exported by SGI/Cray's XPMEM shared memory implementation for Linux systems [36], which allows processes to selectively export regions of their address space to be mapped by other processes. Because the XEMEM API is supported across each enclave OS/R environment, any application targeting the API can be deployed across any multi-enclave topology without modification. XEMEM provides a single global shared memory address space through the use of globally unique memory segment IDs managed by a global name service. In addition to naming, the name service also provides global discovery of shared memory regions allowing applications to transparently map memory regions from any other OS/R instance.

ADIOS. The Adaptable I/O System (ADIOS) [24] is a high performance I/O middleware for enabling flexible data movement for leadership class scientific applications. In addition to simple I/O to a file system, ADIOS supports scalable data movement between multiple applications and between a simulation and multiple workflow components without changing the writing or reading syntax. In past work, ADIOS has been used as the interface for enabling online streaming[37], coupling multi-physics applications [13], and performing interactive visualization [11]. The design of the ADIOS interface has expanded since its inception to provide a common policy for accessing data from disks (file based methods) and for accessing data from the network (stream based methods). This common interface allows high performance applications to work with files and streams with minimal modifications.

For this work, we created a pair of new ADIOS *transport methods* for writing and reading using the XEMEM shared memory segments to work in a cross-enclave configuration. This affords changing from writing to or reading from a file, another node, or shared memory simply by changing the transport method selected for the output or input. We demonstrate this using the same unchanged application for multiple deployments.

TCASM. Transparently Consistent Asynchronous Shared Memory (TCASM) [5] is a shared memory based transport that allows a producer process, typically an HPC application, to share read-only data in situ with an observer process, typically analytics or checkpointing, without needing

to rely on manual synchronization between the producing and observing process. TCASM is implemented using copy on write semantics: the producer registers a region of memory corresponding to its data and then exports a snapshot of that memory when it reaches a consistent state. The implementation of TCASM varies according to its host system. In Linux, TCASM is implemented on top of the Linux VMA subsystem. In Kitten, TCASM is implemented as a set of extensions to the *aspace* system, Kitten's memory management layer.

However, TCASM alone is insufficient to implement cross-enclave synchronization. To enable sharing via TCASM in Hobbes, we needed to share memory not just across virtual memory regions, but across operating systems as well. To enable this, we integrated TCASM virtual regions with XEMEM cross-enclave communication. TCASM provides the copy-on-write semantics to an individual shared memory region on an enclave while XEMEM shares the read only snapshot to other enclaves and provides a mechanism for making those regions accessible.

4. EXAMPLE APPLICATIONS

To demonstrate the capabilities of the Hobbes OS/R to provide effective application composition we have focused on three example applications that represent common HPC compositions of the simulation + analytics type. These involve molecular dynamics, plasma microturbulenece, and neutronics with corresponding analytics codes. These applications had previously been coupled using ADIOS (molecular dynamics and plasma microturbulence) and TCASM (neutronics), and no changes were required at the application level to deploy them in the Hobbes environment utilizing versions of ADIOS or TCASM which had been ported to use XEMEM.

4.1 Crack Detection in Molecular Dynamics Simulations

LAMMPS (Large Scale Atomic/Molecular Massively Parallel Simulator) [29] is a molecular dynamics simulation used across a number of scientific domains, including materials science, biology, and physics. It is written with MPI (and also has options to use OpenMP and CUDA) and performs force and energy calculations on discrete atomic particles. After a number of user-defined epochs, LAMMPS outputs atomistic simulation data (positions, atom types, etc.), with the size of this data ranging from megabytes to terabytes depending on the experiment being performed.

SmartPointer [35] is an associated analytics pipeline that ingests and analyzes LAMMPS output data to detect and then scientifically explore plastic deformation and crack genesis. Effectively, the LAMMPS simulation applies stress to the modeled material until it cracks and the goal of the SmartPointer analysis is to understand the geometry of the region around that initial break. The SmartPointer analytics toolkit implements these functions to determine where and when plastic deformation occurs and to generate relevant information as the material is cracked. The toolkit itself consists of a set of analysis codes that are decomposed as separately deployable applications that are chained together via data transfers identified by named channels, i.e., an ADIOS "file name." Further details of many of the SmartPointer functions can be found elsewhere [35]. For this set of experiments, we focus on the "Bonds" analytics code, which ingests LAMMPS atomistic data and performs a nearest neighbor calculation to output a bond adjacency list, which is a pairwise representation indicating which atoms bonded together.

4.2 Plasma Microturbulence

The Gyrokinetic Toroidal Code (GTC) [26] is a 3-Dimensional Particle-In-Cell code used to study microturbulence in magnetic confinement fusion from first principles plasma theory. It outputs particle data that includes two, 2D arrays for electrons and ions respectively. Each row of the 2D array records eight attributes of a particle including coordinates, velocities, weight, and label. The last two attributes, process rank and particle local ID within the process, together form the particle label which globally identifies a particle. They are determined on each particle in the first simulation iteration and remain unchanged throughout the particle's lifetime. These two arrays are distributed among all cores and particles move across cores in a random manner as the simulation evolves resulting in an out of order particle array. In a production run at the scale of 16,384 cores, each core can output two million particles roughly every 120 second resulting in 260GB of particle data per output. GTC employs the ADIOS BP format [23], a log-structured, write-optimized file format for storing particle data.

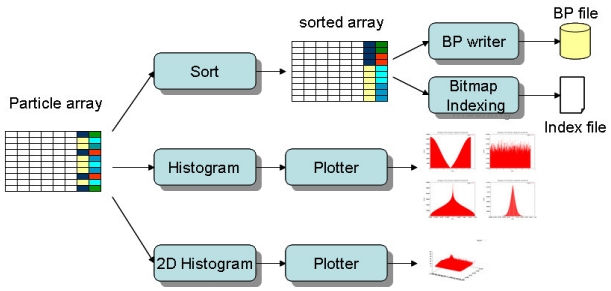

Figure 2: Illustration of PreDatA Operations on GTC Particle Data

As illustrated in Fig. 2, three analysis and preparation tasks are performed on particle data. The first involves tracking across multiple iterations a million-particle subset out of the billions of particles, requiring searching among the hundreds of 260GB files by the particle index label. To expedite this operation, particles can be sorted by the label before searching. The second task performs a range query to discover the particles whose coordinates fall into certain ranges. A bitmap indexing technique [31] is used to avoid scanning the whole particle array and multiple array chunks are merged to speed up bulk loading. The third task is to generate 1D histograms and 2D histograms on attributes of particles [17] to enable online monitoring of the running GTC simulation. 2D histograms can also be used for visualizing parallel coordinates [17] in subsequent analysis. Our example focuses on integrating the 1D and 2D histograms with GTC.

This architecture was previously demonstrated in PreDatA [37] using node-to-node or even file on disk techniques to connect the workflow components. These previous experiments showed the importance of placing analytics carefully for the best overall system performance. For this work, we have repurposed this example to use the XEMEM connec-

tion using the new ADIOS transport method. This eliminates the need to move data off node while eliminating the need for source code changes to change from node-to-node to file on disk to the shared memory interface. To facilitate operating in the limited Kitten operating environment, the GTC-P proxy application for GTC is used. GTC-P is used for porting tests, performance testing, and optimization investigations.

4.3 Neutronics Energy Spectrum Analysis

SNAP [2] is a proxy application, developed to simulate the performance workload of the discrete ordinates neutral particle transport application PARTISN [6]. PARTISN solves the linear Boltzmann equation for neutral particle transport within a material. The solution of the time-dependent transport equation is a function of seven independent variables describing the position, direction of travel, and energy of each particle, and time. PARTISN uses a domain decomposition strategy to parallelize the problem, distributing both the data and computations required to solve the Boltzmann equation. The inner loop of the application involves a parallel wavefront sweep through the spatial and directional dimensions, in the same fashion as the Sweep3D benchmark [3]. SNAP does not perform the actual physics calculations of PARTISN, rather it is designed to perform the same number of operations, use the same data layout, access the data in (approximately) the same pattern.

SNAP is coupled to an analytics code which evaluates the energy spectrum of the simulation at each time step. This application, coupled using TCASM, was originally described elsewhere [27]. At the end of each time step, the SNAP simulation publishes its data via TCASM's copy-on-write approach. The spectrum analysis code accesses each time step's data in turn and computes the spectrum, printing the results to standard output.

In the Hobbes environment, the application and analytics codes are unchanged. As preciously mentioned, TCASM itself has been modified to use XEMEM to share memory between enclaves, as opposed to using Linux VMA in the original implementation.

5. EVALUATION

In order to demonstrate the effectiveness of our Hobbes OS/R architecture to support composed applications we have evaluated both the LAMMPS and GTC compositions on a single node. The experiments were conducted on one of the compute nodes of the "Curie" Cray XK7 system at Sandia National Labs. Curie consists of 52 compute nodes, each with a 16-core 2.1 GHz 64-bit AMD Opteron 6200 CPU (Interlagos) and 32 GB in 4 channels of DDR3 RAM. The compute nodes run Cray's customized Linux OS, referred to here as Compute Node Linux (CNL), which is based on a modified SUSE version 11 (SLES 11) kernel coupled with a BusyBox user space.

For each of our experiments we compared our Hobbes based multi-enclave environment against the standard CNL environment provided by Cray. Hobbes augments Cray's standard HPC-optimized Linux OS image with the Hobbes infrastructure, which consisted of two new kernel modules (XEMEM and Pisces) that needed to be loaded in the Cray OS and a number of Hobbes user-level tools for launching and managing new enclaves.

Our experiments consisted of multiple runs of each composed application in which the application components were mapped to different enclave topologies. Application binaries used for each configuration were identical, with the only change needed being an ADIOS configuration file update to select the underlying transport. For each environment we recorded the average runtime of the application along with the standard deviation in order to evaluate performance consistency. We present these results for both the LAMMPS and GTC applications.

5.1 LAMMPS

We ran two components of our LAMMPS composition example, the LAMMPS executable and the separate Bonds executable, in several multi-enclave configurations and compared against the baseline of running both components in a single OS instance. The tested configurations along with performance results are shown in Table 1. The statistics reported were calculated from 10 runs. The baseline configuration was to run the LAMMPS and Bonds components as separate processes in the default Cray OS image. In the past these components have been coupled through the filesystem, using the ADIOS POSIX transport ('Cray-Linux (POSIX)' in the table). This is problematic because of the filesystem overhead and the difficulty of detecting when the LAMMPS output is ready for Bonds to start processing it. As can be seen, this configuration has approximately 7% higher overhead than the other configurations, which used shared-memory instead of the filesystem. Switching to the ADIOS XEMEM transport (developed by Hobbes) improved the baseline Cray OS performance significantly.

For the multi-enclave examples, the LAMMPS and Bonds components were split to run in two separate OS images (enclaves). One component was run in the Cray OS while the other component was run in either a Kitten enclave or in a Palacios enclave hosting a Linux virtual machine (VM). Additionally we evaluated running the components in two separate Kitten enclaves. These enclaves were started by using the Hobbes tools to *offline* CPU and memory resources from the Cray OS and then 'booting' either a Kitten or Palacios enclave on the offlined resources (i.e., the enclaves space shared the node's resources). The components were then cross-enclave coupled using the Hobbes XEMEM memory sharing mechanism – LAMMPS exported a region of its memory with a well known name, which Bonds then attached to via the well known name.

As can be seen in Table 1, all of the multi-enclave configurations that we evaluated produced roughly the same performance as the single OS (single enclave) baseline with shared-memory coupling. This is what we had hoped to show – that we could run this composition across multiple OS images without significantly impacting performance. This enables the use of system software customized to the needs of the code. We plan to evaluate this aspect in future work looking at multi-node scalability.

5.2 Gyrokinetic Toroidal Code (GTC)

The second set of experiments we ran focused on the GTC application. Similar to the LAMMPS experiments, we executed each application component on a collection of OS/R configurations to measure any difference in performance. Due to the OS/R feature requirements of GTC's analytics package we were unable to run the simulation directly on Kitten, and instead had to deploy it inside a Linux

Table 1: LAMMPS multi-enclave runtimes (secs)

LAMMPS Enclave	Bonds Enclave	Average	StdDev
Cray-Linux (POSIX)		165.02	0.20
Cray-Linux (XEMEM)		153.48	0.08
Cray-Linux	Kitten	153.50	0.15
Kitten	Cray-Linux	153.91	0.04
Cray-Linux	Linux-VM	153.35	0.17
Linux-VM	Cray-Linux	156.10	0.15
Kitten-Enclave1	Kitten-Enclave2	153.83	0.03

Table 2: GTC multi-enclave runtimes (secs)

GTC Enclave	Analysis Enclave	Average	StdDev
Cray-Linux (POSIX)		148.42	0.12
Cray-Linux (XEMEM)		147.40	0.09
Cray-Linux	Linux-VM	147.52	0.09

VM hosted on a Palacios/Kitten instance. In these tests, GTC runs for 100 timesteps performing output every five timesteps. It generates approximately 7.2 MB of data per process and the analysis routine generates histograms written to storage. For our examples, we run with a single process for both the simulation and the analysis routines as a proof of concept. The times presented when using POSIX represent solely the time for writing data to a RAM-disk file system while the XEMEM times include the time for generating all but the last set of histograms written to the RAM-disk file system. We chose not to include the histogram generation time as it is less than two seconds and the typical workflow coordination overheads and delays would unfairly penalize the POSIX files approach. For these tests, we perform five runs for each configuration and show the mean and standard deviation for each. The results of these experiments are shown in Table 2.

The experiments show that using the Hobbes XEMEM shared memory transport provides slightly improved and slightly more consistent performance over the native POSIX file based interface available on CNL. Furthermore, the XEMEM performance remains consistent even when the analytics workload is moved to a virtual machine based environment.

6. RELATED WORK

A wide range of virtualization systems and inter-VM communication techniques have been used to support multiple applications or enclaves with customized runtime environments. For example, cloud systems based on virtualization support multiple independent virtual machines running on the same hardware. Communication between VMs in these systems is generally supported through virtual networking techniques [33, 10] or shared memory [34]. Furthermore, systems based on OS-level virtualization and resource containers (e.g., Docker [1]) can leverage OS inter process communication tools to support communication between containers.

The primary goal of each of these systems is to maximize throughput, while providing security and hardware performance isolation to the independent VMs or resource containers. However, Hobbes differs from these systems through its focus on providing performance isolation through each layer of the system stack, including the node's system software and resource management frameworks which are independent to each enclave. The result is that Hobbes can guarantee a higher degree of performance isolation, while at the same time can selectively relax the isolation to support enclaves which *cooperate*, as needed by composed applications.

In addition to its use in cloud systems, virtualization has also been proposed for use in HPC systems to varying degrees; this includes the use of a virtual cluster to allow customization of the application development environment [21] as well as to improve application performance [19]. The work described in this paper builds on these techniques, with the additional focus of supporting multiple communicating, coordinating customized enclaves.

Much work in the HPC space has focused on facilitating application composition and coupling between simulations and analytics codes. Initial work has focused on providing streaming style data transfers between concurrently running simulation and analytics codes, both in-transit [13, 11, 15] and in situ [39, 8, 13]. More recent work has focused on providing management capabilities for these mechanisms to address interference when applications share physical resources [38, 4, 16], as well as resource allocation issues that span entire science workflows [12, 16]. The work focused on in this paper can leverage and adapt these techniques to facilitate coordination of applications composed of multiple communicating enclaves in a virtualized environment.

7. DISCUSSION AND FUTURE WORK

This paper presents an existence proof of the capabilities of the Hobbes OS/R to provide application composition support across a multi-enclave exascale environment. While the performance results show a relatively small performance benefit for our current prototype system, it is important to note that it enables a wide range of possibilities that would not otherwise be possible. For example, the Linux VM that we tested with used a more modern kernel.org 3.12.29 Linux kernel and more full-featured user-level environment than the Cray CNL environment, which uses an older 2.6.32 Linux kernel and a bare-bones user-level environment (e.g., there is no bash, no top, no strace). Many application components, such as Bonds or the GTC analytics package, benefit from having a more modern Linux kernel and/or require functionality that is missing in the vendor's base OS image. In addition, highly scalable simulation workloads often benefit from the streamlined lightweight kernel environment provided by Kitten.

While in this work our focus has mainly been on lower level system support for composition, there remains a significant amount of work left to do in regards to providing a full featured application runtime environment. Future work will likely focus on developing APIs to support the complex interactions required by composed applications, as well as supporting additional cross enclave communication constructs.

A significant challenge for deploying composed applications in a multi-enclave OS/R is the configuration and management of each component individually and collectively. In order to provide deployment and composition flexibility to application developers and users, we will need to develop a set of management interfaces and configuration languages. In particular we are investigating languages to describe the

composition of components into an application, as well as languages to describe the deployment of the application components on the target system. Ideally these languages will allow applications to be written once and deployed in any underlying system configuration, regardless of the individual enclave OS/R and communication mechanisms.

In addition to addressing the challenges of configuring and managing application and OS/R components on a single node, we must also address the complexity of managing and coordinating these enclaves across the entire exascale system. In order to enable global machine management and coordination we plan to develop interfaces to allow the integration of the Hobbes OS/R into a global information bus that is being proposed for exascale architectures.

Finally, while thus far we have focused on space shared system configurations where each application component has fully dedicated resource allocations, in the future we plan to address time shared configurations where multiple components may cooperatively share resources in order to improve resource utilization. Such support will likely require the ability to send scheduling notifications between processes in separate enclaves that could be configured to share processors, namely native processes and processes executing in Palacios VMs. Such support would enable cooperative execution in a manner similar to that proposed in GoldRush [38].

8. CONCLUSION

In this paper we have presented an implementation and evaluation of an early exascale OS/R prototype designed to enable application composition across multiple independent enclaves. Our prototype consists of the Hobbes OS/R environment which provides a set of unified application APIs to allow composed applications to be arbitrarily composed and deployed across multiple OS/Rs at runtime. We have provided an initial evaluation of our system, and shown that we can provide slightly better performance compared to current single system image environments.

9. ACKNOWLEDGMENTS

This work has been supported by US Department of Energy, Office of Science, Advanced Scientific Computing Research program.

This work was performed in part at Los Alamos National Laboratory, supported by the U.S. Department of Energy contract DE-FC02-06ER25750. This work was performed in part at the Oak Ridge National Laboratory, which is managed by UT-Battelle, LLC under contract DE-AC05-00OR22725. Sandia is a multiprogram laboratory operated by Sandia Corporation, a Lockheed Martin Company, for the United States Department of Energy's National Nuclear Security Administration under contract DE-AC04-94AL85000.

10. REFERENCES

[1] Docker. `http://www.docker.com`. Accessed: 2015-05-15.

[2] Sn application proxy. `https://github.com/losalamos/SNAP`. Accessed: 2015-03-20.

[3] Sweep3d benchmark. `http://wwwc3.lanl.gov/pal/software/sweep3d/`. Accessed: 2015-03-20.

[4] H. Abbasi, M. Wolf, G. Eisenhauer, S. Klasky, K. Schwan, and F. Zheng. DataStager: scalable data staging services for petascale applications. *Cluster Computing*, 13:277–290, 2010. 10.1007/s10586-010-0135-6.

[5] H. Akkan, L. Ionkov, and M. Lang. Transparently consistent asynchronous shared memory. In *Proceedings of the 3rd International Workshop on Runtime and Operating Systems for Supercomputers*, page 6. ACM, 2013.

[6] R. E. Alcouffe, R. S. Baker, J. A. Dahl, S. A. Turner, and R. Ward. Partisn: A time-dependent, parallel neutral particle transport code system. *Los Alamos National Laboratory, LA-UR-05-3925 (May 2005)*, 2005.

[7] N. R. Barton, J. V. Bernier, J. Knap, A. J. Sunwoo, E. K. Cerreta, and T. J. Turner. A call to arms for task parallelism in multi-scale materials modeling. *International Journal for Numerical Methods in Engineering*, 86(6):744–764, 2011.

[8] D. Boyuka, S. Lakshminarasimhan, X. Zou, Z. Gong, J. Jenkins, E. R. Schendel, N. Podhorszki, Q. Liu, S. Klasky, and N. F. Samatova. Transparent in situ data transformations in ADIOS. In *CCGrid '14*. IEEE.

[9] R. Brightwell, R. Oldfield, A. B. Maccabe, D. E. Bernholdt, E. Brewer, P. Bridges, P. Dinda, J. Dongarra, C. Iancu, M. Lang, J. Lange, D. Lowenthal, F. Mueller, K. Schwan, T. Sterling, and P. Teller. 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*, ROSS '13, pages 2:1–2:8, New York, NY, USA, 2013. ACM.

[10] Z. Cui, L. Xia, P. G. Bridges, P. A. Dinda, and J. R. Lange. Optimizing overlay-based virtual networking through optimistic interrupts and cut-through forwarding. In *Proceedings of the International Conference on High Performance Computing, Networking, Storage and Analysis*, SC '12, pages 99:1–99:11, Los Alamitos, CA, USA, 2012. IEEE Computer Society Press.

[11] J. Dayal, D. Bratcher, G. Eisenhauer, K. Schwan, M. Wolf, X. Zhang, H. Abbasi, S. Klasky, and N. Podhorszki. Flexpath: Type-based publish/subscribe system for large-scale science analytics. In *2014 14th IEEE/ACM International Symposium on Cluster, Cloud and Grid Computing, Chicago, IL, USA, May 26-29, 2014*, pages 246–255. IEEE, 2014.

[12] J. Dayal, J. Cao, G. Eisenhauer, K. Schwan, M. Wolf, F. Zheng, H. Abbasi, S. Klasky, N. Podhorszki, and J. F. Lofstead. I/O containers: Managing the data analytics and visualization pipelines of high end codes. In *HPDIC '13*, pages 2015–2024.

[13] C. Docan, M. Parashar, and S. Klasky. DataSpaces: an interaction and coordination framework for coupled simulation workflows. In *HPDC 2010*. ACM.

[14] M. Giampapa, T. Gooding, T. Inglett, and R. Wisniewski. Experiences with a Lightweight Supercomputer Kernel: Lessons Learned from Blue Gene's CNK. In *Proceedings of the 23rd International*

Conference for High Performance Computing, Networking, Storage and Analysis (SC), 2010.

[15] M. Hereld, M. E. Papka, and V. Vishwanath. Toward Simulation-Time Data Analysis and I/O Acceleration on Leadership-Class Systems. In *IEEE Symposium on Large-Scale Data Analysis and Visualization*, 2011.

[16] T. Jin, F. Zhang, Q. Sun, H. Bui, M. Parashar, H. Yu, S. Klasky, N. Podhorszki, and H. Abbasi. Using cross-layer adaptations for dynamic data management in large scale coupled scientific workflows. In *SC '13*.

[17] C. Jones, K.-L. Ma, A. Sanderson, and L. R. M. Jr. Visual interrogation of gyrokinetic particle simulations. *J. Phys.: Conf. Ser.*, 78(012033):6, 2007.

[18] S. M. Kelly, J. P. V. Dyke, and C. T. Vaughan. Catamount N-Way (CNW): An implementation of the Catamount light weight kernel supporting N-cores version 2.0. Technical Report SAND2008-4039P, Sandia National Laboratories, June 2008.

[19] B. Kocoloski and J. Lange. Better than native: using virtualization to improve compute node performance. In *Proceedings of the 2nd International Workshop on Runtime and Operating Systems for Supercomputers*, ROSS '12, pages 8:1–8:8, New York, NY, USA, 2012. ACM.

[20] B. Kocoloski and J. Lange. Efficient Shared Memory for Composed Applications on Multi-OS/R Exascale Systems. In *Proceedings of the 24th International ACM Symposium on High Performance Distributed Computing (HPDC)*, 2015. To Appear.

[21] J. Lange, K. Pedretti, P. Dinda, P. Bridges, C. Bae, P. Soltero, and A. Merritt. Minimal-Overhead Virtualization of a Large Scale Supercomputer. In *Proceedings of the 7th ACM International Conference on Virtual Execution Environments (VEE)*, 2011.

[22] J. Lange, K. Pedretti, T. Hudson, P. Dinda, Z. Cui, L. Xia, P. Bridges, A. Gocke, S. Jaconette, M. Levenhagen, and R. Brightwell. Palacios and Kitten: New High Performance Operating Systems for Scalable Virtualized and Native Supercomputing. In *Proceedings of the 24th IEEE International Parallel and Distributed Processing Symposium (IPDPS)*, 2010.

[23] J. Lofstead, F. Zheng, S. Klasky, and K. Schwan. Input/output apis and data organization for high performance scientific computing. In *In Proceedings of PDSW 2008 at Supercomputing 2008*, 2008.

[24] J. Lofstead, F. Zheng, S. Klasky, and K. Schwan. Adaptable, metadata rich io methods for portable high performance io. In *In IPDPS'09, May 25-29, Rome, Italy*, 2009.

[25] National Center for Atmospheric Research. CESM: Community earth system model. `http://www2.cesm.ucar.edu/`, 2015.

[26] L. Oliker, J. Carter, michael Wehner, A. Canning, S. Ethier, A. Mirin, G. Bala, D. parks, patrick Worley Shigemune Kitawaki, and Y. Tsuda. Leading computational methods on scalar and vector hec platforms. In *Proceedings of SuperComputing 2005*, 2005.

[27] D. Otstott, N. Evans, L. Ionkov, M. Zhao, and M. Lang. Enabling composite applications through an asynchronous shared memory interface. In *Big Data (Big Data), 2014 IEEE International Conference on*, pages 219–224. IEEE, 2014.

[28] J. Ouyang, B. Kocoloski, J. Lange, and K. Pedretti. Achieving Performance Isolation with Lightweight Co-Kernels. In *Proceedings of the 24th International ACM Symposium on High Performance Distributed Computing (HPDC)*, 2015. To Appear.

[29] S. Plimpton. Fast parallel algorithms for short-range molecular dynamics. *Journal of Computational Physics*, 117(1):1–19, 1995. `http://lammps.sandia.gov/index.html`.

[30] R. Schmidt, K. Belcourt, R. Hooper, R. Pawlowski, K. Clarno, S. Simunovic, S. Slattery, J. Turner, and S. Palmtag. An approach for coupled-code multiphysics core simulations from a common input. *Annals of Nuclear Energy*, 2014. `http://dx.doi.org/10.1016/j.anucene.2014.11.015`.

[31] R. R. Sinha and M. Winslett. Multi-resolution bitmap indexes for scientific data. *ACM Trans. Database Syst.*, 32(3):16, 2007.

[32] A. Strachan, S. Mahadevan, V. Hombal, and L. Sun. Functional derivatives for uncertainty quantification and error estimation and reduction via optimal high-fidelity simulations. *Modelling Simul. Mater. Sci. Eng.*, 21(6):065009, Sept. 2013.

[33] M. O. Tsugawa and J. A. B. Fortes. A virtual network (vine) architecture for grid computing. In *20th International Parallel and Distributed Processing Symposium (IPDPS)*, April 2006.

[34] J. Wang, K.-L. Wright, and K. Gopalan. Xenloop: A transparent high performance Inter-VM network loop back. *Journal of Cluster Computing – Special Issue on High Perform ance Distributed Computing (HPDC)*, 12(2):141–152, 2009.

[35] M. Wolf, Z. Cai, W. Huang, and K. Schwan. Smartpointers: Personalized scientific data portals in your hand. In *Proceedings of SuperComputing 2002*, Nov 2002. `http://www.sc-2002.org/paperspdfs/pap.pap304.pdf`.

[36] M. Woodacre, D. Robb, D. Roe, and K. Feind. The SGI Altix 3000 Global Shared Memory Architecture. Technical report, Silicon Graphics International Corporation, 2003.

[37] F. Zheng, H. Abbasi, C. Docan, J. Lofstead, Q. Liu, S. Klasky, M. Parashar, N. Podhorszki, K. Schwan, and M. Wolf. Predata - preparatory data analytics on peta-scale machines. In *Proceedings of the 24th IEEE International Parallel and Distributed Processing Symposium (IPDPS 2010)*, April 2010.

[38] F. Zheng, H. Yu, C. Hantas, M. Wolf, G. Eisenhauer, K. Schwan, H. Abbasi, and S. Klasky. GoldRush: Resource Efficient in Situ Scientific Data Analytics Using Fine-grained Interference Aware Execution. In *Proceedings of the 26th International Conference on High Performance Computing, Networking, Storage and Analysis (SC)*, 2013.

[39] F. Zheng, H. Zou, G. Eisenhauer, K. Schwan, M. Wolf, J. Dayal, T.-A. Nguyen, J. Cao, H. Abbasi, S. Klasky, N. Podhorszki, and H. Yu. Flexio: I/o middleware for location-flexible scientific data analytics. In *Parallel Distributed Processing (IPDPS), 2013 IEEE 27th International Symposium on*, pages 320–331, May 2013.