Hobbes: Composition and Virtualization as the Foundations of an Extreme-scale OS/R

Ron Brightwell, Ron Oldfield
Sandia National Laboratories
Center for Computing Research
Albuquerque, NM
{rbbrigh,raoldfi}@sandia.gov

Arthur B. Maccabe, David E. Bernholdt
Oak Ridge National Laboratory
Computer Science and Mathematics Divsiion
Oak Ridge, TN
{maccabe,bernholdtde}@ornl.gov

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

Keywords

operating system, supercomputing, virtualization, application composition

ABSTRACT

This paper describes our vision for Hobbes, an operating system and runtime (OS/R) framework for extreme-scale systems. The Hobbes design explicitly supports application composition, which is emerging as a key approach for applications to address scalability and power concerns anticipated with coming extreme-scale architectures. We make use of virtualization technologies to provide the flexibility to support requirements of application components for different node-level operating systems and runtimes, as well as different mappings of the components onto the hardware. We describe the architecture of the Hobbes OS/R, how we will address the cross-cutting concerns of power/energy, scheduling of massive levels of parallelism, and resilience. We also outline how the "users" of the OS/R (programming models, applications, and tools) influence the design.

1. INTRODUCTION

Application composition is a critical capability that will be the foundation of the way extreme-scale systems must be used in the future. The high-performance computing (HPC) community is already seeing the need for tighter integration of modeling and simulation with advanced analysis, and ad hoc solutions for coupling multiple simulations as well as integrating simulation and analysis are being developed and deployed. These ad hoc approaches are often hindered by system interfaces not designed to provide the full semantic capability required, making it difficult to deliver scalable high-performance implementations.

A recent workshop report [4] published by the U. S. Department of Energy (DOE) describes the challenges facing OS/Rs for future extreme-scale scientific computing systems. Many of these challenges, such as the need for increased reliability and the desire to reduce power and energy use, are largely driven by limitations in hardware technology, and the computer architecture community is vigorously pursuing potential approaches and solutions. While the OS/R is an important component in exploiting hardware-based solutions, addressing near-term hardware limitations without considering application composition will lead to incomplete solutions. A more effective approach is to consider these challenges in the context of application composition and to allow for integration of the hardware or algorithmic solutions required to address them.

This paper outlines our vision for an OS/R that enables application composition, and leverages a lightweight virtualization capability to provide a system software infrastructure for future extreme-scale scientific computing platforms. The rest of this paper is organized as follows. Sec. 2 provides background on the important factors that influence OS/R design and how each of these factors influences our approach to application composition. Sec. 3 describes the scenarios that motivate the need for more advanced support for application composition on current and future extreme-scale systems. Sec. 4–7 describe our approach, based on lightweight virtualization, the architecture of the OS/R framework, the cross-cutting concerns we consider in the design, and how we incorporate the needs of the various "users" of the OS/R stack into our design. Finally, we summarize the key ideas and contributions of this paper in Sec. 8.

2. BACKGROUND

In this section, we provide our perspective of the key considerations that impact the design and development of the OS/R, briefly describe previous work on supporting application-specific OS/R functionality, and discuss the rationale for a development process appropriate for extreme-scale systems.

2.1 Factors Influencing OS Design

Despite the rapidly changing landscape of HPC, OS/R design continues to be influenced by a small number of factors. We describe each of these factors in terms of extreme-scale parallel-computing platforms and the challenges they present, considering application composition and the perspective that the operating system needs to *enable explo-*

ration of potential solutions to these challenges, but not necessarily *dictate* solutions.

Usage Model. The usage model for HPC machines is evolving from sequences of simulation and analysis tasks, communicating via long-term storage, toward a more dynamic, "compositional" approach, where applications consist of complex combinations of coupled codes, data services, and tools. Projected limitations in the I/O system motivate the integration of simulation and analysis, coupling of complex physics codes, and development of fully-integrated application workflows [35, 27, 5]. Expected power constraints motivate the need to co-locate applications to avoid data movement wherever possible [27]. New HPC use cases for streaming and graph analytics require features of the OS/R such as global addressing, massive multithreading, and event-based processing that are not well supported on traditional HPC systems [14]. Finally, the expected resilience challenges for extreme scale drive the need for innovative solutions throughout the software stack [8].

Hardware Architecture Advancements. We are focused on the requirements of leadership-class systems that will be available in the 2020 time frame. These systems will contain a very large collection of "compute nodes" connected by a high-performance communication network. This compute infrastructure will include a monitoring and control system that provides extensive capabilities for observing the health of the system and supports "out-of-band" communication capabilities.

The compute nodes will provide a wide range of *computational resources*, including traditional, load-store processors, streaming processors, and specialized processors located near resources (e.g., in- and near-memory processors). Parts of the *memory system* will have very different performance characteristics in terms of bandwidth, latency, energy consumption, resilience, and persistence. The inter-node *communication network* will continue to be a distinguishing characteristic of leadership-class systems and the network endpoints will likely be more tightly integrated into the compute nodes. Finally, we expect that the Hardware Monitoring and Control (HMC) system will provide more opportunity for customized *monitoring and control* of critical resources, enabling customized event-detection, information routing and response.

Programming Environments and Tools. Advanced programming models, and the application-facing runtime systems (RTS) used to realize these models, present several challenges. Since the introduction of massively parallel processors (MPPs) in the 1990's, advanced programming models have emphasized methods for communication. Future programming models will increasingly explore abstractions for other critical resources. Among these, abstractions needed to support deep memory hierarchies (e.g., exposing address mapping hardware), enable use of specialized processors (e.g., shipping runtime code to near-memory or on-network interface processors), and resilience/power management (exposing the HMC infrastructure) will be critical.

Performance and debugging tools also present challenges. Performance tools need to capture, filter, and communicate observations, some of which may be difficult to collect (e.g., memory bus utilization). Debugging tools need low-level access to the resources used by an application.

External Shared Services. The most important external shared service for extreme-scale parallel machines has been the storage system, but external services for visualization and wide-area networking are also common. There is a trend for the storage system to become more tightly integrated into the compute platform to help address the bandwidth limitations of spinning media. The OS/R will need to consider the evolving requirements of this storage hierarchy, and of other external services.

Legacy Applications. Support for legacy applications, system services, protocols, and devices used to be a considerable challenge for OS/R design and implementation. Years, and often decades of work have gone into today's applications, and a new OS/R must provide a path to support them rather than throw away this investment.

2.2 Previous Approaches

Over the last twenty years, the success of distributed-memory parallel computing allowed the OS to be highly customized. When the factors influencing OS design, described above, become sufficiently stable, it becomes straightforward to constrain the OS to provide a minimal set of services and tailor resource allocation and management techniques to a subset of potential methods. For example, the lightweight kernel (LWK) approach popularized by Sandia National Laboratories in the 1990's [26] was based on providing a limited set of system services needed for performance and scalability for message-passing based applications running on a space-shared, batch-scheduled, distributed-memory parallel computer with an attached parallel file system. In contrast, a general purpose OS is designed to maximize the throughput of all of the job sharing a node rather than maximize the performance of a single job using the node. However, as high-performance computing became more general-purpose – moving from custom MPP systems to commodity clusters – using a general-purpose OS became more feasible. More recently, scalability issues such as OS noise [23] led to renewed interest in understanding the benefits of an LWK approach [11, and references therein] relative to a general-purpose OS for extreme-scale systems.

Much OS research has been focused on offering resource management policies that are optimized for a particular application, and providing the ability to have an OS specialized for the needs of a particular application [32]. An LWK can be viewed as a specific instance of this capability targeted at HPC applications. In Sandia's previous attempt at providing a configurable OS [33, and references therein], we discovered that most applications were either designed for a LWK environment or they expected the full functionality of a general-purpose OS like Linux. There was essentially no middle ground where configurable services needed to be provided or where alternative implementations of those services was desirable given this experience. Consequently, we believe that virtual machine technology is the most effective approach to providing an application-specific set of system services and coupling the OS with the application.

2.3 Addressing Extreme Scale

The other perspective driving HPC OS/R research and development is *scale up* versus *scale down.* The scale-up approach leverages a general-purpose OS like Linux and enhances it through excision or replacement of services to allow it to scale up to meet the demands of extreme scale. The alternative approach is to start with an OS designed and proven for the extreme scale and devise a method of scaling it down to smaller systems. In our experience, scaling down is a much easier problem than scaling up. Again, we believe that virtualization technology will enable an approach for meeting the demands of extreme scale and will also provide a viable environment for developing and running applications at smaller scale [16]. We believe it is much more effective to explore solutions in an unconstrained environment like an LWK, which has a relatively small code base and a limited amount of complexity. Once discovered, implementing a solution in a more general-purpose OS is more straightforward than trying to scale up approaches used at a smaller scale.

3. MOTIVATING USE CASES

Our goal is a flexible OS/R, capable of hosting existing petascale applications, and supporting new operating modes better suited to new architectures and programming models. To motivate our approach, we present a set of use cases that represent existing and developing codes important to the DOE mission.

Exploratory Analytics. The "exploratory analytics" use case represents application workflows that include a large-scale simulation that feeds an analysis or visualization code. The traditional approach performs analysis as a post-processing step, storing intermediate results to the file system. On extreme-scale systems, the sheer volume of data could make this approach impractical for many types of analysis. Instead, we are interested in workflows that execute simulation, analysis, and visualization concurrently. Examples of existing exploratory analytics applications include coupling shock physics with ParaView for fragment detection [21], a Quantum Monte Carlo (QMC) materials code coupled with a service to generate *observables* in a different coordinate system [29], and the Pixie3D magnetohydrodynamics (MHD) code coupled to PixiePlot and ParaView [24].

Streaming Analytics. In the "streaming analytics" use case, a simulation produces small data products to be consumed in an event-based control-system. The events follow a distributed data-flow model. A streaming analytics application can be instantiated as a directed graph of computing components. It does not require the full features of traditional HPC runtimes, but does require mechanisms for event routing and transfer, proper placement in a network topology, and signals for event activation. Examples include topological analysis for the S3D combustion simulation code [6], and streaming network analysis [20].

Graph Analytics. The "graph analytics" use case represents a host of problems including simple statistics gathering of marketing profiles, strong correlating methods, and knowledge creation with inferencing and deduction on large data structures such as semantic networks. Graph-analytics codes often require massive multithreading and global shared memory not supported by traditional HPC runtimes. Examples include social and economic modeling [14], as well as modeling of the power grid.

Code Coupling. The "code coupling" use case demonstrates interactions between concurrent HPC simulations. Although the coupled applications execute concurrently, they may have dramatically different requirements for programming models and runtime support. Data sharing and synchronization needs may benefit from co-locating portions of the codes on the same physical nodes. Examples include coupling the XGC1 fusion code to the reduced-model code XGC-1A to allow for longer running times [2], and the LIME multiphysics coupling environment [22].

Application Frameworks. The "application frameworks" use case focuses on approaches to coordinate execution and management of analysis results for large numbers of parallel jobs, for example, to conduct parameters studies or sensitivity analyses. This example represents the Many-Task Computing approach used by frameworks such as Falcon [25], Swift [34], and the Integrated Plasma Simulator (IPS) [9]. On current HPC platforms, frameworks make heavy use of scripting languages (e.g., Python) and communicate results through a global file system; however, we do not expect this to be a practical use of the file system at extreme scales.

4. APPROACH

This section describes our approach to providing an OS/R stack with explicit support for application composition and virtualization. This OS/R stack, Hobbes[1], is intended to be both a practical solution to support extreme-scale applications, and a vehicle for research in critical technologies required for the effective use of extreme-scale HPC systems.

Our ability to develop technologies that meet the requirements while minimizing overhead and carefully managing application isolation is critical. Our approach leverages and extends technologies developed for the Kitten LWK and the Palacios lightweight virtualization system [18] as the starting point for our work, with substantial restructuring to meet the needs of the Hobbes software stack.

4.1 Application Composition

Ad hoc composition of applications is already taking place in the HPC community, typically requiring that applications be adapted to a common runtime environment. In contrast, we propose to support all of the use cases described in the previous section with a unified *application composition framework*. This framework will support the composition of applications developed for different programming models (e.g., MPI and UPC) with potentially incompatible RTSs. Each application will have a single runtime environment and will run in an independent *enclave*(as defined in [4]). As such, application composition becomes "enclave composition."

Although most of our discussion focuses on coarse-grain composition of applications, where there is a clear separation in runtime requirements, we do not prevent finer-grain compositions, such as the desire to mix multiple threading packages within the same component (e.g., OpenMP, Threading Building Blocks [1], and pthreads). While this is fundamentally an application/runtime issue, our thread creation and scheduling mechanisms are flexible enough to support all common threading models.

One key challenge with our approach is the possibility that the applications being composed may have markedly different runtime requirements. For example, one application may have minimal runtime requirements, while another might require a full-featured OS, like Linux.

Another challenge involves the mapping of enclaves onto the compute nodes of an extreme-scale system. The most straightforward mapping would allocate independent nodes for each enclave, using the interconnection network to join

[1]Hobbes is the stuffed tiger in the Calvin and Hobbes comics created by Bill Watterson. This follows the Sandia tradition of naming its LWKs for cats.

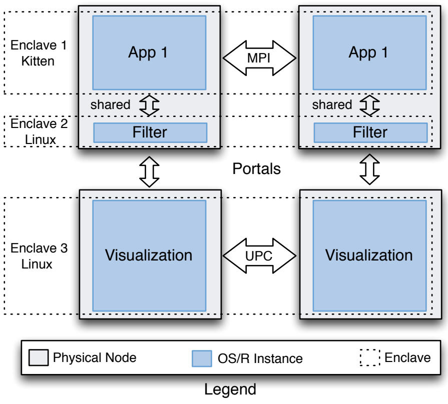


Figure 1: Composition for an "exploratory analytics" use case that uses a filter co-located with the simulation to reduce data communicated to the visualization enclave.

(compose) the enclaves (applications). While this mapping is easy to implement, performance considerations dictate that we also support mappings based on direct sharing of node-level resources. For example, in the Exploratory Analytics use case (Fig. 1), a small "filter" needs to be interposed between the application and visualization enclaves. While we will describe the use case as a standard composition of applications, our intent is that the implementation will be lightweight (e.g., a function call) and will reduce the data communicated to the visualization enclave. Ultimately, mapping decisions must be based on a careful analysis of performance trade-offs and it is our intention to maintain a separation between composition and mapping: composition is used to describe computations, while mapping defines the physical resources used to realize these computations.

It is worth noting that our approach in mapping enclaves to shared physical resources provides the opportunity to, effectively, "move code to data," thus reducing (network) data movement and thereby power consumption, as well as increasing performance. With the addition of appropriate trust, sandboxing, and resource management mechanisms, the same approach can be used to allow interactions between applications and shared services. For example, an application could "push code" to an enclave implementing a storage service (i.e., *active storage*").

4.2 Virtualization

To address the challenges of variable runtime requirements and complex application mappings, we will develop a wide range of virtualization technologies. These technologies will support the definition of "virtual compute nodes," enabling the full system virtualization that can be used to support almost any RTS that could be required. By supporting the creation of multiple virtual nodes from a single physical node, these technologies will also enable sharing of the physical resources available on a compute node.

5. HOBBES ARCHITECTURE

The main components of the Hobbes architecture (Fig. 2) include a *System-Global OS*, *Enclave OS/R*, *Node OS/R*, *Node Virtualization Layer*, and the *Global Information Bus*. In this section, we describe the role each component plays in the Hobbes architecture and the APIs needed to support application composition.

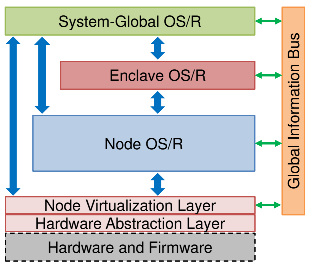


Figure 2: Schematic representation of the main components of the Hobbes architecture and their interactions via APIs (vertical arrows) and data exchange (horizontal arrows).

5.1 System-Global OS

The *System-Global OS* (SGOS) is the portion of the system responsible for scheduling, monitoring, controlling, and coordinating the resources of a single system. We pay particular consideration to how applications use shared system services, such as shared storage or visualization systems; indeed, the need to effectively use shared services is a primary motivator for our approach to application composition. In Hobbes, however, a shared service is simply a specialized, possibly long-running application whose enclave may include specialized nodes, e.g., storage or visualization nodes.

The APIs for the the SGOS include interfaces to a scheduling and resource management subsystem responsible for mapping enclaves and assemblies of enclaves onto physical resources; interfaces to hardware management and control systems that manage information about the health and productivity of associated hardware and software in the system, possibly gathered from sensors; interfaces for autonomic management [31] that support a wide range of policies for adapting resource allocations, for example based on power or resilience constraints [15, 30]; and interfaces to basic global services like authentication and authorization.

In addition to the core APIs for the SGOS, application composition requires a much richer specification of jobs than in the past. Job descriptions may include OS/R selections for enclaves, information to guide the mapping of the logical computation structure onto the physical resources, and policies to be enforced by the autonomic management systems. We will draw on related work in the cloud and grid computing communities to inform our designs [3, 17].

5.2 Enclave OS/R

An *enclave* is a partition of the system allocated to a single application or service [4]. As such, an enclave is primarily a container and a unit of coordination. The *Enclave OS/R* (EOS) as a distinct component of the OS architecture extends the enclave concept with functionality, in much the same way that an object extends a data structure with functionality. The primary reason for introducing the EOS as a distinct component of an extreme-scale OS is to allow each enclave to operate with a degree of autonomy, rather than current practice of explicit top-down control of all aspects of the system from the equivalent of the SGOS.

The APIs of the EOS provide support for enclave membership, collective operations for launching, terminating, pause, and resume enclaves; and autonomic management similar to what is provided in the SGOS. Enclave membership is a key capability of the EOS. Since we intend to support dynamic enclaves, negotiations for addition or removal of resources may be initiated by the RTS, the SGOS, or by the EOS itself, for example due to a node failure.

The composition of enclaves in order to assemble complex applications is the most important and novel feature of the EOS. The idea that, by definition, each enclave uniformly executes a single OS/R configuration also implies that each one can be tailored to the needs of the application. Composition of enclaves, then, is the selective breaking of the normal isolation between enclaves to allow direct interactions between Node OS/R instances. At the lowest level, composition of enclaves on distinct sets of nodes involves the network, while enclaves on virtual nodes co-located on the same physical nodes share hardware (e.g., memory), (Fig. 3). However, we propose to develop suitable abstractions to provide a uniform mechanism so users can decide how to map composite applications onto the hardware simply based on the performance requirements of the coupling.

5.3 Node OS/R

The *Node OS/R* (NOS) provides interfaces and abstractions to the underlying compute, memory, and network resources and also provides APIs for use by the Enclave OS/R (EOS) and System Global OS (SGOS) to support higher-level allocation and management of node-level resources.

Important features of the NOS relate to managing compute, memory, and network resources. The NOS will need to expose compute capabilities from an increasingly complex set of compute resources, for example logic layers embedded in sophisticated devices like the Hybrid Memory Cube [7]. Memory hierarchies will become much more complex and the NOS needs to provide interfaces allowing the runtime to manage this memory effectively. Finally, to maximize network efficacy and isolation, we will develop an infrastructure for *differentiated network services* that provides latency and bandwidth guarantees to traffic participants.

5.4 Node Virtualization Layer

The *Node Virtualization Layer* (NVL) provides management of virtual node instances on physical nodes and provides hardware abstractions, through the *Hardware Abstraction Layer* (HAL). The NVL, which is not called out in the workshop report, is central to our approach to composition, as well as providing a flexible environment capable of supporting the diverse Node OS/R requirements of applications.

The NVL must handle three distinct use cases: (1) the ability to support an application running with a minimal (native) node OS; (2) the ability to support full featured operating systems; and (3) the ability to create a multiplicity of "virtual nodes" to support the sharing of physical resources between applications in distinct enclaves. In previous work, we used the Kitten/Palacios [18] code base to address the first two cases. The third use case provides motivation for much of the fundamental research that will be required for Hobbes. The mechanisms developed to create multiple virtual nodes must be efficient enough to realize the full benefit of shared resources, but flexible enough to provide the required degree of isolation between enclaves.

5.5 Global Information Bus

The *Global Information Bus* (GIB) is the logical software layer that encapsulates mechanisms for sharing status information needed by other components in the OS/R. Core functionality in the GIB will support the gathering, publication and subscription of status information. This includes basic performance information (e.g., current energy consumption for each node) and information defined by a runtime (e.g., length of the unexpected message queue for MPI). Information communicated through the GIB has two distinguishing characteristics: first, the information is communicated in small parcels (perhaps a few words) and second, there is no expectation of coherence in this information (it is very unlikely that a scan through all of the nodes would result in a consistent view of the machine). The intention is to provide hints about the status of nodes that can be used in initial management decisions by the EOS, an adaptive runtime system, or tools.

In many respects, the GIB represents an abstraction of the information collection and dissemination portion of the SGOS Hardware Monitoring and Control (HMC) subsystem, which on high-end HPC systems is implemented on distinct Reliability, Availability, and Serviceability (RAS) hardware. We expect to implement the functionality of the GIB on the RAS system where possible, but also using the main high-performance network on commodity clusters that may not have a separate RAS system.

6. CROSSCUTTING AREAS

To effectively address existing and upcoming requirements of extreme-scale systems, Hobbes must support and enable research and development in a number of crosscutting areas. Perhaps the most critical of these are power and energy, scheduling, and resilience.

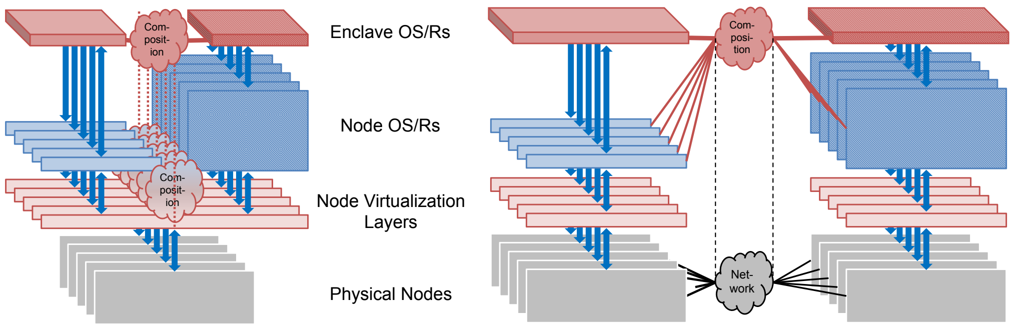
6.1 Power and Energy

Power and energy consumption will be significant constraints on next-generation systems. Accordingly, we need to ensure that Hobbes is "power aware" so that it can be a suitable tool for power management research, as well as production operation of power-managed systems.

Hobbes provide APIs for measurement and control of power throughout the system. We will leverage our recent work [13, 12, 19] to develop a broad system power API specification. The power-measurement API will support measurement at the component, node, enclave, and system granularities (at a minimum). The API will also facilitate measuring existing and new processors, memory, or network technologies. The power control API provides a mechanism to manage the power consumption of devices that expose these features. Although this type of control is not common among computing devices, recent activity to expose more controls for managing power is encouraging. For example, power clamping has recently been introduced on Intel Sandy Bridge processors [28], and similar mechanisms exist on IBM Power-7 and AMD Bulldozer processors. Our API will provide an abstraction to explicitly control these devices as well as reason about their usage.

6.2 Scheduling

Virtually every one of the APIs defined in the earlier sections will have a scheduling component. A key challenge



(a) Intra-node composition (b) Inter-node composition

Figure 3: Basic scenarios for composition of enclaves: (a) enclaves share physical nodes, and (b) enclaves are on distinct sets of physical nodes. The enclaves are illustrated as running different node OS/Rs (i.e., lightweight and general-purpose). The System-Global OS and Global Information Bus components are not shown.

is the design of bi-directional interfaces to support coordination of scheduling decisions between multiple layers in the OS/R stack. These interfaces must support scheduling-related dialog between an upper-level component that establishes policy and a lower-level component that provides the mechanisms needed to implement the policy. As an example, the SGOS establishes policies for the full machine; however, the mechanisms needed to affect these policies will be provided by the EOS, the NOS and possibly the NVL. The needed dialog is frequently complicated by the fact that the lower-level component is the first to be invoked when a scheduling decision is needed.

6.3 Resilience

The OS/R must handle faults in all of its layers, masking faults when appropriate, and providing sufficient information and mechanisms to other OS layers and applications to handle unmasked faults. We expect Hobbes to be a vehicle for research in resilience mechanisms to handle faults in different layers of the software stack and to understand fault sensitivity and coverage.

One area of particular interest is the development of mechanisms that identify critical OS/R data structures (those most susceptible to failures and critical for stability) and creating a methodology for resilient OS/R data structures. For example, previous analysis [10] of dynamic memory profiles of Kitten and Linux indicates ample opportunities for enhancements, such as using hashes, redundant data, and self-checks and repairs, of OS/R data structures for specific allocation regions (e.g., memory/process management, kernel/shared memory), which we will incorporate into the Hobbes OS/R components as appropriate.

7. USER SUPPORT

To effectively address the needs of the HPC community, Hobbes must support the entire set of HPC system "users", including support for developers of existing and experimental programming models, application developers, and performance and debugging tool developers. The following describes the key interfaces in our design.

7.1 Programming Model Support

Programming model support is a critical enabling feature of the proposed OS/R infrastructure. Hobbes provides both low level OS mechanisms and global privileged services for system management. The low level mechanisms need to enable both RTS and OS support, while global services provide for functionality outside the purview of applications. Our main goal is to enable better system management by providing a path for the flow of information from the RTS to the NOS and SGOS.

7.2 Application Support

Much of the required work in the APIs and cross-cutting areas is motivated by the requirements and challenges associated with integrating analytics and simulation. In particular, we expect support for composition of applications with different OS/R requirements to have a dramatic impact on development, deployment, and usability of integrated analytics codes on extreme-scale systems. For example, consider recent work for the National Nuclear Security Administration Advanced Simulation and Computing program (NNSA/ASC) to explore integrated simulation analysis. As part of that project, an SNL team has developed both "in situ" and "in transit" analysis techniques that use ParaView to detect material fragments from simulation results of the CTH shock physics code [27]. This work revealed a number of weaknesses in existing HPC systems that could be dramatically improved by technologies included in the Hobbes design. In particular, this type of application requires, at a minimum, portable mechanisms for inter/intra-enclave communication and coordinated scheduling. Advanced features like dynamic allocation, contraction, and expansion of enclaves; control over placement of data and code; and data-movement scheduling would significantly expand the capability and utility of this type of approach.

7.3 Tools Support

As with programming models and applications, tools define requirements for the components of Hobbes. Our goal is to ensure that Hobbes includes technologies to enable scalable and efficacious debugging, performance, and system administration tools. Traditional OS/Rs have sufficient support for intra-enclave tools; however, we must also support tools that work with multiple enclaves. For instance, we must provide the support for a debugger to inspect and control processes of a software composition running in multiple enclaves, so that the user can debug interactions between the parts running in the various enclaves.

8. SUMMARY

We have described our vision for the Hobbes extreme-scale OS/R framework. The design is motivated by anticipated changes in the architecture of extreme-scale systems, as well as by trends in applications as they respond to the same hardware changes. In particular, we incorporate explicit support for application composition into the OS/R design. We rely on virtualization technologies to provide the flexibility to support different node-level OS/R requirements that different application components may have, as well as to allow co-location of components on the same physical nodes, for performance reasons. The design addresses rising concerns for extreme-scale computing, including power and energy management, dealing with massive ("billion-way") parallelism, and resilience. We also incorporate requirements arising not only from the applications, but also from current and novel programming models, and from tools, such as debuggers and performance analyzers.

9. ACKNOWLEDGMENTS

We would like to thank all of the members of the Hobbes team for their contributions to filling in the details and refining the vision outlined here.

Sandia National Laboratories is a multi-program laboratory managed and operated by Sandia Corporation, a wholly owned subsidiary of Lockheed Martin Corporation, for the U.S. Department of Energy's National Nuclear Security Administration under contract DE-AC04-94AL85000.

Oak Ridge National Laboratory is managed by UT-Battelle, LLC for the U.S. Department of Energy under Contract No. DE-AC05-00OR22725.

10. ADDITIONAL AUTHORS

Additional authors: Eric Brewer (University of California at Berkeley), Patrick Bridges (University of New Mexico), Peter Dinda (Northwestern University), Jack Dongarra (University of Tennessee) Costin Iancu (Lawrence Berkeley National Laboratory), Michael Lang (Los Alamos National Laboratory), Jack Lange (University of Pittsburgh), David Lowenthal (University of Arizona), Frank Mueller (North Carolina State University), Karsten Schwan (Georgia Institute of Technology), Thomas Sterling (Indiana University), and Patricia Teller (University of Texas-El Paso).

11. REFERENCES

[1] Intel threading building blocks (Intel TBB 4.1 update 2. `http://threadingbuildingblocks.org/`.

[2] "Center for Edge Physics Simulation". `http://http://epsi.pppl.gov/`, 2013.

[3] A. Anjomshoaa, F. Brisard, M. Drescher, D. Fellows, A. Ly, S. McGough, D. Pulsipher, and A. Savva. Job submission description lanuguage (JSDL) specification, version 1.0. Technical Report GFD-R.056, Global Frid Forum, 7 Nov 2005. `www.ggf.org/documents/GFD.56.pdf`.

[4] P. Beckman, R. Brightwell, B. R. de Supinski, M. Gokhale, S. Hofmeyr, S. Krishnamoorthy, M. Lang, B. Maccabe, J. Shalf, and M. Snir. Exascale operating systems and runtime software report. Technical report, U. S. Department of Energy, December 28 2012. `http://science.energy.gov/~/media/ascr/pdf/research/cs/Exascale%20Workshop/ExaOSR-Report-Final.pdf`.

[5] J. C. Bennett, H. Abbasi, P.-T. Bremer, R. W. Grout, A. Gyulassy, T. Jin, S. Klasky, H. Kolla, M. Parashar, V. Pascucci, P. Pébay, D. Thompson, H. Yu, F. Zhang, and J. Chen. Combining in-situ and in-transit processing to enable extreme-scale scientific analysis. In *Proceedings of SC2012: High Performance Networking and Computing*, Salt Lake City, UT, Nov. 2012. ACM Press.

[6] J. H. Chen, A. Choudhary, B. de Supinski, M. DeVries, E. R. Hawkes, S. Klasky, W. K. Liao, K. L. Ma, J. Mellor-Crummey, N. Podhorszki, R. Sankaran, S. Shende, and C. S. Yoo. Terascale direct numerical simulations of turbulent combustion using S3D. *Computational Science & Discovery*, 2(1):31pp, 2009.

[7] H. M. C. Consortium. `http://www.hybridmemorycube.org`.

[8] N. DeBardeleben, J. Laros, J. Daly, S. Scott, C. Engelmann, and B. Harrod. High-end computing resilience: Analysis of issues facing the HEC community and path-forward for research and development. Technical Report LA-UR-10-00030, LANL, SNL, DoD, ORNL, DARPA, January 2010.

[9] W. Elwasif, D. E. Bernholdt, A. G. Shet, S. S. Foley, R. Bramley, D. B. Batchelor, and L. A. Berry. The design and implementation of the SWIM Integrated Plasma Simulator. In *Parallel, Distributed and Network-Based Processing (PDP), 2010 18th Euromicro International Conference on*, pages 419–427, 2010.

[10] K. Ferreira, K. Pedretti, P. G. Bridges, R. Brightwell, D. Fiala, and F. Mueller. Evaluating operating system vulnerability to memory errors. In *Workshop on Runtime and Operating Systems for Supercomputers*, June 2012.

[11] K. B. Ferreira, P. G. Bridges, R. Brightwell, and K. Pedretti. Impact of system design parameters on application noise sensitivity. In *Proceedings of the 2010 IEEE International Conference on Cluster Computing*, September 2010.

[12] V. Gupta, P. Brett, D. Koufaty, D. Reddy, S. Hahn, K. Schwan, and G. Srinivasa. Heteromates: Providing high dynamic power range on client devices using heterogeneous core groups. In *2012 International Green Computing Conference (IGCC)*, volume 0, pages 1–10, Los Alamitos, CA, USA, 2012. IEEE Computer Society.

[13] V. Gupta, R. Knauerhase, et al. Attaining system performance points: Revisiting the end-to-end argument in system design for heterogeneous many-core systems. *SIGOPS Operating System Review*, 2011.

[14] B. Hendrickson and J. Berry. Graph analysis with high-performance computing. *Computing in Science and Engineering*, 10(2):14–19, March/April 2008.

[15] H. Hoffmann, J. Eastep, M. D. Santambrogio, J. E. Miller, and A. Agarwal. Application heartbeats: a

generic interface for specifying program performance and goals in autonomous computing environments. In *Proceedings of the 7th international conference on Autonomic computing*, ICAC '10, pages 79–88, New York, NY, USA, 2010. ACM.

[16] 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.

[17] L. Lamers et al. Open virtualization format specification. version 2.0.0. DMTF Standard DSP0243, Distributed Management Task Force, Dec 13 2012. `http://www.dmtf.org/sites/default/files/standards/documents/DSP0243_2.0.0.pdf`.

[18] 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*, April 2010.

[19] J. H. Laros, III, K. T. Pedretti, S. M. Kelly, W. Shu, and C. T. Vaughan. Energy based performance tuning for large scale high performance computing systems. In *Proceedings of the 2012 Symposium on High Performance Computing*, HPC '12, pages 6:1–6:10, San Diego, CA, USA, 2012. Society for Computer Simulation International.

[20] J. Lofstead, R. A. Oldfield, and T. H. Kordenbrock. Experiences applying data staging technology in unconventional ways. In *13th IEEE/ACM International Symposium on Cluster, Cloud and Grid Computing (CCGrid)*, Delft, The Netherlands, May 2013. IEEE/ACM.

[21] K. Moreland, R. Oldfield, P. Marion, S. Joudain, N. Podhorszki, V. Vishwanath, N. Fabian, C. Docan, M. Parashar, M. Hereld, M. E. Papka, and S. Klasky. Examples of in transit visualization. In *Proc. of PDAC 2011 : 2nd International Workshop on Petascale Data Analytics: Challenges and Opportunities*, Seattle, WA, Nov. 2011.

[22] R. Pawlowski, R. Bartlett, N. Belcourt, R. Hooper, and R. Schmidt. A theory manual for multi-physics code coupling in LIME. Technical Report SAND2011-2195, Sandia National Laboratories, March 2011.

[23] F. Petrini, D. Kerbyson, and S. Pakin. The case of the missing supercomputer performance: Achieving optimal performance on the 8,192 processors of ASCI Q. In *Proceedings of the ACM/IEEE International Conference on High-Performance Computing, Networking, and Storage (SC)*, 2003.

[24] N. Podhorszki, S. Klasky, Q. Liu, C. Docan, M. Parashar, H. Abbasi, J. Lofstead, K. Schwan, M. Wolf, F. Zheng, et al. Plasma fusion code coupling using scalable I/O services and scientific workflows. In *Proceedings of the 4th Workshop on Workflows in Support of Large-Scale Science*, page 8. ACM, 2009.

[25] I. Raicu, Y. Zhao, C. Dumitrescu, I. Foster, and M. Wilde. Falkon: a fast and light-weight task execution framework. In *Proceedings of the 2007 ACM/IEEE conference on Supercomputing*, number 43, Reno, NV, November 2007. ACM Press.

[26] R. Riesen, R. Brightwell, P. G. Bridges, T. Hudson, A. B. Maccabe, P. M. Widener, and K. Ferreira. Designing and implementing lightweight kernels for capability computing. *Concurrency and Computation: Practice and Experience*, 21(6):793–817, April 2009.

[27] D. Rogers, K. Moreland, R. Oldfield, and N. Fabian. Data co-processing for extreme scale analysis. Technical Report SAND2013-XXXX, Sandia National Laboratories, March 2013. To appear.

[28] E. Rotem, A. Naveh, D. Rajwan, A. Ananthakrishnan, and E. Weissmann. Power management architecture of the second generation Intel Core microarchitecture formerly codenamed Sandy Bridge. In *Hot Chips: A Symposium on High Performance Chips*, August 2011.

[29] A. Sayed and H. El-Shishiny. Computational experience with nano-material science quantum monte carlo modeling on BlueGene/L. In *MEMS, NANO, and Smart Systems (ICMENS), 2009 Fifth International Conference on*, pages 213–217. IEEE, 2009.

[30] F. Sironi, D. B. Bartolini, S. Campanoni, F. Cancare, H. Hoffmann, D. Sciuto, and M. D. Santambrogio. Metronome: operating system level performance management via self-adaptive computing. In *Proceedings of the 49th Annual Design Automation Conference*, DAC '12, pages 856–865, New York, NY, USA, 2012. ACM.

[31] R. Sterritt, M. Parashar, H. Tianfield, and R. Unland. A concise introduction to autonomic computing. *Advanced Engineering Informatics*, 19(3):181–187, July 2005.

[32] J.-C. Tournier. A survey of configurable operating systems. Technical Report TR-CS-2005-43, University of New Mexico, Computer Science Department, 2005.

[33] J.-C. Tournier, P. Bridges, A. B. Maccabe, P. Widener, Z. Abudayyeh, R. Brightwell, R. Riesen, and T. Hudson. Towards a framework for dedicated operating systems development in high-end computing systems. *ACM SIGOPS Operating Systems Review*, 40(2):16–21, April 2006.

[34] Y. Zhao, M. Hategan, B. Clifford, I. Foster, G. von Laszewski, V. Nefedova, I. Raicu, T. Stef-Praun, and M. Wilde. Swift: Fast, reliable, loosely coupled parallel computation. In *2007 IEEE Congress on Services*, pages 199–206, July 2007.

[35] F. Zheng, H. Abbasi, C. Docan, J. Lofstead, S. Klasky, Q. Liu, M. Parashar, N. Podhorszki, K. Schwan, and M. Wolf. PreDatA - preparatory data analytics on Peta-Scale machines. In *In Proceedings of 24th IEEE International Parallel and Distributed Processing Symposium, April, Atlanta, Georgia*, 2010.