Exascale Operating Systems and Runtime Software Report

DECEMBER 28, 2012

EXASCALE OS/R TECHNICAL COUNCIL

Pete Beckman, ANL (co-chair)
Ron Brightwell, SNL (co-chair)
Bronis R. de Supinski, LLNL
Maya Gokhale, LLNL
Steven Hofmeyr, LBNL
Sriram Krishnamoorthy, PNNL
Mike Lang, LANL
Barney Maccabe, ORNL
John Shalf, LBNL
Marc Snir, ANL

PROGRAM MANAGERS

Bill Harrod (DOE/ASCR)
Thuc Hoang (NNSA/ASC)
Sonia R. Sachs (DOE/ASCR-CS)

U.S. DEPARTMENT OF ENERGY
Office of Science

Sponsored by the U.S. Department of Energy, Office of Science, Office of Advanced Scientific Computing Research

U.S. Department of Energy
Exascale Operating System and Runtime Software Report

November 1, 2012

Exascale OS/R Technical Council
Pete Beckman, ANL (co-chair)
Ron Brightwell, SNL (co-chair)
Bronis R. de Supinski, LLNL
Maya Gokhale, LLNL
Steven Hofmeyr, LBNL
Sriram Krishnamoorthy, PNNL
Mike Lang, LANL
Barney Maccabe, ORNL
John Shalf, LBNL
Marc Snir, ANL

Program Managers
Bill Harrod (DOE/ASCR)
Thuc Hoang (NNSA/ASC)
Sonia R. Sachs (DOE/ASCR)

The Exascale Operating Systems and Runtime (OS/R) Software Technical Council was convened by the U.S. Department of Energy (DOE) Office of Science and National Nuclear Security Administration (NNSA) to develop a research agenda and plan to address the OS/R challenges associated with future extreme-scale systems.

The technical council met from March 2012 through October 2012, concluding with an open workshop held October 3–4 in Washington, DC. Their charter comprised the following tasks:

- Summarize the challenges of exascale OS/R.
- Assess the impact on exascale OS/R software requirements from facilities and production support, applications and programming models, and hardware architectures.
- Describe a model for interaction between DOE-funded exascale OS/R research to vendors building supported products. The model must allow proprietary innovation, encourage APIs to allow interoperability and portability, and define a minimum set of requirements for functionality.
- Identify promising approaches to addressing challenges and requirements by engaging the HPC research community and by drawing on novel ideas from other related areas such as embedded computing and cloud computing.
- Articulate dependencies, conflicting requirements, and priorities within the OS/R research agenda.
- Submit findings to the DOE Office of Science and NNSA in a report.

Contents

Executive Summary

U.S. Department of Energy (DOE) workshops and reports have identified four key exascale challenges: dramatically improving power efficiency; improving resilience in the presence of increasing faults; enabling efficient data movement across deepening memory hierarchies and new storage technologies; and managing dramatically increased parallelism, especially at the node level. Software solutions that address these challenges must also improve programmability, expanding the community of computational scientists who can use leadership-class platforms. To address these challenges, DOE must develop new techniques, novel designs, and advanced software architectures for next-generation exascale software infrastructure. In this report, we discuss challenges and approaches to exascale operating system and runtime (OS/R) software.

The Exascae Operating Systems and Runtime (OS/R) Software Technical Council canvassed hardware architects, application programmers, parallel tool developers, DOE high-performance computing (HPC) facilities, and the vendors that sell and support integrated platforms. After considering the collective requirements of these constituents and examining the future research challenges and current software gaps, the council recommends that DOE invest in targeted advanced computer science research and in specific coordinating activities to enable the effective and successful development of anticipated exascale systems.

Computer Science Research Areas

- *Lightweight message and thread management:* In order to hide latency and support dynamic programming environments, low-level message handling and lightweight thread activation must be co-optimized. New techniques are needed to handle lightweight and resilient message layers; scalable message-driven thread activation and fine-grained active messages; global address spaces; extremely large thread counts; buffer management, collective operations, and fast parallel reductions; thread scheduling and placement; and improved quality-of-service (QoS) and prioritization.
- *Holistic power management:* Extreme-scale systems will manage power and energy as a first-class resource across all layers of software as a crosscutting concern. Novel techniques are needed for whole-system monitoring and dynamic optimization; trading of energy for resilience or time to solution; power-aware scheduling and usage forecasts; goal-based feedback and control strategies; coscheduling; and adaptive power management of storage, computing, and bandwidth.
- *Tools*: Extreme-scale OS/Rs need low-level mechanisms in order to support dynamic adaptation, performance and power optimization, and debugging. New concepts and approaches are needed to enable extremely lightweight performance data collection; thread and task-based performance attribution and debugging controls; low-overhead asynchronous event handling for shared-control tools; whole-system collection and synthesis; and feedback loops to enable autonomic real-time performance and power tuning.
- *Resilience*: Extreme-scale OS/Rs must support scalable mechanisms to predict, detect, inform, and isolate faults at all levels in the system. Therefore, resilience is a crosscutting concern. The OS/R must be resilient and support an array of low-level services to enable resilience in other software components, from the HPC application to the storage system. Innovative concepts to support multilevel, pluggable collection and response services are needed.
- *OS/R architecture*: Extreme-scale systems need agile and dynamic node OS/Rs. New designs are needed for the node OS to support heterogeneous multicore, processor-in-memory, and

HPC-customized hardware; I/O forwarding; autonomic fault response mechanisms; dynamic goal-oriented performance tuning; QoS management across thread groups, I/O, and messaging; support for fine-grained work tasks; and efficient mechanisms to support coexecution of compute and in situ analysis.

- *Memory*: Deep hierarchies, fixed power budgets, in situ analysis, and several levels of solid-state memory will dramatically change memory management, data movement, and caching in extreme-scale OS/Rs. Clearly needed are novel designs for lightweight structures to support allocating, managing, moving, and placing objects in memory; methods to dynamically adapt thread affinity; techniques to manage memory reliability; and mechanisms for sharing and protecting data across colocated, coscheduled processes.
- *Global OS/R*: Extreme-scale platforms must be run as whole systems, managing dynamic resources with a global view. New concepts and implementations are needed to enable collective tuning of dynamic groups of interacting resources; scalable infrastructure for collecting, analyzing, and responding to whole-system data such as fault events, power consumption, and performance; reusable and scalable publish/subscribe infrastructures; distributed and resilient RAS (reliability, availability, and serviceability) subsystems; feedback loops for tuning and optimization; and dynamic power management.

Coordinating Activities

- *Software stack coordination*: The DOE X-Stack research projects are working broadly on research software to enable exascale computing. Some of those projects include efforts on upper-level run-time systems to support programming models, advanced I/O, fault tolerance, etc. DOE OS/R research must be aware of these projects and collaborate where appropriate on X-stack upper-level run-time layers that could impact the features or designs of low-level OS/R components.
- *Codesign coordination*: At the heart of a large system is the OS/R software. It both manages the low-level hardware and provides interfaces for upper levels of software. OS/R features both influence and are influenced by novel hardware features and new programming models. DOE supports an active codesign community; partners include key DOE applications and the hardware companies. Exascale OS/R software research must be tightly integrated within the broader DOE plans for developing and deploying exascale platforms and scientific applications. The OS/R Technical Council recommends that research projects enthusiastically participate in codesign and describe their plans accordingly.
- *Vendor integration*: Exascale OS/R research projects must articulate a clear path for integrating research results into future extreme-scale systems supported by the HPC industry. In order to facilitate collaboration, open source is a vital part of this research effort; projects should include their plans for publishing and sharing code and potential processes for moving reference implementations into stable, production-quality codes.
- *Ecosystem*: For extreme0scale computation to remain successful, it must be supported by a broad ecosystem. Common APIs promote portability and future standards efforts. Exascale OS/R research projects should include plans for identifying, designing, and sharing common APIs where appropriate. APIs could include control interfaces for power, messaging, scheduling, memory, and threads as well as higher-level global OS/R features such as creation of node collections, dynamic resizing of resources, and tool interfaces. Some high-quality software artifacts that implement parts of the OS/R stack could be transitioned directly to HPC facilities

for deployment and support. Research projects should articulate how resulting software fits within the broader ecosystem.

- *Testbeds*: To enable researchers to explore the performance and scalability of novel OS/Rs on future hardware and large-scale systems, DOE will provide access to moderate-scale experimental testbeds that support remote superuser (root) access, building of kernels and software tools from scratch, and access to attached power-measuring devices. In addition to these testbeds, exascale OS/R research projects should pursue compelling INCITE or ASCR Leadership Computing Challenge proposals that support extreme-scale testing and performance measurements.

1 Introduction

Since 2008, the Department of Energy (DOE) has convened several workshops to address the significant challenges facing extreme-scale computing in the coming decade. Three orders of magnitude performance increase beyond current petascale systems is needed in order to continue pushing the frontiers of scientific discovery by using modeling and simulation; but fundamental issues that threaten the feasibility of reaching the level of computational capability. Through a series of workshops and subsequent reports, four key exascale challenges were identified: dramatically improving power efficiency; improving resilience in the presence of increasing faults; achieving efficient data movement across deepening memory hierarchies and new storage technologies; and managing dramatically increased parallelism, especially at the node level. Software solutions that address these challenges must also improve programmability, expanding the community of computational scientists who can use leadership-class platforms.

While such technology changes have been encountered and overcome in the past, the field of high-performance computing (HPC) has reached a level of maturity and has become firmly established as a critical capability in many science domains. This dependence on computation and the continued growth of HPC systems, applications, and users add a new dimension to the problem. Solutions must also consider other important aspects, such as legacy applications, programmability, and the desire to continue to grow the community of computational scientists who can use leadership-class platforms effectively.

As part of the ongoing DOE effort to identify, characterize, and address extreme-scale computing challenges, in early 2012 the Office of Advanced Scientific Computing Research (ASCR) within DOE and the Advanced Simulation and Computing (ASC) program within the National Nuclear Security Administration (NNSA) convened a Technical Council (TC) composed of leading researchers from several DOE national laboratories, with the aim of developing a research agenda and a plan to address the operating system and runtime (OS/R) software challenges associated with extreme-scale scientific computing systems. As the intermediary between applications and the underlying hardware, the OS/R plays a critical role in determining the effectiveness and usability of a computing system. Because the OS/R can insulate applications from many of the complexities of the hardware and machine architecture, it is highly desirable to focus initially on this layer of software in order to better understand the scope of the challenges, promising new approaches, and potential solutions. For several months, the OS/R TC met with key stakeholders, including hardware architects, application programmers, parallel tool developers, DOE HPC facility representatives, and HPC vendors that sell and support integrated platforms, in order to examine in depth the future research challenges and current system software gaps. This report details the organization, activities, findings, and recommendations of the OS/R TC.

1.1 Charter and Goal

The charter of the OS/R Technical Council comprises the following tasks:

- Summarize the challenges of exascale OS/R.
- Assess the impact on exascale OS/R software requirements from facilities and production support, applications and programming models, and hardware architectures.
- Describe a model for interaction between DOE-funded exascale OS/R research to vendors building supported products. The model must allow proprietary innovation, encourage APIs to allow interoperability and portability, and define a minimum set of requirements for functionality.

- Identify promising approaches to addressing challenges and requirements by engaging the HPC research community and by drawing on novel approaches from other related areas such as embedded computing and cloud computing.
- Articulate dependencies, conflicting requirements, and priorities within the OS/R research agenda.
- Submit findings to the DOE Office of Science and NNSA in a report.

1.1.1 History and Timeline

The OS/R TC was formed in early February 2012 at the request of Bill Harrod, director of the ASCR Division of Computational Science Research and Partnerships. The initial meeting was held on March 21, 2012, in Washington, D.C., where the organization and charter of the TC was discussed and established. The TC established a plan for monthly meetings in person or via video teleconferencing and had initial discussions about requirements, potential approaches, and identification of important stakeholders. A highly desirable goal was to engage the OS/R research community in a workshop in the October 2012 timeframe that would enable ASCR to develop a research program and subsequent funding opportunity announcement in the area of operating systems and runtime systems for extreme-scale scientific computing systems.

1.1.2 Membership

The OS/R TC comprises the following individuals:

Pete Beckman, Argonne National Laboratory
Ron Brightwell, Sandia National Laboratories
Maya Gokhale, Lawrence Livermore National Laboratory
Steven Hofmeyr, Lawrence Berkeley National Laboratory
Sriram Krishnamoorthy, Pacific Northwest National Laboratory
Mike Lang, Los Alamos National Laboratory
Arthur B. "Barney" Maccabe, Oak Ridge National Laboratory
John Shalf, Lawrence Berkeley National Laboratory
Marc Snir, Argonne National Laboratory
Bronis de Supinski, Lawrence Livermore National Laboratory

In addition, the following DOE staff attended one or more of the OS/R TC meetings:

Jim Ang, ASC
Richard Carlson, ASCR
Bill Harrod, ASCR
Thuc Hoang, ASC
Steven Lee, ASCR
Robert Lindsay, ASCR
Lucy Nowell, ASCR
Karen Pao, ASCR
Sonia Sachs, ASCR

1.1.3 Meetings

March 21–22, 2012 – Washington, D.C. – Initial meeting
April 19, 2012 – Portland, OR – DOE/NNSA Exascale Research Conference and Research Planning

Workshop
May 14–15, 2012 – Washington, D.C
June 11–12, 2012 – Washington, D.C
July 20–21, 2012 – Washington, D.C. – Meeting with vendors
August 21 – Video teleconference
September 12–13 – Video teleconference
October 3–4, 2012 – Research workshop

1.2 Exascale Ecosystem

The proposed research plan presented here must be tightly integrated with existing exascale efforts across DOE. This "exascale ecosystem" includes several key projects that have already been launched and are making significant progress. This proposed research agenda for exascale OS/R fulfills a role for these additional research projects, bringing together the low-level OS/R software with the needed advanced capabilities to explore architectures, middleware, and performance analysis.

- *FastForward* is a two-year research initiative for HPC vendors to explore exascale, focusing on power, performance, and resilience for future CPUs, memory systems, and file systems. The research agenda described here must be tightly integrated with the new and proposed features supported by future hardware. OS/R research projects should be organized to track, interact, and improve the designs of the FastForward projects and the software layers supporting the hardware.

- *Co-design centers* have brought together hardware, system software, and computational scientists working together in a specific DOE problem domains to target exascale. Since the co-design centers [1] [2] [3] have deep connections to the system software and middleware needed to support their programming model, they are ideal partners for exploring novel OS/R features and their impact and performance.

- *X-Stack* projects (http://science.energy.gov/ascr/research/computer-science/ascr-x-stack-portfolio/) have been funded by DOE to explore a wide range of computer science topics targeting exascale, from new programming models to methods for fault containment and explorations of nonvolatile random-access memory (NVRAM) technologies. The research agenda described here complements those projects, focusing on the efficient low-level OS/R features that can best support advanced research proposed by X-Stack projects, as well as new ways to tightly integrate the OS/R with middleware.

- *Synergistic activities* have been launched by DOE, much like the X-Stack projects and tightly focused on exascale:

  - CAL: Computer Architecture Laboratory for Design Space Exploration [4]
  - Execution Models for Exascale [5]
  - Quantifying Overhead in Today's Execution Models [6]
  - Beyond the Standard Model: towards an integrated methodology for performance and power [7]
  - Evolving MPI to Address the Challenges of Exascale Systems [8]

2 Challenges

The challenges for OS/R at exascale can be grouped into two broad categories: technical and business/social.

2.1 Technical Challenges

The DOE exascale reports have identified four key challenges: a significant increase in the number of faults; the need to reduce power requirements; an increased emphasis on memory access and new technologies; and increased parallelism, especially at the node level. These challenges are directly relevant to the OS/R layer. Additional challenges are related to hardware heterogeneity, structural and resource challenges within the OS, legacy issues, new programming models, and dynamic environments.

2.1.1 Resilience

Efforts to reduce power (e.g., by running at close to threshold voltages) and increase component counts will lead to significant increases in the number of faults. Furthermore, there will be an increased potential for "silent errors," faults in parts of the system that are not protected by extensive error detection and correction. Specific challenges within resilience include the following:

- **Fault Detection:** Detecting faults will be more difficult across high component counts and at multiple levels within heterogeneous systems.
- **Fault Notification:** It will be difficult to ensure timely propagation of fault notifications across large networks, where communication bandwidth is limited.
- **Fault Management:** The operating system will have to provide support for the management of faults in runtime systems and applications, including the ability to respond to and contain faults, at unprecedented scales.
- **OS/R Survival:** Despite the high prevalence of faults and silent errors, the OS will have to ensure the containment of faults and continuation of OS functions after faults, in order to avoid fail-stop behavior.

2.1.2 Power

Power management has been identified as potentially the primary challenge for exascale systems. Arguably, all the challenges that are unique to exascale systems derive from the need to stay within a tight power budget (20–30 MW). More specific aspects relevant to OS/R include the following:

- **Hierarchical Power Management:** Hardware power management decisions may be relegated to the OS, which may in turn pass these to the runtime system, which may in turn pass these to the application.
- **Dynamic Resource Changes:** Resources could change dynamically to adapt to power requirements, for example, cores changing speed or powering off nodes. The OS/R must be able to cope with the varying quality of service that will result from these dynamic changes.
- **Global Optimization:** The OS will need to provide the basis for a system for global control of power management. The problem of global management could be exacerbated by the possibility that local optimal power management decisions may be globally suboptimal.

2.1.3 Memory Hierarchy

The memory system is a significant consumer of power in modern computing systems. This is expected to drive future programming models as they focus on reducing the costs associated with moving data within a node and between nodes. Moreover, new memory technologies will be aimed at reducing the power costs associated with memory. The operating system will have to provide support for these new programming models and new technologies. Specific challenges include the following:

- **Integrating NVRAM:** The integration of NVRAM directly into the memory hierarchy will likely require revising aspects of the OS organization.
- **Managing Overheads:** Memory access overheads will have increasing impact at scale. The challenge is how to further reduce the current OS overheads.
- **Software-Managed Memories:** The OS will need to provide more support for runtime and application management of memory (possibly moving traditional OS services, like updating translation tables to runtime systems).

2.1.4 Parallelism

In order to achieve 10^{18} operations per second, applications will need to have billions of calculations in flight at any point in time. This situation will be a significant challenge for application developers. It will also create significant challenges in the OS and runtimes. Since performance cannot increase through additional clock scaling, additional parallelism will need to be supported in the next generation of systems, both on the node and across nodes. Operating systems and runtimes will have to meet a number of challenges related to this increasing parallelism:

- **Efficient and Scalable Synchronization:** Synchronization overheads will need to be minimized in order to reduce the costs associated with blocking and to reduce the need for application-level parallelism to hide latency. A further challenge will be making synchronization mechanisms scale while maintaining consistent management of shared resources.
- **Scheduling:** Scalable scheduling and dispatch of execution contexts will be needed in order to manage large numbers of computing resources.
- **Scalable Resource Management:** The OS will need to provide scalable management (coordination) to support fair access to constrained resources (e.g., memory and network bandwidth).
- **Global Consistency, Coordination, and Control:** Global consistency, coordination, and control will need to be managed across an application or system with tens to hundreds of thousands of nodes, each with thousands of processing elements.

2.1.5 Additional Hardware-Related Challenges

The computational resources of an exascale system will present several challenges for effective resource management, including:

- **Heterogeneity:** The hardware resources in an exascale system will almost certainly be heterogeneous, including multiple types of processing elements, multiple types of memory, and so forth. Different types of resources may be capable of providing the same functionality, but will have different performance characteristics (e.g., high throughput versus low latency), complicating the assignment of these resources to different parts of the computation.

- **Locality and Affinity Management:** The node resources will have complicated affinities, for example, memory and network or processing unit and memory. Effective use of these resources requires that the OS, runtime, and/or application be aware of these affinities to attain reasonable performance.
- **Hardware Event Handling:** Establishing handlers for hardware events will be more critical in order to support responsiveness to faults, energy management, system and application monitoring, and so on. OS and runtime systems will need to provide flexible mechanisms for associating handlers with hardware events and lightweight scheduling mechanisms to ensure that the desired handlers are executed with the desired priority.

2.1.6 OS/R Structural Challenges

The new operating system for exascale will have to deal with several structural challenges:

- **Misalignment of Requirements:** Current node operating systems are designed for multiprocessing, which can needlessly interfere when a node is dedicated to a single application. OS interference needs to be minimized, while at the same time still providing the required level of support for all applications.
- **User-Space Resource Management:** Operating systems traditionally have mediated and controlled access to all resources in the system. However, a variety of new programming models and runtimes will have significantly different requirements for resource management, which will likely result in inefficiencies if the OS has to directly manage all resources. In the future, the application and runtime will need increased control over resources such as cores, memory, and power and user-space handling of control requests and events.
- **Parallel OS Services:** Parallel services (e.g., parallel I/O) will be more common: effective development and support for these interfaces will be critical.

2.1.7 Legacy OS/R Issues

Reusing constructs from existing operating and runtime systems may, at first glance, seem desirable. While doing so can reduce development time, several challenges must be addressed when considering the reuse of these constructs:

- **Fundamental Changes in Design Assumptions:** Many of the design assumptions underlying current OS are fundamentally broken for future hardware (e.g., the file centric fork/exec process creation model of Unix).
- **Managing Intranode Parallelism:** The node operating and runtime systems for an exascale system will need to be highly parallel, with minimal synchronization. Legacy operating and runtime systems tend to be monolithic, frequently assuming mutual exclusion for large portions of the code.
- **Enclaves:** The node operating and runtime systems for an exascale system will need to support tight interaction across sets of nodes (enclaves). Legacy operating and runtime systems have been designed to impose strict barriers between nodes.

- **Right-Sizing OS Functionality:** The node operating and runtime systems for an exascale system will need to be right-sized in order to meet the needs of the application while minimizing overheads. Legacy operating and runtime systems do not emphasize restructuring to match the needs of an application.

2.1.8 Applications Structure

As machines grow larger, applications are not just adding more grid points but are adding new modules for improving fidelity with new physics and chemistry. Rather than a monolithic source code tree, these sophisticated applications are built from multiple libraries, programming models, and execution models. Future operating and runtime systems must support the composition of several types of software components, from legacy math libraries written in MPI to new dynamic task-graph execution engines running simultaneously with a data reduction component.

2.1.9 Dynamic Environments

Current runtime systems are often static, mapping processes and threads to physical resources exactly once, at program launch. This strategy can make dynamic load balancing and resilience difficult to achieve. An exascale runtime system, however, must assume a more dynamic programming environment and physical resource pool. Adaptive power management and transient faults will necessitate a runtime system that can adaptively remap logical program constructs to physical resources as well as respond to elastic resources.

2.2 Business and Social Challenges

In addition to the technical challenges, we have identified several challenges that arise from apparent conflicts in business/operating models and broader community issues. While these challenges are essentially orthogonal to the technical challenges, approaches to the technical challenges that do not consider the business and social challenges will not be successful.

2.2.1 Preservation of Existing Code Base

Given DOE's enormous investments in current code bases and the cost of reverifying and requalifying new codes, the new operating systems must provide comprehensive and high-performance support for codes that were developed under current, legacy OS/R models. The challenge is not only to support existing application investments but also to establish a clear migration path (on-ramp) from current OS/R models to the new environments that will be developed for exascale.

2.2.2 Lack of Transparency from Vendors

Vendors are understandably reluctant to share much information about their future plans related to hardware and software. This reluctance is due partly to the need to protect proprietary information and partly to the need to change their plans quickly in response to market forces. Lack of details from the vendors about future hardware plans will make it more difficult to design OS/R systems that can take full advantage of future vendor developments.

2.2.3 Sustainability and Portability

Organizations funded purely for research cannot be expected to be the sole source of ongoing support for the results of that research. Successful research results must be connected to a vendor community that will provide sustainable support for these results, while enabling continued development of new directions in OS/R research.

2.2.4 OS Research Not Focused on HPC

Until recently, interest in fundamental OS research had been declining. The broader computer science research community regarded OS design as complete and therefore focused on innovation in runtime systems and OS implementation details. Recently, research in operating systems has been reinvigorated, driven by changes in hardware and the development of novel applications. This new OS research, however, has not focused on HPC issues, instead being driven by trends such as the development of lightweight virtualization layers and mobile operating systems (e.g., Apple's iOS and Google's Android). The challenge is to redirect some of this renewed interest in OS research into areas that are more relevant to HPC.

2.2.5 Scaling Down the Software Stack

The HPC ecosystem is not sufficiently large to support multiple software stacks. Currently, the highest-performing supercomputers such as those built by Cray and IBM have custom software stacks. At the other end of the spectrum are simple Linux clusters connected with either GigE or InfiniBand. An exascale OS/R research effort cannot result in a third, independent software stack. Instead, new research to improve extreme-scale OS/R components must be integrated into future software stacks for the most capable HPC systems. Moreover, such components must scale down to smaller development systems as well as permit software development on desktop-class systems.

3 Requirements

To determine requirments for future OS/R systems, the TC conducted surveys of application developers, vendors, facility managers, tools developers, and the research community.

3.1 Applications, Libraries, and Solvers Input

Applications are often indirectly influenced by the core OS through their interaction with runtime systems and libraries. The technical council surveyed the application scientists to understand the needs imposed on OS/R by applications, solvers, and libraries. Due to the indirect nature of the needs, we sought to understand current application user practices and their plans for the future with respect to runtime systems and the job scheduling environment. We solicited feedback from developers of applications, numerical libraries, and solvers by distributing the following questionnaire among the application teams in the DOE and NNSA labs. We requested feedback from all application teams that were interested in exploiting extreme scale systems.

- What runtime system(s) do you currently rely on (MPICH/OpenMPI, pthreads, Charm++, PGAS, OpenMP, TBB, Cilk, Python, etc.)?
- Are there specific areas where the runtime is deficient?
- How do you expect this to change in the future (i.e., do you expect that the runtimes will incorporate critical issues like resilience and power management, or will you adopt a wholly different runtime)?
- Which aspects of the runtime are most important to your application (choose your definition of "important"): nonblocking collectives, dynamic process creation, active messages, local persistent memory, etc.?
- How important are issues related to system (job) scheduling (e.g., topology aware mapping, workflow scheduling, coscheduling of critical resources such as visualization)?
- How important are issues related to intrajob isolation (e.g., protection across workflow components or composed applications to support use cases such as in situ analysis)?
- What else do you want to make sure we consider in OS/R research activities?

Application developers are focused on the design of their codes for use on today's systems while also preparing for the exascale. Many of the applications are currently being developed using MPI and OpenMP due to their widespread availability and support. Application teams are increasingly employing dynamic languages such as Python as flexible scripting tools to steer the application. At the same time, they are exploring alternative programming models and runtime systems – PGAS, OpenACC, CUDA, TBB, etc. – to program accelerators, multi-core, and distributed memory systems. Early exploration of these runtime systems helps application teams evaluate their usefulness in helping applications meet future programming challenges.

While application teams are concerned about managing resilience and power on the exascale systems, they expect significant support from the runtime and other layers of the software stack in easing the challenge. The application developers want the OS/R to support efficient checkpointing and "better" reporting of errors to the application. Some responders also expressed interested in exploring new application-specific approaches to resilience. However, most of the teams were expecting other layers of the software stack to handle many of the aspects of supporting resilience.

The position of the application teams we surveyed with respect to the power/energy challenge was unanimous. None of them were exploring application-level solutions or anticipating the use of specific OS/R support that could aid applications in managing power. There was universal expectation that power and energy challenges will be handled by other software layers with little, if any, application involvement.

The application developers hoped that job schedulers would continue to efficiently support their applications and workflows through job isolation and coupling of application and analysis components. In particular, topology-aware allocation of resources was often identified as a crucial aspect at exascale. Several applications currently employing workflows for scientific visualization were interested in efficient OS/R support for real-time visualization. Sharing of resources between application and analysis components was also identified as a requirement to support in situ analytics. The feedback focused on the need for such support without prescribing specific solutions or approaches that were being explored.

Beyond broad needs, application teams also identified specific requirements and potential limitations. Several teams stressed the need to efficiently support dynamic languages such as Python that enable flexible scripting. Improved support for finer-grained thread management and locality-aware memory management was often requested. This included efficient support for thread creation, thread-to-processor mapping, context switching, and fast synchronization primitives. Interoperability between existing and proposed programming models and runtimes was a common concern. Examples cited include inter-operability between active messages and MPI, and the co-ordination of NUMA mapping between the operating system, intra- and inter-node runtimes, and the job scheduler.

Several teams repeatedly stressed the importance of performance and level of support for existing and proposed functionality on par with the design of new features. Many of the limitations identified were those of functionality already supported in extant runtime systems, such as MPI collectives and one-sided communication. The teams were also exploring new additions such as transactional memory support but finding the overheads excessive in practice. A related concern was the increasing complexity of runtime systems necessitating additional layers of abstractions in application software.

The usability of the features supported and its impact on productivity was identified as a fundamental requirement. Teams were often interested in the availability of the OS/R on a variety of platforms such that applications can be developed and tested on smaller scale systems before being deployed on production platforms. In particular, support for debugging at scale was specifically identified as a productivity challenge. Several developers pointed out that existing features do not work as needed, possibly identifying a mismatch between the implementation of functionality provided by the runtimes and the needs of the application developers.

3.2 Vendor Input

Although much research is needed in OS/R to support exascale-class systems, it is not feasible for DOE to be the sole maintainer or developer of an exascale OS/R. For progress to be made in this area we need a collaborative effort from the HPC community; vendors and researchers from universities and from DOE labs will need to be involved in order to develop a sustainable OS/R for exascale.

In this section we summarize the views from the vendors who responded to our query for information. Vendors were chosen from their participation in past government procurements and research programs, To focus the responses of the vendors, we provided them with guiding questions (detailed below) and asked for their strategy to provide an exascale OS/R in the projected environment we outlined. Not all vendors contacted provided a representative or responded to all of the questions. Note that the responses from the vendor OS/R researchers do not reflect any plan of record for their respective companies.

Questions to drive vendor engagement:

- What models do you feel work best for research collaboration and the transition to products?
- How do you foresee the dynamics for exascale OS/R research and development?
- What relationship do you expect between the exascale OS/R and your product line?
- What licensing issues (BSD, GPL, LGPL), copyright agreements, patents, contributor agreements, etc. do you foresee?
- What are the key technical issues in the OS/R that must be resolved to achieve exascale?
- Can you outline what you expect to be open source and what you expect to be proprietary?
- What will be the key OS/R APIs (at node, partition/job, system levels), including power mgmt, memory, fault, etc.?
- What tools and programming model influences on the OS/R
- What I/O interfaces (remote filesystems, local NVRAM, etc.) do you feel re needed for OS/R?

3.2.1 Cray

Collaboration is a standard business practice for Cray. Examples include Red Storm, ACES Interconnect, MPICH, MRNet, STAT, OpenMP, OpenACC, OpenSFS, Chapel, Linux, HPCS, and UHPC. Cray leverages community solutions wherever possible, filling gaps in software or capabilities with Cray solutions. Cray does see some of this software as differentiating and keeps it as proprietary; the company expects to continuing to do so in the future, protecting IP with patents. Cray, does, however, comply with all existing open-source licenses such as GPLv2 (Linux kernel) and expects collaborative projects to be open-sourced with a BSD or Apache-style license. Cray also publishes detailed APIs to promote adoption in the community. From Cray's viewpoint, the issue with the model we proposed is the need for funding support, not only for R&D but also for maintenance of the open-source stack, since customers expect full support. One such model is the Lustre parallel filesystem, which is supported and managed by OpenSFS.

The key OS and runtime issues identified by Cray include the following:

- Resource management
- Power
- Memory hierarchy
- Processing elements
- Parallel work distribution
- Resiliency
- Programming model interoperability
- Standardization of APIs across programming models and vendors

Application co-design centers provide one venue for these interactions, but they focus more on the application interface than on lower levels of OS/R. Other community venues are needed for coverage and interaction. Standardizing on APIs across vendors and implementations is critical for adoption. Linux provides some context for this, but not necessarily for HPC and exascale-specific work.

Application power management will affect all levels of the OS/R and will require fine-grained control through a combination of hardware and software. Heterogeneity will add to the complexity of balancing performance and energy usage. Runtimes will require constructs for placement and control of energy, along with tools for autotuning and prediction of energy consumption. Deep-memory hierarchies and complex processing elements will be a challenge (e.g., see the Echelon node design, presented by Bill Dally at SC10, for an idea of the complexity of a node).

Concurrency management and work distribution are mostly a runtime responsibility, because of to its better understanding of the programmers intent; but these will require operating system infrastructure. Directed acyclic graphs (DAGs) are one approach for work scheduling with dependency information. DAGs allow for efficient distribution of work and can take into account the placement of tasks and address processor heterogeneity. Support is also needed to enable monitoring and information collection for analysis and autotuning.

With the latest MPI 3.1 revisions, the MPI fault tolerance infrastructure appears satisfactory; for instance, resilience to network failures is achieved. Although node failures are not accounted for, it is hoped that this issue will be addressed in the MPI 3.1 standard and certainly should be available in the exascale timeframe. Cray has current work in operating services infrastructure. Job launch and control will be resilient to node failures, but communication for these services also needs to be resilient to network failures (often using TCP). The runtime system has a significant role in dealing with both network and node failures.

Issues facing application resiliency include the challenges of shared-memory models with fine-grained communication. Atomic memory operations are particularly challenging to support efficiently. Many of these issues stem from the lack of language semantics that encompass the idea of node failures. Research is also planned to address infrastructure to handle node failures within the context of workload management.

Currently, application resiliency is based on external I/O-based checkpoint, but these solutions are not expected to scale. Localized I/O may help but may not be enough, depending on the failure rates. Other promising solutions include fine-grained and independent rollback and recovery, such as *containment domains* proposed by Mattan Erez, University of Texas at Austin. With *containment domains*, one preserves data on start, then detects faults before committing; to recover, one restores the data and re-executes.

Core operating system issues that need to be addressed include support for heterogeneous processors and resources, noise, I/O bandwidth scaling, and storage hierarchy. Cray's experience with core specialization has shown the benefit of isolating certain activities to a subset of cores; this benefit is seen even at core counts as low as a dozen. The impact to compute resources will diminish with future high core counts; dedicating a single core to OS processes when core counts exceed the hundreds and move to thousands will no longer be an issue. Future processors may also have heterogeneous hardware, and these may be less amenable to traditional OS. Further work on core specialization could include further segregation of services from computation, possibly different kernels in a single OS instance, or both. Multi-instance operating systems on the same node are interesting; but solutions without the overhead of virtualization would be preferred, and the solution must be supported and maintainable.

Currently, external I/O bandwidths are not scaling with computation: there is I/O interference on interconnects and storage devices. "Nearby storage is one solution that can help scaling and reduce interference from other jobs. However, nearby solutions do not provide global semantics. Research is needed in proper provisioning of storage and access methods, especially for global semantics. One complication is the different use cases for I/O: checkpoint/restart files, application input files, staging data to and from globally shared file system, application scratch space, and kernel-managed swap space are all possible use cases. Possible solutions include I/O middleware and I/O forwarding. Existing projects considering such solutions include DVS, IOFSL, SCR, and PLFS.

Supervisor and management system issues include site-level power management and scalable monitoring and control of the system. Site-level power management requires interoperable control between multiple systems down to the ability to control individual jobs. (This is in addition to the application-level power optimization that runtimes are developing.) This is a global optimization problem and will have to take into account variable power pricing, such as day to night, and variable loads at sites and external limits such as external load on the power grid.

For the RAS system we need good out-of-band monitoring. More work is needed on the monitoring of shared resources. We also need predefined classes for workload manager support that accounts for CPU, memory, and network interface controls. Coarse-grained capping should be available based on resource type and class. Power and capping information should be propagated by ALPS (Cray job launcher), and finer-grained static job-level controls should be available at job launch time. Moreover, power and energy will have to be supported in job accounting.

Scalable monitoring and control of exascale systems will require tight integration rather than a collection of ad hoc solutions. Current supervisory systems work but are fragile at large node counts. Additional research and engineering are required for robust monitoring and control. Detailed power monitoring and management also will be needed along with fine-grained fault awareness and reporting. All the data collected will have to be made available to third-party tools through open formats and APIs. Cray has not seen any solutions coming from the research in predictive failure analysis. Some progress has been made in the area, but whether it is sufficient remains unclear.

API standardization issues still exist at the boundaries of these systems. Open-source and industry standards are available for monitoring and control, but more venues are needed for the exchange of ideas and standardization of common APIs.

3.2.2 IBM (Kyung D Ryu)

IBM has a long history of collaboration and R&D with the government and external researchers. It also has specific examples of successful collaboration on OS/R topics such as HPC operating systems with Argonne and the Blue Gene research program. Good areas for architecture-independent research include operating systems, programming models, and the definitions of APIs for power and reliability. IBM has a well-defined strategy for working with and contributing to open-source projects. Following are the current research and technical challenges identified by IBM that are key to OS/R for exascale systems and fit with IBM's exascale vision.

OS: IBM has already proved scaling up to 1.5 million cores, 6 million threads on BG/Q in a production environment using the CNK. Relevant research projects that will help future exascale systems include FusedOS, which partitions cores between light-weight kernel and full-weight kernel, and dynamic large-page support for CNK, which allows a node to share memory pages from other remote nodes.

Challenges in the OS area include supporting next-generation architectural elements such as heterogeneous cores, specialized cores, service decomposition, and virtualization. In addition, supporting and managing various active elements in memory, network, and storage layers will be critical research issues.

Messaging: IBM has converged HPC message-passing interfaces into PAMI—the Parallel Active Messaging Interface. This mature layer supports active messaging and multiple programming models and has incorporated recent innovations such as nonblocking collectives; multicontext communications that allow concurrent network access and endpoint addressing rather than communication based on processes or tasks; dedicated hardware threads for communication; low overhead interrupts; and work stealing by means of asynchronous communication threads. Challenges in exascale messaging include supporting a flat address-space model, reliability of message transport, message ordering, and the interaction of the messaging layer with multithreaded and hybrid programming models.

Storage: Storage challenges include supporting legacy APIs concurrently with new APIs and the need for scalable workflows with abundant concurrency. Creating a balanced storage architecture, including efficient use of storage class memory and other storage innovations, will be an important focus area for future software and hardware investigations and research.

Programming model: Current research in programming models includes a light-weight OpenMP, a new thread affinity proposal for OpenMP, and a new performance tool for OpenMP. These have been submitted to the standards committee for OpenMP. Future programming models must address many challenges including system and node heterogeneity and use models for new memory technologies, while maintaining the large body of codes using existing programming models and also supporting the broad range of new users and applications.

Power: Power optimizations under investigation include aggressively pursuing the idling of hardware resources; development of a power-aware system and application stack; firmware-based optimizations; and ongoing work with BG/Q to improve monitoring, management, and application power characterization. A key challenge is the lack of common APIs, scaling, and cross-stack management. Power is at odds with reliability, performance, and determinism. In order to motivate users to construct energy-efficient programs, power utilization needs to be considered when scheduling and managing jobs. Another research issue is to determine what level of control vendors should give users or operators; too much control could lead to accidental equipment damage, and not enough control could hamper innovation in the runtime.

Reliability: Current research that is applicable to resilience includes transaction-based soft error recovery, thermal-aware scheduling, fault injection experiments to improve fault tolerance, and an incremental and asynchronous checkpointing library. The ASCR X-Stack collaboration project FOX explores alternative OS/R models. IBM also has a collaboration with DOE and PNNL developing a performance health monitor, which looks at automatic performance monitoring to detect and mitigate system failures. The challenges in reliability include thread migration, software-based balancing or wear leveling, and user-based or compiler-based hints for checkpointing. Application tolerance to faults needs more investigation, as does the propagation of fault information through the stack. Also, more hardware sensors and diagnostics are needed in order to detect and react to failure events.

Global management: What should be left to middleware, and what should be supported by more standard system software? Should standard APIs or proprietary APIs be supported? A workflow-based global programming model requires a new execution model and rich system services from OS/R. Global and application-level power and reliability management also requires common APIs. Global OS management allocation and scheduling require support for elasticity, which will require adaptation to varying resources in the runtime and application. Addressing all of these issues in an integrated design is difficult, but such a design is the most promising solution to meet the goals of exascale.

3.2.3 Intel (Rob Knauerhase)

Intel has strong ties to academia, the overall HPC community, and the national labs. Many products from Intel have beginnings as research demonstrations from Intel labs and then are moved through different groups at Intel until they mature enough for integration into products. Key problems identified include fault tolerance, large overhead required to recover from events (in a checkpoint-restart model), new memory organizations, parallelism, and concurrency. One idea is to make a system that can be independently optimized by a domain export, a tuning expert, and an execution expert, so that no one person has to know the whole system. An exascale OS/R will have to adapt and be self-aware, features that will involve the collection and processing of much more information collected through hardware sensors and software sensors. To retain the information compilers collect from the code and to transfer this knowledge to the runtime, we need something better than current ELF binaries. We also need to propagate optimization hints up and down the stack.

Cross-stack cooperation is needed in order to support new memory hierarchies. The ability to dynamically reposition code and data for energy, performance, or faults is a key requirement of an integrated OS/R.

The key exascale philosophy with Intel's Runnemede and UHPC projects has been the development of support for a fine-grained, event-driven execution model, with sophisticated observation. One method to address the resilience issue is the concept of *execution frontiers*, which schedule codelets in a dataflow fashion but keep enough state to roll back the execution when a failure occurs. This method is currently in the research stage but has been developed for the Intel UHPC work supported by DOD. Execution frontiers also allow for algorithm substitution, which could address the heterogeneity of resources, or allow the selection of algorithms with different precisions in order to optimize for power or performance.

Another collaborative effort from Intel is the Open Community Runtime. This is a shared framework for resource control with an exascale focus that allows modular components. Here Intel seeks to integrate existing work such as Rice University's Habanero-C and Sandia's Q-threads along with Intel's research from UHPC and Runnemede. This work is funded though industry, academia, and government and will eventually result in a *crossbar* of runtime systems to hardware available through common APIs and OS support.

3.2.4 Nvidia (Donald Becker)

Nvidia is focused on hybrid computing and power efficiency, key elements for the success of exascale. Nvidia has internal OS teams with broad expertise, and it has broad HPC application experience as a result of the focus of the Tesla business unit. The company also has experience with open-source collaborations and is a top 10 contributor to the Linux kernel.

Nvidia sees an evolving role for the OS in HPC systems, where the OS/R is more involved in the broader system constraints of compute efficiency, power steering and node and rack thermal management. The company believes that OS/R should be less involved in the controlling the hardware and more involved in setting policy that the hardware will adhere to. Nvidia envisions *OS bypass* as the norm, where activities such as thread and task creation are handled by hardware. Nvidia also believes that the OS will have an expanded role in managing memory as a result of different types of memory becoming available.

Nvidia has a lead in power-efficient computing from hybrid HPC systems with GPUs, down to the mobile Tegra and ARM processors in phones and tablets. HPC systems are limited by power and thermal constraints, however, and the systems need the ability to manage through a combination of firmware, node OS, and systemwide software. The current state of the art is the Intelligent

Platform Management Interface (IPMI), but this has limitations (e.g., it does not monitor network status). Much can be leveraged from the advancements of NVidia's work in mobile devices. These devices have stringent power requirements and can attribute power use down to processes and applications, something that HPC runtime and systems software could make use of. HPC needs better definitions for the interfaces and structure for power-related control. *Device trees* in Linux are a path forward, but OS-independent and other OS-dependent APIs are also needed—such as the reporting and configuration of IPMI, application interfaces to request performance levels, and accounting tools that relate power use back to applications.

Fault isolation is another area with much room for improvement—containing faults to the local applications and attribution of faults. Also needed is propagation of faulty information from the hardware to the OS and runtime. These techniques could then be extended beyond the node level. Fault recovery methods that do not impact other applications on the node, such as a soft reboot of a GPU, is an example of this type of fault containment.

Memory management will have to broaden to new memory hierarchies. We will need management of peer memory types, such as grouping DDR, GDR, and LPDDR resources. The nonuniform performance of this new hierarchy should be taken into account, especially for PGAS programming models. In order to support this, multiple memory management units will be required in hardware, and the OS will have to use these efficiently.

NVidia has an in-house testbed devoted to addressing the issues identified. Currently, power-efficient control is under development, and performance trade-offs for HPC applications are being investigated.

3.2.5 Vendor – common themes

Common themes from vendors include the opinion that an external organization is needed to hold common specifications and open-source projects and code. A third-party foundation is viewed as a requirement for any multivendor collaborative effort, its main purpose being to mitigate liability for the vendors. Most vendors prefer a BSD-type license if working with open-source software; other acceptable licenses mentioned were IOSL, Eclipse, Apache, CPL, and MIT. Since vendors do not want to give up the option of having patents, we need well-defined content areas. Vendors also agree that if there is a open stack for exascale, a model and funding are essential in order to maintain that stack.

3.3 Facility Input

People who manage facilities identified requirements for OS research advances in several key areas, as detailed below.

3.3.1 System Analysis

The main OS challenge for system analysis in the move to exascale is implementing support for gathering comprehensive data efficiently at scale. An urgent need exists for global integration of disparate monitoring and control systems, with sufficient granularity to provide per-user, per job data that includes energy usage, errors/faults and memory usage. Some of this information will have to be available to users as well as administrators, for example, to verify memory usage. A further challenge is the requirement for real-time monitoring (e.g., faults, RAS, memory usage), in addition to out-of-band performance monitoring.

The kind of data that should be collected by this out-of-band performance monitoring includes the following;

- Per user, per job data including energy and errors
- Live real-time fault and RAS data that is easy to stream and filter
- Out-of-band performance data on power and energy, memory usage, soft errors, and so forth
- Scalable memory usage monitoring
- Verification of memory usage
- Multiple levels of monitoring, both real time and post hoc
- Better global monitoring of disparate control systems, including integration with building control systems.

In addition, to understand and mine the accumulated data, administrators of these systems require the following features:

- Anonymization of log data
- Mining of log data for information
- Common formats

3.3.2 Fault Management and Maintenance

From the perspective of facilities, several aspects of fault management need to be addressed by the OS for exascale. Systems must be able to cope with failures (e.g., through process migration or other solutions), so that jobs can continue to run with minimal interruption. Support for testing also is needed, specifically the ability to run recurring memory, processor, and network tests on a subset of nodes without impacting jobs running on other nodes. Related to this aspect, the OS must support rolling updates, partial updates, and rollback of updates. Additionally, the OS needs to integrate support for sensor readings on power quality so that nodes and networks can be shut down quickly if required.

3.3.3 Systemwide Energy Management

In order to cope with the issues of energy management at exascale, the OS needs to support full integration of a facility's energy infrastructure. This includes both monitoring at the site level, encompassing multiple security domains and disjoint networks, and control interfaces for power capping and control at multiple levels (e.g., node, rack, set of racks). An added challenge is determining how best to schedule jobs based on energy management requirements, for example, off-peak versus peak power demands.

3.3.4 Performance

Job placement for massively parallel codes at exascale will be a major challenge and will likely require introspection and dynamic adaptation to optimize system usage. This will require advances in many aspects of the OS, including robust topology awareness, performance predictability, QoS, and accurate monitoring. Performance predictability and QoS are also important for application programmers so that they can more effectively optimize codes, for example through autotuning. The aspects of performance critical to the exascale operating system are minimization of launch time for huge jobs and fast booting for subsets of nodes.

3.3.5 Customization

For an exascale operating system to support a diversity of jobs, programming models and usage patterns on a huge scale will require the ability to customize parts of the system for particular jobs. The challenge is to provide support for user specialization, such as different software stacks, custom schedulers and runtimes. These different customizations must be available simultaneously to different jobs, as needed. The operating system will have to provide extensive user-level support (a full set of system calls and library support) if required by the application; but it also must be customizable so that other jobs can be run with minimal OS interference, as close to the bare hardware as possible.

3.4 Input from the Large-Scale Tools Community

We surveyed a wide range of experts in the community that designs and builds performance analysis and correctness tools, as well as the infrastructure on which they are based. Our questions to these experts, which provided the opportunity to capture their requirements for exascale OS/R, were as follows:

- Are there specific areas where the runtime is deficient?
- How do you expect this to change in the future (i.e., do you expect that the runtimes will incorporate critical issues like resilience and power management or do you expect other software layers to be required to handle them)?
- How important are issues related to system (job) scheduling (e.g., topology aware mapping, workflow scheduling, coscheduling of critical resources like visualization)?
- How important are issues related to intrajob isolation, for example, protection across workflow components or composed applications to support features like in situ analysis?
- What do you think distinguishes tool OS/R requirements from those of applications?

- What is the impact of major exascale challenges (power, resilience, scale, dynamic resource management, heterogeneity, asynchrony) on your needs from the OS and/or runtime?
- Our definition of OS/R is broad (e.g., it includes resource managers) and requires underlying infrastructure similar to that needed by many tools (e.g., scalable multicast and reduction capabilities). What do you consider to be the technical overlaps and interactions between tools and OS/R?
- What do you consider to be the best R&D model for managing interactions between these technical areas? How do you think technical overlaps should be handled? How can we best avoid duplication of effort and achieve reuse across these technical areas?
- Tools often require efficient access to hardware resources that are usually restricted to privileged users by default. What resources do you need to access? What are your requirements in terms of cost for that access? How frequently do you access them? How would you propose to restrict access to those resources?
- Performance and correctness of an application depend on the behavior of many hardware and software subsystems (e.g., load balancing, power management, libraries, OS, file system). How do you want to handle the attribution to a specific component or instance and at what granularity (e.g., frequency, resolution, LOC, or function)?
- What else do you want to make sure we consider in OS/R research activities?

These questions provided the opportunity to gain input not only on technical requirements but also on the overlap between tools and OS/R. Further, we also sought input on how to manage interactions between the communities that work on these topics.

Responses from tool implementers confirmed many of our preconceptions of tool requirements on exascale OS/R but also revealed important additional concerns. At a high level, tool OS/R requirements overlap those of applications; however, they also extend those of applications.

3.4.1 Overlap with Applications

One area in which tool OS/R requirements overlap those of applications is the need for efficient bulk launch for scalability. The mapping of tool threads of control to the overall system and affinity between them, and those of the application, is important both for tools and for applications. Furthermore, tools need OS/R support to handle heterogeneity and scale. Like applications, support for in situ analysis is critical for tools. Indeed, many have already implemented in situ techniques and thus may serve as precursors for general requirements in this area. Tools also require synchronization for monitoring across enclaves and low-overhead ways to cross protection domains. The latter requirement is perhaps more general than for applications, which often need to cross protection domains only for communication between threads of control.

Applications need well-defined APIs for information about key exascale challenges. Tools will frequently provide measurements relevant to these challenges. Occasionally, they will even implement the solutions in generic modules on which applications will build. Overall, tools require these well-defined APIs at least as much as applications. Specific areas identified for creation of portable APIs include power and resilience. Tools also need APIs to handle asynchronous execution; these APIs may be distinct from those used by applications—or at least will extend them, since tools must measure the efficacy of asynchronous approaches as well as use them.

Another potential overlap in the requirements of tools and applications involves quality of service concerns for shared resources. Tools need QoS guarantees to ensure that they do not too severely impact application execution. A particular area of concern is that tools can have extensive I/O requirements.

3.4.2 Additional Requirements

Despite the cited overlaps, tool requirements are broader than those of applications. Generally, the boundaries between applications and the OS and runtime are fairly clear, whereas tools often directly implement features that might alternatively be incorporated into the OS or runtime. Such incorporation would serve the same purpose as reusable tool components in reducing the effort to implement end-user tools. Further, it could, in many cases, provide lower overhead implementation of the functionality.

A specific area in which tool OS/R requirements extend those of applications is the need to launch with access to application threads of control. Often tools must directly launch those threads of control. However, tools also must be able to locate the application threads of control and attach to them after they are launched through the typical production (non-tool) mechanism.

Generally, tools need low-overhead timers, counters, and notifications. Most systems currently provide such timers, and low-overhead access to performance monitor hardware is also common. However, notifications (e.g., interrupts) often have too high an overhead, and the need to minimize the overhead of notifications appears likely to grow in exascale systems.

Related to low-overhead timers and counters, some researchers suggested that generic measurement conversions might best be provided within the operating or runtime system. This functionality is typically provided for timers, which often provide a raw cycle count that OS/R routines can convert to a more standard time measurement. However, even this currently available functionality will become more complex with the likely prevalence of processors that can change frequencies without any software intervention.

Tools require access to a wider range of resources than those that applications access. Such resources include counters related to shared resources, such as network hardware. A particular concern is that tools require attribution mechanisms for these resources. They must be able to associate use of shared resources to specific applications, ideally to the specific threads of control and to lines of code and data structures being accessed. With respect to code lines, low-overhead mechanisms to determine call stacks are of critical importance; the tools community has already designed techniques that provide this functionality, although, like measurement conversions, they might be more efficient (and easier to use) if integrated into the OS/R. When considering attributions for shared resources, such as network hardware, solutions probably need to include hardware support. The OS/R must facilitate access to this special-purpose hardware, rather than preventing it, as often occurs. Overall, tool implementers expect that OS/R mechanisms for aggregation and differentiation could greatly assist their efforts.

One unique requirement for tools compared with the requirements of applications comes in the form of multicast/reduction networks. Tools share this requirement with many aspects of the OS/R. Often, the enclave OS must combine data from multiple compute nodes in the enclave or must distribute information to them. For example, I/O forwarding services often provide combining functionality. Similarly, bulk launch facilities provided by the global OS must contact a number of compute nodes. While resource managers (RMs) already use tree-based networks for this task, the networks are often contained within the RM implementation. Further, tools have used the combining functionality much longer than OS/R implementations. In general, tool accesses often stress different performance aspects from those of the OS/R. While implementing a generic multi-

Table 1: Position papers received and accepted for presentation at the workshop

Topic Area	Received	Accepted
OS/R Architecture/Structure	13	3
Autonomic/Adaptive Approaches	9	6
Core Specialization	9	4
Fine grained / Dynamic tasks	5	3
Power/Energy	6	3
Resilience	7	3
Unified Runtime System	3	0
Global OS	6	0
Other	6	3

cast/reduction capability may seem primarily an engineering exercise, significant research is needed to design techniques that balance the competing performance requirements that are partially captured in the concepts of latency sensitivity versus bandwidth sensitivity but also involve issues with asynchrony and other exascale challenges such as resilience.

3.4.3 Summary

Tool requirements create many important OS/R research topics. Perhaps the most significant is how to maintain the existing level of service and access for tools as system sizes increase and applications increasingly adopt asynchronous execution paradigms. Other critical questions involve low-overhead notifications and attribution of activities involving shared resources. Moreover, multicast/reduction networks that can be shared efficiently between OS/R activities and tools—and balance competing performance requirements—are important areas of research.

3.5 Research Community Input

Input from the research community was solicited in the form of position papers. The call for position papers was circulated widely, including HPCwire. We received 80 responses to the call. Sixteen of the submissions were ruled out of scope because the position paper did not describe strategies to address core topics in operating or runtime systems for extreme-scale systems. The remaining 64 papers were partitioned into categories and reviewed by members of the technical council. Twenty-five of these were selected for presentation in a workshop (see Table 1). Priority was given to papers that emphasized novel, high-risk/high-reward approaches.

The OS/R Software Technical Council Workshop was held at the American Geophysical Union in Washington, D.C. on October 4–5. The agenda for the workshop was organized around topic areas, using a format that designed to promote interaction among workshop participants. Each session covered two of the topic areas and concluded with a panel discussion. The topic areas included the presentation of three or four position papers. Each presentation was limited to 10 minutes; and the topic area concluded with 15 minutes for questions, answers, and general comments from the audience. In the panels, invited presenters addressed a set of specific questions prepared by the technical council (see Table 2).

In the final panel of the workshop, speakers presented their impressions of the workshop and offered recommendations for future directions.

Table 2: Topic areas and panel areas of presentations

Topic 1	Topic 2	Panel
OS Structure	Core Specialization	Global and Node OS Architectures
Memory Structures	Fine grained execution	Support for Novel Programming Models
Power/Energy	Resilience	Power and Resilience
Adaptation	Adaptation	Adaptation

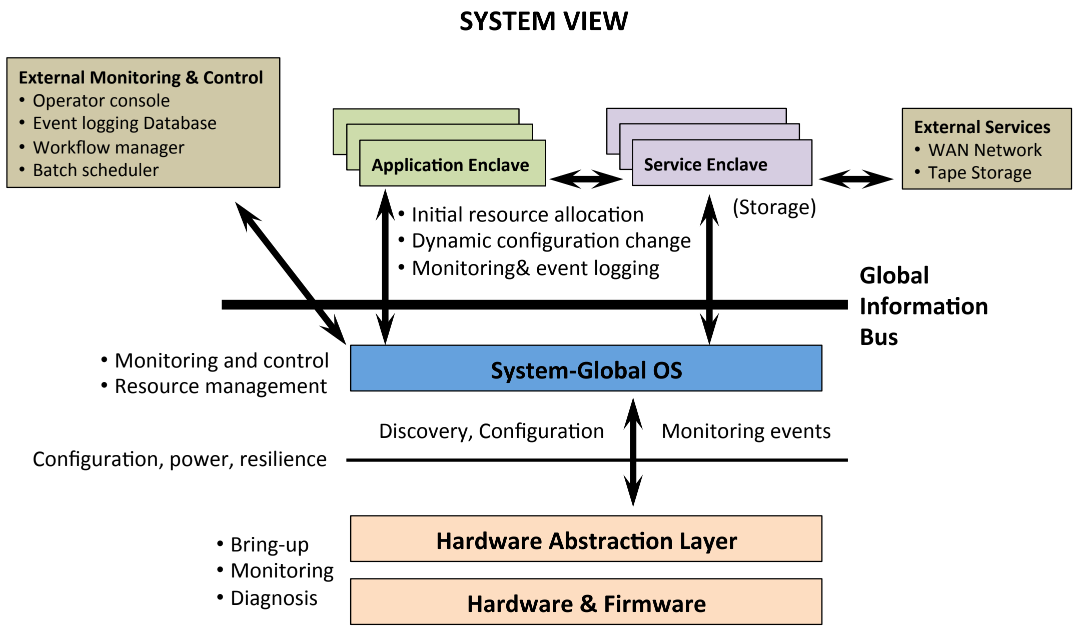

Figure 1: Software stack, machine level

The workshop was lively and highly interactive. The participants collectively shared a deep and wide expertise in OS and runtime for both HPC and enterprise. The size and organization of the workshop enabled all to share their thoughts and debate research issues. The discussion and interactions during the panel were captured by scribes assigned to each panel. The technical council also received written feedback from panelists and workshop participants. The position papers, presentations, notes from the panels and Q&A sessions, and postworkshop feedback were invaluable to the technical council in putting together the report and recommendations.

4 Reference Architecture

We describe in this section a reference architecture for exascale OS and runtime. The basic functions are similar to functions of various components of the current software stack, with some extensions; however, the organization of these components may be different, in order to address scalability and resilience issues at exascale. We see this reference architecture as a starting point of a community effort to further refine and validate this design.

This generic structure is illustrated in Figure 1. The bottom layer includes the system's hardware (compute nodes, storage nodes, network, service processors, etc.). The hardware/software interface is defined by a *Hardware Abstraction Layer* (HAL) The *System-Global OS/R* (SGOS) layer includes all the system functions that are local to one system but can affect the entire system; in particular, they can affect multiple concurrent jobs. Physically, the SGOS encompasses OS/R code that runs on compute nodes, I/O nodes, service processors, consoles, and so forth.

An *enclave* (i.e., partition) is a set of resources dedicated to one application or one service. To the greatest possible extent, system functionality is encapsulated within enclaves. In particular, the failure of an enclave should not cause global system failure, and different enclaves can provide different implementations of the same function. As a concrete example, we propose that a parallel loader be an enclave function that involves only nodes within the enclave. The interaction with a system global function (the resource manager) will be through standard interfaces: the resource manager passes the load file to the enclave loader, and the enclave loader returns a success indicator. An interactive partition may use another loader.

The major subsystems of the SGOS are described below.

4.1 Hardware Monitoring and Control

The hardware monitoring and control subsystem (HMC) monitors the health of the system's hardware and manages the hardware configuration. In current systems, the monitoring subsystem feeds information into operator consoles and event logs. As resilience becomes a key issue, this infrastructure will need to feed into a real-time *autonomic control system* that automatically initiates reconfiguration actions. Some of these actions (running diagnostics, isolating faulty components, reconfiguring nodes to use spare components) will be internal to the HMC subsystem. Others, such as restoring a file system or recovering a running application, are performed by higher-level subsystems. Therefore, the HMC subsystem will have multiple, dynamically bound clients that can communicate both ways: be informed of failures of hardware components that may affect them, and inform the HMC subsystem of higher-level failures that may be indicative of HW failures. The HMC system will provide standard interfaces for getting information on the hardware configuration and on events that change this configuration (e.g., HW failures).

The HMC subsystem can be further subdivided into components that monitor and control various hardware components: compute nodes, network, racks, and so on.

4.2 System Monitoring and Control

The system monitoring and control (SMC) module monitors and controls the low-level initial software configuration of the system components. In particular, the SMC is responsible for the bootstrap procedure that installs an initial kernel on each node and creates an initial communication infrastructure that connects all systems components. This subsystem may be integrated with the HMC subsystem. Like the HMC subsystem, it will provide interfaces to get information in the low-level software configuration of the system and events changing it. The SMC may leverage existing standards (e.g., bootp), but those will need to be adapted in order to improve scalability and reduce cold-start time of large systems.

4.3 Resource Manager

The resource manager (RM) allocates resources to enclaves and controls the external interfaces of the enclave (connections and permissions). We say that a resource is *managed* if the system can provide some QoS guarantee to different sharers of this resource. Currently, nodes are a managed resource, and the allocation is static (for batch jobs). In the exascale timeframe, power will become a managed resource. It is also possible (and desirable) for other resources to become managed: internal storage volume, storage bandwidth, network bandwidth, external bandwidth, and the like. Consider, in particular, network bandwidth. Currently, the system network is shared, and the performance of one application can be affected by the traffic of another application, resulting in significant fluctuations in performance. One could avoid this problem by overprovisioning network bandwidth, ensuring that congestion is extremely rare. But such a solution is expensive. Alternatively, one can avoid sharing by allocating disjoint network partitions to each enclave. When high-radix networks are used, however, this will lead to a significant loss of overall bandwidth, since a large fraction of the links are not utilized. Instead, one may want to time-share links, but use virtual channels to guarantee a fixed fraction of the channel bandwidth to each application. In such a situation, network bandwidth becomes an allocatable resource.

We also expect that resources will be allocated more dynamically, because of the dynamics of the underlying platform (due to power management and frequent failures) and the shift from monolithic applications to heterogeneous workflows. Thus, an enclave may acquire new nodes, lose nodes because of failures, or return nodes to the global pool; the power available to it may be increased or decreased.

The resource manager aslo sets permissions and sets communication channels that connect an enclave to another or to external interfaces. For example, the RM will connect a newly created enclave to the service enclave that provides it with I/O services, specifying levels of service, permissions, allocations, and so on.

Multiple enclaves can be connected in order to support workflows. Workflows can be also supported within an enclave; components within an enclave are tightly coupled, while communicating enclaves are loosely coupled. This configuration is analogous to the difference between coupled threads within one OS process vs. processes communicating via sockets. Efficient workflow support across enclaves will require the definition of new parallel communication and control mechanisms; see, for example, [9].

The SGOS interacts with the underlying hardware, with the active enclaves, and with external clients and services. The interaction with the underlying hardware includes a *discovery* process, where the SGOS discovers the hardware configuration, and the monitoring and control interaction. Much of the discovery process is done today by reading configuration files. This should be complemented by protocols that query the actual hardware, in order to avoid inconsistencies. Also, to the extent possible, the interaction with hardware should be mediated by the *Hardware Abstraction Layer*, in order to insulate the SGOS from minute hardware details.

The external interfaces of the SGOS enable external monitoring, control, and resource allocation. Information about the system health is fed to an operator console and to an event database; the operator can control system configuration; the total power consumption of the system can be adjusted, depending on external factors (electricity price, external temperature); and resources can be put under the control of an external resource manager, for example, an external workflow engine. In effect, we have an hierarchy of "operating systems": grid operating system, machine room operating system, machine operating system, enclave operating system and node operating system. The view we embrace is that the SGOS is in full control of a machine; it can delegate capabilities to the higher levels of management (machine room, grid) or to the lower levels (enclave, node). These various "operating systems" are embodied by software running at different locations; the SGOS keeps tracks of the location(s) of components that it interacts with.

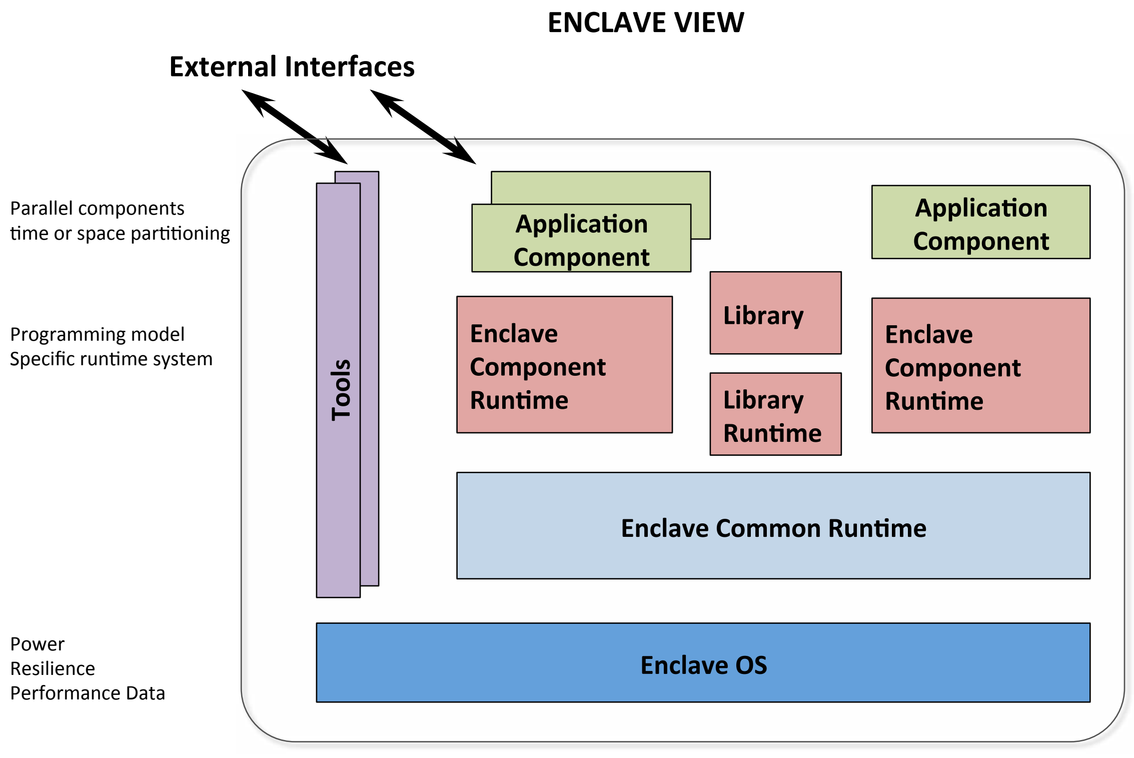

Figure 2: Structure of enclave software.

4.4 Enclave

Figure 2 illustrates the putative structure of an application enclave. The *Enclave OS* (EOS) and *Enclave Common R/T Services* handle functions that are enclave-global. These functions include the initialization and management of the communication infrastructure within the enclave; work migration and load balancing; and recovery from failures that the enclave can handle on its own. The current division between "kernel" and "runtime" may change; as a rule, services will be provided in kernel mode only when this is necessitated by the hardware or when it is necessary in order to ensure partition isolation. Therefore, we do not describe the division of labor between these two layers but instead consider them as a whole, using the acronym *EOS/R* to cover both.

We distinguish between common services that are used by all programming models, libraries, and languages and component-specific services that are provided in the run-time of a specific library or language. The assumption is that different programming models, libraries and languages use a common set of services and map their specific services atop these common services. This will reduce development costs for each environment. More important, the common services provide interoperability across the different environment and avoid conflicting resource requirements.

The EOS/R is implemented by kernel code and libraries running on each node. Some of the services are node-local, such as the management of local resources (core scheduling, memory and power allocation), and the handling of coordination across threads. Others require coordination across multiple nodes, such as user-space communication, including rDMA, collectives and active messages. New technology will be needed to better integrate internal communication and synchronization mechanisms with external ones (e.g., transactional execution of active messages, message-driven task scheduling).

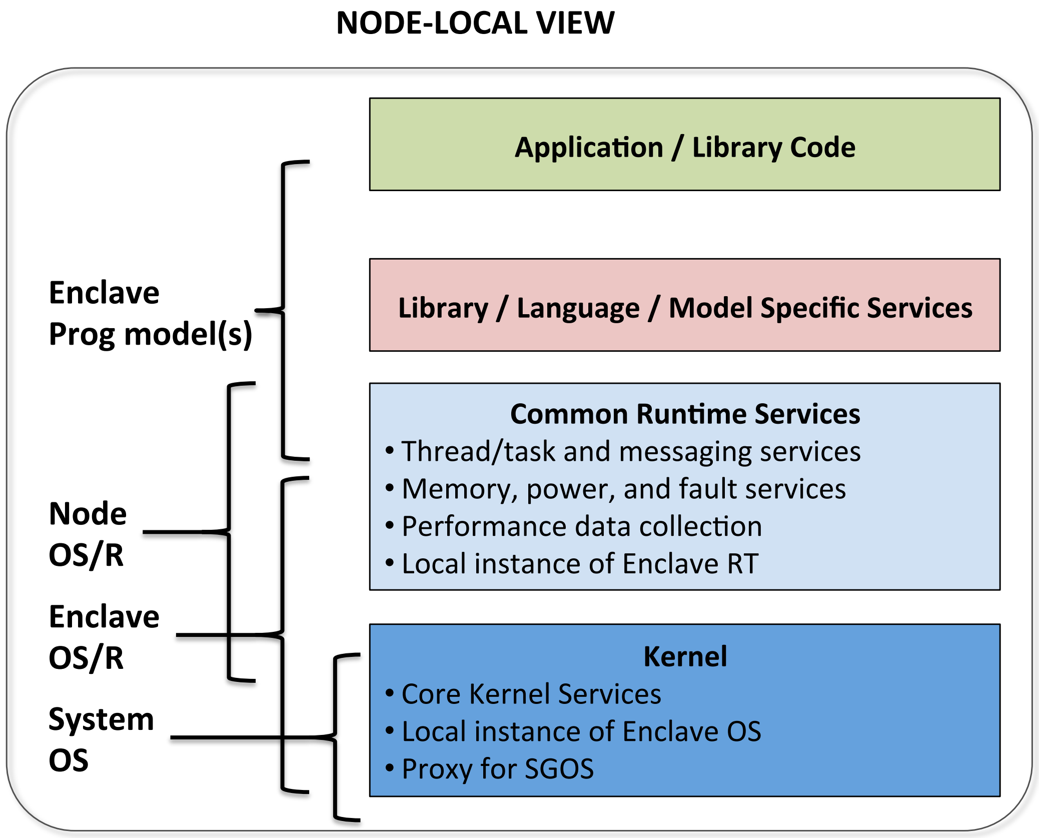
NODE-LOCAL VIEW
Application / Library Code
Library / Language / Model Specific Services
Common Runtime Services
• Thread/task and messaging services
• Memory, power, and fault services
• Performance data collection
• Local instance of Enclave RT
Kernel
• Core Kernel Services
• Local instance of Enclave OS
• Proxy for SGOS
Enclave Prog model(s)
Node OS/R
Enclave OS/R
System OS

Figure 3: Structure of node software.

The enclave interfaces with the SGOS, with other enclaves, and with external agents and services. The SGOS sets up the required interfaces when the enclave is instantiated, connecting the newly created enclave to other enclaves and negotiating the dynamic interface between enclave and SGOS. The SGOS may require the enclave to periodically report on its health and performance. The enclave may request to be informed of various hardware events that are captured by the hardware monitoring subsystem. The enclave also may declare its ability to handle various failures, for example, the failure of a node; or it may require that such failures be handled by the SGOS (e.g., by invoking a restart callback function on a newly established enclave). The interface may include a dialogue for dynamic resource management, allowing the SGOS changing the resource allocation, for example, by reducing power or giving the enclave additional nodes.

Similar dialogues involve the EOS/R and the libraries, tools, and application-specific services. For example, an enclave may register with the SGOS its ability to handle node failures. Handling such failures will involve the EOS/R for recreating the execution environment (e.g., re-establishing communications), possibly replacing the failed node with a spare node; at this point, the EOS/R would invoke callback functions of the various components, libraries, tools and applications, enabling each to repair its state.

4.5 Node View

Figure 3 illustrates the various software layers from the viewpoint of an individual node. The node is controlled by kernel and common services software installed at each node. Some of the software is installed when the node boots, and some is installed when the enclave is configured. Some of the kernel software is the local "representative" (or proxy) of the SGOS. For example, the node may run a process that provides a periodic heartbeat to the SGOS, sends to it periodically performance information, gets informed about global events (e.g., hardware failures), or passes to the SGOS information about enclave events (e.g., normal or abnormal termination). This proxy need not be present at each node: An enclave may have one dedicated "master node" that interfaces with the SGOS; or, for redundancy, it could have two masters, one active and the other passive, with a takeover protocol managed by the global information bus (described below).

Similarly, the node will run software that is the local instance of the EOS/R, handling services that require interaction of multiple enclave nodes: communication and synchronization, reconfiguration after hardware failures (at the local node or at other nodes), load balancing, and so forth. Moreover, the node has a local OS/R component that manages node local resources. Above these layers, language- or library-specific runtimes implement various programming models that can be enclave-global or node-local.

We expect significant hardware heterogeneity, with different hardware dedicated to event-driven (i.e., *reactive*) computations and to long-running (i.e., *transformational*) computations. This specialization can occur at the node level (with different cores used for different functions), at the enclave level (with different nodes dedicated to different functions) and at the system level.Node-level specialization can be exploited by mapping kernel and runtime functions on cores that supports fast even handling and mapping computations on cores that have efficient pipelines for long-running computations.

4.6 Global Information Bus

The proposed architecture requires the services of a resilient, scalable, and efficient *publish and subscribe* (pub/sub) infrastructure, in order to carry information about events and commands across the various components: Information about the health of hardware components has to be carried to the enclaves that use this components, assuming they have registered for this information; information about the health of enclaves may be relevant to storage server, as well as to the SGOS and so on. A similar pub/sub infrastructure may be needed within each enclave, for example, to handle dynamic resource management (load balancing), power management, and failures. A similar infrastructure is needed for interfacing with performance tools and controlling applications interactively (for debugging or steering), although this case is simpler since it requires only a one-to-many communication infrastructure (a tree), rather than a many-to-many.

4.7 Global Object Store

A fast, transactional database or object store is another generic service that could greatly facilitate the design of scalable and resilient systems. Recovery is slowed when the state to be recovered is mutable, is distributed across many sites, and must be consistent across sites. Therefore, to the greatest possible extent, components should be stateless (i.e., have immutable state) or should have centralized state. However, centralizing the location of mutable state may hamper scalability. We need logically centralized but physically distributed state—something that can be provided by a fast object store, possibly implemented by using nonvolatile memory.

5 Interfaces

The proposed architecture requires multiple interfaces across components and functions. Some of them are well established (e.g., bootstrap protocols for installing the initial kernel on each node). Others exist in a limited form on current HPC platforms (e.g., various global kernel services). And some will require new invention.

We describe these interfaces in generic terms, without specifying implementation mechanisms. Some of the interfaces could be implemented by using objects stored in a shared object store (e.g., shared XML files or shared databases). Others could be implemented by using message exchanges (e.g., passed by the global information bus). Still others could have functional implementations, using calls or callback functions. The choice of implementation will depend on the volume of interaction, the communication mechanisms (push vs. pull, point-to-point vs. multicast), requirements for performance and reliability, and so forth.

5.1 Hardware Abstraction Layer to Global (and Local) OS

The HAL interface exposes the capabilities of the underlying hardware to the SGOS, EOS/R, and node OS. This includes node-level interfaces that support functions such as the following:

- Query hardware node configuration
- Bootstrap an initial communication channel and an initial kernel on a node
- Set up user space communication channels
- Query dynamic information (temperature, power, performance metrics, faults and other events)
- Control HW configuration (in particular, power)

Some of these interfaces are already defined by industry standards. Others, such as the interface to control performance counters, are partially defined by libraries such as PAPI [10]. For yet other interfaces (such as drivers setting user space communication channels), no standards exist.

At the SGOS level, the HAL interface exposes information on the health of other hardware components (racks, cooling system, fans, etc.) and enables control of the hardware configuration (power, connectivity) One will need to integrate the HAL interfaces in the general publish and subscribe infrastructure, so as to allow for multiple customers of the data, while avoiding duplication of the sensing and control functions.

5.2 System Global OS External Interfaces

Global OS external interfaces include interfaces and tools to control the system and to effect interactions with external subsystems, in particular the following:

- **Interfaces for SGOS booting.** As systems become more complex and more heterogeneous (e.g., supporting multiple kernel configurations within the same system), the tools and interfaces for system boot must become more open and standard, enabling various subsystems to plug in their own startup procedures and validity checks at appropriate points in the boot sequence

- **Interfaces to monitor and control the hardware configuration from an external control point** (human or automated operator). Before the information is sent out, it will be filtered and processed by the SGOS HMC subsystem; the interfaces enable the SGOS to control which information is captured and how it is processed before being sent out. By "hardware configuration" we mean static configuration (the machine description), dynamic configurations (nodes that are up or down, power levels, etc.), and events (alarms).

- **Interfaces to monitor and control the software configuration from an external control point.** Again, the interface enables one to query current configuration, receive information about alarms, set up various software sensors (e.g., heartbeats or periodic statistics dumps), and so on. The interface should allow the plug-in of monitoring and control interfaces for new subsystems; and it should support multiple clients for monitoring and control, with a clear hierarchy: for example, a fast, automatic response by suitable scripts and a slower response by an operator.

- **Event databases.** We expect that event logs will be replaced by searchable event databases, with standard schemas.

- **Interfaces for remote schedulers.** Also needed are interfaces for batch schedulers and workflow control engines. The interfaces should allow different system resources to be dynamically allocated to different external schedulers. As described before, schedulable resources may include not only compute nodes but also I/O nodes or fractions of shared services, as well as power. The allocation can be dynamic, changing over time.

5.3 SGOS to Enclave

The global operating system allocates resources to enclaves, controls their configuration, connects different enclaves in order to support workflows or provide services, and negotiates the communication and control protocols at execution. Today, these protocols are largely fixed. Future systems will need more flexibility. For example, an enclave may or may not want to be informed of failures of hardware components allocated to the enclave. In the former case, the SGOS will pass the information to the enclave manager(s); in the latter case, the alternatives may be to terminate the enclave or to re-establish an enclave with the same hardware and system configuration and invoke a recovery callback function at the new enclave manager. Similarly, an enclave may or may not want to be able to request or be requested for changes in configuration (more or fewer nodes, more or less power). The enclave may be required to provide a heartbeat or periodic information indicating that the application is making suitable progress (e.g., a heartbeat generated by the application at each iteration). Again, the external interface (heartbeat) should be separated from the internal implementation (the mechanisms generating the heartbeat within the enclave). Further, the enclave OS should have full information on the underlying hardware configuration, including network topology.

5.4 Enclave OS to Enclave RT

To the same extent that different enclaves will have different capabilities for managing power or handling failures, different runtimes, libraries, and applications will differ in their capabilities. Here, again, the EOS has to negotiate with the higher software layers what information they produce and consume and what events they can handle. Consider, for example, a node failure. Such a failure can be handled by the global OS by allocating a new partition, initializing the enclave OS and invoking a recovery callback function at the enclave manager. Alternatively, the enclave may use spare nodes, together with global checkpoint/restart. In this case, the EOS is informed of the node failure; it recreates the communication infrastructure with the spare node replacing the failed node and invokes a recovery callback function to restore application state. The runtime, libraries and applications may be able to recreate the state of a failed node locally, without requiring a global restart. In this case, the enclave OS may invoke the recovery callback function(s) only at the newly allocated node and other nodes that need to participate in the recovery. If the runtime, libraries, and application is able to continue executing with one node less, the enclave OS need not recreate the original communication infrastructure but simply update the running nodes with the information on the missing node.

Note that the capabilities that an EOS will register with the SGOS may depend on the capabilities of the runtime, libraries, and applications: The EOS will register its ability to overcome a node failure only if the application or runtime can do so. Therefore, one will need to properly order the different information exchanges, probably using a top-down pass, followed by a bottom-up pass.

5.5 General Communication and Coordination Services

The proper functioning of the proposed architecture requires the availability of scalable, resilient, and efficient communication and synchronization services. These can be standardized and made available as standard "global information bus services." Among the issues to address are the following:

- The services provided by such information bus: publish and subscribe, ordered broadcasts, consensus protocols, and so on.
- The QoS and resilience guarantees, possibly with differentiated levels of service.
- Security and capability management, to restrict which agents can produce and consume which information (including commands).
- Priorities and conflict resolution schemes, to handle potential conflicts between agents
- Physical configuration management, to control which physical resource are used by different communication channels.

An important issue concerns agent interfaces. An agent (e.g., an enclave) can be represented by one manager, or one active manager and one backup manager, or multiple active managers, and so on. In each case, one needs to specify the contract with the client (e.g., only one manager can issue commands) and the process for enforcing it (e.g., the protocol for replacing the active manager with a backup manager).

The other generic component discussed in Section 4 is a scalable, transactional object store. Our community may be able to reuse technologies developed for fast online transaction processing; but new interfaces are likely to be required, for example, for supporting user space remote accesses and updates.

We discussed the functionality of different components and their interfaces. However, the system will also require performance and resilience specifications. In particular, the larger size of exascale systems will force scalability issues; and the more frequent failure rates will necessitate much faster recovery. While acknowledging the importance of performance and resilience specifications, we leave this to later work.

6 Recommended Strategy / Path Forward

The HPC community needs a well-defined, consistent, and complete set of programming interfaces (HPC-PIs), to enable and support application execution on HPC systems. Developers of runtime systems and tools will know that they can rely on the existence of these services and can be assured of reaching a broad audience. Moreover, system vendors will know the target set of services to be supported and will be free to innovate in the most efficient approaches to providing these services.

Without common HPC-PIs, developers of runtime systems and tools must target the "least common denominator of services that are available on all HPC platforms or must selectively ignore HPC platforms that do not provide services that are deemed important. And without such common HPC-PIs, system vendors must support all the services that might be used by an HPC runtime system or tool, frequently compromising performance.

6.1 Goals

The path to exascale presents several key challenges—power management, resilience, deeper memory hierarchy, and increased parallelism—that fundamentally change the cost trade-offs that have driven the design of operating and runtime systems since the early 1970s. To meet these challenges, the HPC-PIs must provide the following:

- Conceptual framework for implementing resource management policies that extends from an exascale system to the discrete resources used by an application
- Support for the dynamic characteristics of resources (e.g., degraded performance in the presence of a partial failure)
- Support for the management of power consumption by an application;
- Support enabling applications to manage extended memory hierarchies
- Support for the creation and management of lightweight execution contexts, to enable a much higher degree of application parallelism.

The system management strategy used in HPC systems (space sharing) means that many of the resource management issues traditionally relegated to the operating system can be left to the runtime system, where they can be tailored to the specific needs and expectations associated with the programming model supported by the runtime system.

- The lowest-level interfaces *should* be as thin as possible (e.g., initialize the hardware, and get out of the way) to enable effective management of the resources by higher levels of the software stack (e.g., the runtime system) that are closer to the application.
- Drivers for specialized hardware *should* be easy to accommodate within the overall architecture of the system.
- Many functions that have traditionally been embedded deep within the operating system (e.g., hardware monitoring and control) *should* be exposed to the runtime system and possibly the application.
- Interfaces *should* be designed for change, embracing "open designs that support selective replacement of components.

- “Loose definitions of the interfaces may be needed to support future change or when there is close collaboration on both sides of the interface.
- Hardware transparency, the ability of the interface to reflect hardware costs accurately, is *desirable.*

6.2 Proposed Activities

The development of the required HPC-PIs will require distinct steps. While these steps will likely overlap or proceed in a spiral fashion, each step is expected to produce a concrete artifact (e.g., a document or prototype implementation).

High-level architecture Develop components, functions, interfaces, and their interaction.

Interfaces Define possible partitions of the architecture into subsystems. Standardization of interfaces is critical when subsystems could be supplied by different vendors.

New Interfaces Identify where new interfaces are needed (and where existing ones can be reused) and define the new interfaces.

Metrics Develop approximate performance targets

Prototype Develop prototype/reference implementations of the architecture; validate interfaces.

Product Develop, test, deploy, and support the software stack.

6.3 Migration Path from Current to Future Architecture

The proposed architecture extends and generalizes the current software stack of supercomputers. The architecture described in Figures 1 and 2 is sufficiently generic to match the current organization of IBM or Cray systems. In some cases, however, we have collected together subsystems that are now disjoint. For example, the SGOS contains the hardware monitoring and control infrastructure, as well as the resource management infrastructure; and the node common runtime includes thread management, low-level communication services, and memory management. These are supported today by distinct libraries, and such a separation may extend into the future. However, the clustering we propose aims at raising awareness to the tight interaction among these different components. For example, fault handling requires a tight interaction between hardware monitoring and resource management; asynchronous computation models require tight interaction between the communication subsystem and task scheduling; and locality requires a tight interaction between memory management and task scheduling. Thinking about these layers in a more integrated manner will ensure they interact properly.

However, the evolution needed from where we are now to what we have outlined requires several developments.

6.3.1 Cleaner separation of functionality across components

For example, no clean separation currently exists between the SGOS stack and the EOS stack. Such a separation would allow running different parts of the machine with different node kernels (for testing ahead of upgrades or for customization). It would also provide a good separation of concerns: For example, an exascale system need not run many more jobs than a current petascale system but would run bigger jobs; the scalability concern can be handled within the enclave stack.

Another example is provided by the current structure of many batch schedulers that handle both the allocation of resources and the actual management of these resources: job loading and health monitoring. A separation between resource manager and scheduler enables the use of external, (mostly) platform independent schedulers.

6.3.2 New components

One major new component is the general publish and subscribe communication infrastructure that we posit. Currently, different components have each a different logical communication infrastructure, usually with a tree topology. The hardware monitoring and control subsystem is likely to use out-of-band communication using a separate ethernet network; the resource management infrastructure typically uses in-band communication on the high-speed network. This separation is seen to enhance the reliability of the monitoring infrastructure. Future approaches to fault tolerance will require a closer interaction between the two subsystems, with communication between them occurring not only at the top.

We propose an integrated logical communication infrastructure; a proper mapping of it onto the physical infrastructure will enable a trade-off between reliability and performance.

6.3.3 New functionality for existing components

Most changes needed will be in the form of new functions added to existing components. For example, more resources will become managed by the resource manager component of the SGOS. This clearly is the case for energy, but it also is the case for I/O bandwidth or I/O storage.

Moreover, allocations that are static become dynamic: these may include changes in power allocation, changes in the number of nodes allocated to an enclave, and changes in I/O resources.

Most important, failures are reflected up to the topmost and most local level that has registered a capability to handle this type of failures: The proposed architecture makes no a priori assumptions about the mechanisms for fault handling and the software layers that provide these mechanisms. Notifications of failures are routed to the software components that registered their ability to handle the failure and that are affected by that failure.

6.4 Activities and Roadmap

The general principles detailed above need to be translated into concrete actions. We outline below a set of key activities; these activities will require the involvement of DOE, vendors, and, in some cases, the academic community.

Agreement on Architecture

The goal of this activity is to agree on the component architecture for the exascale software stack and the division of functions across components in this stack. As part of this activity, each nominal component may be subdivided into subcomponents. This exercise should also involve a detailed mapping of the current product offerings onto these components, in order to identify mismatches.

Agreement on levels of technology readiness for the different components and their interfaces

A rough outline of the level of readiness for various technologies is provided in Appendix A. This list needs to be further refined and to become a continued activity of a technical group within DOE, interacting with academia and vendors.

Agreement on desired levels of standardization

The purpose of this activity is to develop a common view of interfaces that will be standardized, as distinct from interfaces that will be proprietary. This activity may not lead to a consensus, but it will indicate areas of agreements and disagreement among the various players. We discuss standards in more detail in Section 6.6.

Agreement on Timetable

We expect that each technology will evolve through phases that include initial research, prototype development, standardization of interfaces, deployment on production platforms at the 100 PF performance level, and deployment at exascale.

This report includes an initial proposal for each of the four "agreement" activities: the overall system structure, their level of readiness, desired standards, and timetable. We believe this initial document is sufficient to drive decisions about DOE research initiatives in OS and runtime in 2013. However, we expect that this report will be a live document that continues to be updated as we learn more from research and from the evolution of vendor technology and as the exascale activities continue to evolve.

6.5 Research Plan

The challenges listed in Section 2 and the estimates of TRLs in Appendix A provide a roadmap and a prioritization for DOE's research in extreme-scale OS and runtime. While we do not have the entire picture, we have clearly identified high-priority items with a long lead time.

We note, however, a chicken and egg problem: On one hand, research in OS and runtime will be more easily assimilated in future products if it fits in the overall architecture proposed in this document and uses the proposed interfaces; on the other hand, research may be needed to determine the effectiveness of the proposed structures and interfaces. Thus, research and standardization efforts need to proceed in parallel, with continued interaction between the two efforts.

We propose the following interlock between the various activities:

- DOE-funded research projects will be expected to conform to the nominal architecture outlined in this document, in terms of how various functions are packaged into components and how interfaces are expressed. Projects that want to use a different structure will need to justify their choice; the trade-off between increased research freedom and increased ability to use the research product will be taken into consideration in funding decisions.

- Projects that develop components that interact with one of the proposed interfaces will need to match the proposed interface, if it is defined. If it is not fully defined, then the project will participate in its standardization.

- DOE will fund teams to develop reference implementation of any of the proposed components and interfaces. To the extent possible, the team developing an initial, reference implementation will be the same team that pursues research related to that component or interface.

6.6 Standardization Activities

The different interfaces described in Section 5 need to be standardized whenever different organizations (vendors, consortia or public institutions) will develop the software and firmware on each side of the interface. Some level of standardization can help even when the interface is internal, since it facilitates the division of labor across different development teams. Nevertheless, one can expect that vendors may oppose the standardization of some interfaces, for various reasons:

- The level of vertical integration differs among vendors; hence vendors will differ on the interfaces that need to be exposed and standardized.
- Vendors will be reluctant to expose "internal" interfaces, since doing so may restrict their freedom of action and may expose proprietary information that provides them with a competitive advantage.
- Vendors may be reluctant to expose low-level control interfaces as a misuse of such interfaces may harm the system or lead to bugs that are hard to diagnose.
- Premature standardization may hamper research and development.

These concerns may be alleviated in a variety of ways. Premature standardization can be avoided by carefully choosing the time for and level of standardization. As a rule, standards should be defined only when previous R&D has shown the viability and effectiveness of the proposed approach and when a reference implementation exists. Some of the interfaces we discussed (e.g., a generic publish and subscribe infrastructure or interfaces for tools) may be ripe for standardization; others should wait for further research. This categorization and timeline should be established in consultation with industry.

The risks of exposing low-level interfaces can be reduced by building appropriate "circuit breakers": for example, the user may manage the power consumption of a component, but the component will be stopped (and an appropriate exception will be raised) if the component overheats.

An important consideration is that standardization can be more or less prescriptive. For example, the HAL interface is likely to be hardware specific. While one may not be able to standardize all verbs and objects, one could standardize how each type of information is exposed to the global OS (configuration files, periodic polling or periodic pushing, callback function, etc.); and standardize the major categories of sensors and actuators.

Different levels of exposure to standards can also exist. We propose the following categorization.

- **Standard Interface**
  - The interface is defined by a public document maintained by a standard body.
  - The interface is implemented and supported by multiple vendors.
  - An implementation of one side of this interface will support any software that runs on the other side and uses the interface correctly (correct use is defined by the standard).
- **Open Public Interface**
  - The interface is defined by a public document.
  - The interface is implemented and supported by at least one vendor.
  - The implementations support any software that runs on the other side and uses the interface correctly (correct use is defined by the standard).
  - The vendor guarantees continued support to the interface for an acceptable period of time.
- **Proprietary Public Interface**

  - The interface is implemented by at least one vendor.
  - The vendor is willing to document and provide access to the interface, under a suitable nondisclosure agreement.
  - The vendor is willing to support a system where third-party software runs on the other side of the interface (some restrictions may apply on how this software is verified and tested).
  - The vendor guarantees continued support to the interface for an acceptable period of time.

- **Private Interface**
  - None of the above

For example, Microsoft provides a public interface for vendors that develop Windows device drivers, but it requires these device drivers to pass certain formal verification procedures. Discussions with vendors should lead to a decision about which interface falls in which category.

Decisions on standardization also should be negotiated with vendors and be taken by consensus, to the extent possible. However, one should recognize that the interests of DOE may not be fully aligned with the interests of each vendor. DOE may need to push for more openness, as part of its funding of exascale R&D and its procurements.

6.7 Governance

To pursue the activities outlined in the preceding sections, we recommend that a DOE "nexus" be created to handle the following responsibilities:

- *DOE and community coordination*: The OS/R community can benefit from regular workshops and maintaining up-to-date documents, such as this one, that outlines current challenges and promising research directions. The community can also share challenge problems and research results in focused workshops. A DOE OS/R nexus could help coordinate activities with other DOE research activities, such as the co-design centers and X-Stack projects.

- *OS/R testbeds*: In order to develop novel OS/Rs, early access to prototype and first-access hardware and scalable testbeds is vital. Such testbeds have specialized software frameworks that permit OS/R scientists to load their own operating systems running as a privileged ("root") user. Scientists also must have access to out-of-band management interfaces to control and monitor experimental hardware, including external power monitoring, "console" interfaces, and remote rebooting. A DOE OS/R nexus could coordinate and/or supply testbeds for low-level system software. Such a nexus also could encourage and coordinate INCITE or ALCC proposals for testing extreme-scale OS/R components at scale.

- *API coordination and standards*: Standard interfaces help support a rich software ecosystem. The DOE OS/R nexus could help build teams to participate in coordinating APIs and standards. Committees could be formed as needed to work with vendors and international partners in areas such as power management, low-level messaging layers and fault response.

- *Software and benchmark repository*: Sharing benchmarks, benchmark results, and community-developed reference implementations can foster community collaboration. A DOE OS/R nexus could provide an easy mechanism for sharing results and code, resulting in broader experimentation and testing of new software components as well as clear paths for novel software packages to be made available to vendors and other research teams.

6.8 Business and Sustainability Model

Strong engagement of vendors is essential to the successful deployment of exascale platforms. We therefore present here a model for vendor engagement.

6.8.1 Vendor Engagement

We expect that much of the software will be vendor supported. In particular, vendors will want full control of software they deem critical to the proper functioning of their system. On the other hand, we do not believe that an R&D activity that involves only vendors can be successful, for a variety of reasons:

- Many technologies are still in the research phase. DOE labs have the skills to successfully pursue such research and to make it available to multiple vendors.
- Significant uncertainty remains about the set of vendors that will deploy exascale platforms more that a decade from now. Some of the potential future vendors currently have limited knowledge about HPC software.
- Software developed in DOE labs has been successfully adopted by vendors in the past, including low-level libraries (MPiCH, PAPI). We have little reason to believe the future will be any different.
- System software developed in DOE labs has been an essential "Plan B" for large platforms in the past (e.g., SUNMOS for the Intel Paragon). The large uncertainties of exascale designs mandate that the capability to deploy such Plan B system software be maintained.

We expect vendors to be involved at various levels in exascale OS and runtime:

- Continued activities to refine the proposed exascale software architecture and to standardize interfaces, wherever possible
- Joint research to prototype key components
- Technology transfer from DOE research projects to vendor products
- Joint activities to define, develop, test, and validate capabilities for exascale software.

6.8.2 Engagement Principles

The engagement with vendors should be driven by the following principles:

- Potential vendors *must* be included in the entire process.
- Vendor differentiation *should* be based on quality of implementation rather than functionality whenever possible.
- Prototype (reference) implementations *should* be developed during the design of the interfaces to demonstrate completeness, consistency, and potential for efficient implementation.
- Support for the proposed interfaces *should* be a requirement of every DOE system procurement.

- Reference implementations *should* be developed under an open-source license that provides as much flexibility to potential vendors as possible (e.g., a BSD-style license).
- Interfaces *should* be partitioned to support a rich ecosystem of potential suppliers (e.g., the hardware vendor may not be the ultimate supplier of the operating system, and several vendors may supply runtime systems or tools).

U.S. vendors will want to supply systems to Japan or Europe; foreign customers of extreme-scale systems manufactured in the United S will want to be involved in the evolution of extreme-scale computing technology. Furthermore, the significant development costs of exascale technologies may require an international collaboration, in order to develop these technologies in a timely manner. Therefore, international collaborations are likely to be part of this effort.

6.8.3 Scale-Down and Scale-Out to Enable Scale-Up

One hopes that a software architecture developed for extreme-scale systems will also benefit a broader range of systems. If so, vendors can justify investments in the extreme-scale software stack by the needs of a broader market than the one presented by extreme-scale computing.

Such a hope is justified by two factors:

- The cost of hardware continues to drop, so that a performance level that costs hundreds of millions of dollars today will cost tens of millions in a decade. Furthermore, as clock speed does not improve, a low-cost cluster in a decade will have the same level of parallelism as a high-end supercomputer has today. Therefore, one can expect a continued trickle-down effect, with technologies developed for the very high end becoming relevant for lower-cost platforms over time.
- Significant commonality is expected in the technologies deployed for high-end computing systems that support large-scale simulations, and big-data systems that support large-scale analytics. The latter type of platforms represents a growing market.

However, one should realize that both arguments have limitations.

- Moore's "law" is expected to inflect as we reach feature sizes of 10^7 nm, with future progress being slower and more expensive. If the underlying microelectronic technology plateaus in the next decade, then exascale systems built in the next decade will not have trickle-down benefits.
- The commonality among various large systems is easy to exaggerate. Even though companies such as Google, Amazon, or Microsoft build cloud computing facilities whose compute power easily exceeds that of the largest deployed supercomputers, no common technologies have been developed to support both types of platforms. Even if common hardware will be possible, significant differentiation in software requirements is likely.

Interfaces that will be used at the very high end will need to be supported on more modest platforms, since these will be used for developing the most scalable applications. Users will want the same interfaces in all the platforms they use, both at the very high-capability end and at the more modest capacity level. Conversely, whenever existing interfaces can provide the right functionality and performance at extreme scale, they should be reused. However, it is wrong to assume that technologies developed to leverage extreme-scale computing platforms will benefit more modest platforms or platforms designed for other application domains. More likely, much of the extreme-scale software stack will be unique to extreme-scale computing. Plans for the development of this stack must be consistent with such a scenario, in which the following considerations are important:

- The HPC interfaces *must* be supported on commonly available platforms, most likely on top of commonly used operating systems (e.g., Linux). For example, while many MPI implementations are optimized to use high-performance networks, MPI is also available on top of TCP/IP.
- Application developers *should* be encouraged to use the HPC interfaces and not use additional functionality that might be available on smaller systems.

6.9 Facilities

The development of HPC software is always hampered by the lack of adequate testing facilities. Indeed, software for a new supercomputer usually is not tested at full scale until the supercomputer is deployed at the customer site: It is too expensive, or plainly impossible, for vendors to assemble a system of the same scale in their facilities. As a result, it often takes more than a year before a system is fully functional and fully accepted.

Similarly, research in HPC software, especially system software, is hampered by the lack of platforms where such software can be tested at a reasonable scale: Supercomputing centers are reluctant to allow the deployment of experimental system software on their machines, since such software often causes crashes.

Modeling, simulation, and emulation can partially alleviate this problem, but only if supported by platforms that are within one or two orders of magnitude of exascale. DOE should fund the establishment of large system software testbeds that are tracking the performance of top-capability systems within one of two orders of magnitude. One possibility is to reuse systems that are being decommissioned as system software testing platforms.

The management of such platforms differs significantly from the management of an application platform. In particular, one will want to support multiple concurrent experiments with no risk of one experiment causing the entire system to crash. And one will want to provide full control of a system partition to a remote user. Leveraging the experience gained from previous deployments of such experimental platforms such as *Chiba City* (now decommissioned) and *Breadboard* [11] at Argonne National Laboratory will be critical. The Breadboard system provides a capability not provided by any other testbed system within DOE. Users can log into the Breadboard system, allocate nodes, and completely overwrite the operating system and disk image from scratch. The system provides not only queuing, but isolation, such that collaborators from around the world can gain privileged (root) access to the hardware for low-level system software research.

6.10 Timeline

We outline below a rough timeline for exascale firmware—assuming deployment in 10 years

Years 1–3: Research of core technologies and standardization

Years 4–6: Testing and validation of good-quality prototypes

Years 7–10: Industrial development

This timeline is schematic. On the one hand, research and standardization efforts will continue beyond year 3; on the other hand, technology that is more mature will deployed before year 10. However, it is essential to start research as soon as possible on technologies that are not mature: industry will want to execute a full development cycle on technologies that are essential to the functioning or exascale systems; and industry will adopt only research technologies that have been demonstrated at reasonable scale and shown to perform well and be robust.

Bibliography

References

[1] Cesar co-design center. https://cesar.mcs.anl.gov.

[2] Exact co-design center. http://exactcodesign.org.

[3] Exmatex co-design center. http://codesign.lanl.gov/projects/exmatex/index.html.

[4] Computer architecture laboratory for design space exploration. https://crd.lbl.gov/groups-depts/future-technologies-group/projects/cal-computer-architecture-laboratory-for-design-space-exploration/.

[5] Execution models for exascale. https://sites.google.com/site/executionmodelsforexascale/public.

[6] Quantifying overhead in today's execution models. https://sites.google.com/site/executionmodelsisilbnl/.

[7] Beyond the Standard Model: towards an integrated methodology for performance and power. https://sites.google.com/site/beyondthestandardmodelbsm/.

[8] Evolving MPI to address the challenges of exascale systems. https://collab.mcs.anl.gov/display/mpiexascale.

[9] 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. IEEE, 2007.

[10] Performance application programming interface. http://icl.cs.utk.edu/papi/, 2012.

[11] A testbed for exploring operating systems and low-level system software. http://wiki.mcs.anl.gov/radix/index.php/Breadboard.

[12] United States Department of Defense. Technology readiness assessment (TRA) guidance. http://www.acq.osd.mil/chieftechnologist/publications/docs/TRA2011.pdf, April 2011.

A Levels of Readiness

We show in the table below the DoD definition of *Technology Levels of Readiness* (TRL) [12]. While this analysis may be too fine grained for our purpose, it provides a useful starting point.

Table 3: Technology Readiness Levels in the Department of Defense (DoD)

Technology Readiness Level	**Description**	**Supporting Information —**
1. Basic principles observed and reported	Lowest level of technology readiness. Scientific research begins to be translated into applied research and development (R&D). Examples might include paper studies of a technologys basic properties.	Published research that identifies the principles that underlie this technology. References to who, where, when.
2. Technology concept and/or application formulated	Invention begins. Once basic principles are observed, practical applications can be invented. Applications are speculative, and there may be no proof or detailed analysis to support the assumptions. Examples are limited to analytic studies.	Publications or other references that out-line the application being considered and that provide analysis to support the concept.
3. Analytical and experimental critical function and/or characteristic proof of concept	Active R&D is initiated. This includes analytical studies and laboratory studies to physically validate the analytical predictions of separate elements of the technology. Examples include components that are not yet integrated or representative.	Results of laboratory tests performed to measure parameters of interest and comparison to analytical predictions for critical subsystems. References to who, where, and when these tests and comparisons were performed.
4. Component and/or breadboard validation in laboratory environment	Basic technological components are integrated to establish that they will work together. This is relatively low fidelity compared with the eventual system. Examples include integration of ad hoc hardware in the laboratory.	System concepts that have been considered and results from testing laboratory-scale breadboard(s). References to who did this work and when. Provide an estimate of how breadboard hardware and test results differ from the expected system goals.

5. Component and/or breadboard validation in relevant environment	Fidelity of breadboard technology increases significantly. The basic technological components are integrated with reasonably realistic supporting elements so they can be tested in a simulated environment. Examples include high-fidelity laboratory integration of components.	Results from testing laboratory breadboard system are integrated with other supporting elements in a simulated operational environment. How does the relevant environment differ from the expected operational environment? How do the test results compare with expectations? What problems, if any, were encountered? Was the breadboard system refined to more nearly match the expected system goals?
6. System/subsystem model or prototype demonstration in a relevant environment	Representative model or prototype system, which is well beyond that of TRL 5, is tested in a relevant environment. Represents a major step up in a technologys demonstrated readiness. Examples include testing a prototype in a high-fidelity laboratory environment or in a simulated operational environment.	Results from laboratory testing of a prototype system that is near the desired con-figuration in terms of performance, weight, and volume. How did the test environment differ from the operational environment? Who performed the tests? How did the test compare with expectations? What problems, if any, were encountered? What are/were the plans, options, or actions to resolve problems before moving to the next level?
7. System prototype demonstration in an operational environment.	Prototype near or at planned operational system. Represents a major step up from TRL 6 by requiring demonstration of an actual system prototype in an operational environment (e.g., in an air-craft, in a vehicle, or in space).	Results from testing a prototype system in an operational environment. Who performed the tests? How did the test compare with expectations? What problems, if any, were encountered? What are/were the plans, options, or actions to resolve problems before moving to the next level?

8. Actual system completed and qualified through test and demonstration.	Technology has been proven to work in its final form and under expected conditions. In almost all cases, this TRL represents the end of true system development. Examples include developmental test and evaluation (DT&E) of the system in its intended weapon system to determine if it meets design specifications.	Results of testing the system in its final configuration under the expected range of environmental conditions in which it will be expected to operate. Assessment of whether it will meet its operational requirements. What problems, if any, were encountered? What are/were the plans, options, or actions to resolve problems before finalizing the design?
9. Actual system proven through successful mission operations.	Actual application of the technology in its final form and under mission conditions, such as those encountered in operational test and evaluation (OT&E). Examples include using the system under operational mission conditions.	OT&E reports.

Most, if not all the technologies discussed in this report are at TRL 1 or higher; none are at TRL 8 or 9, since no exascale system is deployed today. The goal of the exascale R&D in the coming 3-5 years is to bring all major components to TRL 5 and above.

We provide below a first (very rough) list of components with low levels of readiness (TRL 2/3); we expect that these estimates will be further refined in interaction with industry and academia.

Hardware Abstraction Layer The least developed components here are

1. The ability to establish operating bounds for hardware – e.g., control of power consumption, turning off components, initializing memory configuration, etc.
2. A generic interface for controlling user-space communication (NIC-HAL).
3. The ability to control what events are reported by HW

All three items have exascale unique aspects; the NIC-HAL is largely HPC-specific.

System Global OS The least developed components are

1. Coordinated management of different resources: nodes, network bandwidth, power, I/O bandwidth. etc. Currently only nodes are an allocatable resource. As multiple resources become allocatable, the selection of an optimal allocation becomes quite complex. The need for reacting to changes in hardware (in particular, failures) means that one will need to design feedback loops that drive the system into a good regime (control theory, introspection, adaptation...), rather than using centralized allocators.
2. Flexible negotiation protocols that enable to discover the abilities of the various system layers (e.g., their ability to handle hardware failures) and set up proper event handling chains through the software stack.

Enclave OS/R Current system stacks do not distinguish between enclave OS and system global OS; need to develop a hierarchical approach with a clearly defined division of labor between GSOS and EOS

Node OS/R Nodes will be heavily "NUMA" and will provide multiple paths from the external network to the internal, NUMA, memory system. They will be heterogeneous and will support fine-grain power management for different node components. Many of the OS/R capabilities needed to handle such nodes are lacking. In particular:

1. Run-time mechanisms for enhancing locality by coordinating data location, thread location and NIC location.
2. Message-driven scheduling for latency hiding.
3. On chip power management
4. Co-design of hardware and node OS/R to enable resource management in user mode.
5. Co-design of chip fault detection/correction/isolation mechanisms with node OS/R resilience mechanisms.
6. OS/R support and leverage of heterogeneity

Global Information Bus The design of a scalable, resilient pub/sub structure that can also control the order in which events are delivered and handled requires major invention.