MultiLibOS: An OS architecture for Cloud Computing

Dan Schatzberg, James Cadden, Orran Krieger, Jonathan Appavoo
Boston University

1 Introduction

Cloud computing is resulting in fundamental changes to computing infrastructure:

- Rather than purchasing their own servers, companies are instead leasing capacity in datacenters provided by Infrastructure as a Service (IaaS) cloud providers.
- Increasingly, these platforms are created using fully integrated datacenter scale systems rather than aggregating commodity network, compute and storage.
- IaaS providers charge their tenants by usage [6] which increases demand for highly elastic platforms.
- Usage of IaaS platforms will increasingly be dominated by scale out applications than can elastically span many nodes.

These changes in the computing infrastructure have not resulted in corresponding changes to operating systems. Are changes required to traditional OSes or can the challenges be efficiently met by higher level middleware? Are there system software techniques to allow applications to efficiently utilize the parallelism of integrated datacenter scale systems? Are today's OSes the right building blocks for constructing elastic applications on IaaS platforms? If new OS techniques, primitives and abstractions are useful, can they be introduced in a way that allow applications to be incrementally modified to use them?

In this paper we discuss some key changes we see in the computing infrastructure and applications of IaaS systems. We argue that these changes enable and demand a very different model of operating system. We then describe the MulitLibOS architecture we are exploring and how it helps exploit the scale and elasticity of integrated systems while still allowing for legacy software run on traditional OSes.

2 Key Changes

In this section we describe important changes that are happening to IaaS infrastructure and applications and discuss why they justify a new OS architecture.

Datacenter-scale systems

IaaS providers that need to provision datacenters will increasingly purchase highly-integrated datacenter scale systems. An integrated system, designed by a single vendor to operate at scale, can have much better aggregate price, performance and manageability than one composed of commodity network, compute and storage.

The rise of initiatives like vBlock [2] from VMware, Cisco and EMC, and similar efforts from other consortia illustrate how service providers want to buy an integrated system even if it consists of an aggregate of commodity parts. Hardware systems that illustrate the trend towards more highly integrated datacenter scale systems include HP's Moonshot [3], IBM's BlueGene [9], and SeaMicro [20] systems.

Integrated systems are carefully balanced to ensure efficient aggregate capacity. To optimized density, price and power consumption, individual nodes tend to be less powerful and the network is more tightly integrated into the individual nodes. To see how this changes system trade-offs, contrast the memory to network performance Google cites out of their

commodity systems to IBM's BlueGene/P systems. Google cites three orders of magnitude increase in latencies from local DRAM to DRAM on a different node in the same rack [8]. Meanwhile, an IBM Bluegene/P system provides latencies and bandwidth between two nodes in the system that are less than an order of magnitude worse than the latencies and bandwidth to local DRAM [17]. The BlueGene system also provides many more communication capabilities, e.g., RDMA, and collective operations; all while having better price and power efficiency for the same compute capabilities.

Elasticity

Since tenants pay for capacity from an IaaS provider on a consumption basis, they are increasingly concerned with the efficiency of their software. Efficiency is far more visible to a customer that pays for every cycle than those that purchased computers large enough to meet peak demand, especially if those computers are idle much of the time. This increased focus on efficiency is driving IaaS providers, hardware architects, systems software designers to build highly elastic platforms. Such platforms allow applications to quickly consume resources when demand increases, and relinquish those resources when demand drops. An example of this trend is the exploration of 'power proportional'[14] systems where the aggregate power consumption is more directly and smoothly proportional to the use of the system.

Isolation

For security and auditability IaaS providers isolate their tenants at a very low level (either physically or virtually.) Individual tenants own and manage their own "logical" computers within the IaaS cloud.

This enables the customer to supply a custom software stack (including OS and middleware) on which to implement their applications. This feature is critical from a business perspective to enable customers with existing corporate policies and controls to be supported. From a performance perspective, it means that the OS can be tuned to the application and platform characteristics. For example, low-level access to a hypervisor allows the application to add and remove cores and memory as the demands on the application changes. Direct interconnect access allows programmers to explore the trade off in exploiting the features and communication models the hardware interconnects provide. In this way, if efficiency is of higher value than development costs (and/or protocol compatibility) custom application level protocols can be directly implemented to exploit features of the interconnect such as RDMA or scatter gather. While we don't expect all applications to exploit this kind of low-level control, high performance datacenter applications will likely warrant this extra effort [21].

Scale-out Applications

Increasingly, applications deployed in these systems are designed to be scale-out applications that can consume many nodes. These distributed applications require programming models not only for scaling and elasticity, but also for fault containment and fault tolerance.

With clusters of commodity hardware, the network latency is typically high enough that the performance of todays system software is adequate. However, in highly-integrated systems, where the relative latency of the network is typically much lower, the system software have much more of an impact on application performance.

The general availability of highly elastic IaaS services and the characteristics of highly integrated systems may well enable new classes of more demanding scale-out applications. What will happen when a scientist can acquire 1000 nodes for a few seconds to solve an important computational task at a cost of only pennies?

Heterogeneity

IaaS datacenters are intrinsically heterogeneous for a number of reasons: 1) non-uniform latencies and bandwidth exist between different parts of the datacenter and even parts of large-scale systems in a datacenter, 2) large datacenters may have many generations of hardware, each with different quantities and characteristics of processing, memory, and net-

working, 3) different (integrated) systems may have different properties, e.g., networking properties like scatter gather and RDMA, or different compute accelerators like GP-GPUs.

Heterogeneity also makes more economic sense in a large datacenter shared by many customers. Even if an exotic system gives order of magnitude performance advantages for some applications, such systems are difficult to sell to individual customers that may have a limited number of such applications and don't want to manage an exotic system. On the other hand, IaaS providers with many tenants are more likely to drive high utilization of the system and hence justify acquiring (and managing) heterogeneous systems.

Application developers will want to span these different types of systems to take advantage of the hardware characteristics and price differences for the appropriate parts of the applications. This, however, will come with complexity, development and maintenances costs. OSes may have a role in mitigating these costs by providing appropriate abstractions and primitives.

3 The Role of the OS

Traditionally we have looked to operating systems to help cope with the complexity associated with hardware – alleviating the burden from applications and middle-ware. Cloud computing, however, has made this very challenging and, to a large extent, OS technologies have failed to provide primitives that help applications scale across the diverse levels while mitigating subtleties of the hardware. Perhaps the best indication of this is the trend of virtualization to make all hardware look like a 1980's flat-network cluster in which an individual OS only manages a small static amount of resources – providing parallel applications with nothing more than traditional processes, threads and LAN based communication primitives.

Most scale-out applications rely on middleware that run on top of legacy operating systems. If performance is important, the developer must discover, adapt, and exploit the system's underlying features and characteristics despite the properties imposed by the hypervisor, legacy OS and middle that the application is implemented on top of.

We believe that much greater efficiency will be achieved if the enablement is done at the operating system level. Changes in our model of an OS will be critical if we want to rapidly grow and shrink resources, exploit very low latency communication between nodes, and enable complex heterogeneous systems.

In this section we first describe the requirements we believe that future datacenter scale system software should meet. We then discuss some of the non-requirements; requirements that existing OSes have that are not necessary in these systems.

Scale: System software will need to be distributed and able to scale across thousands of nodes. It will need to provide services to applications to simplify the task of developing scalable distributed applications.

Elastic: It will be necessary for operating systems to grow and shrink the cores, memory and networking resources used as demands on the application changes. Applications wishing to take advantage of fine grained elasticity will require operating systems that are able to boot on new hardware in milliseconds.

Fault tolerance: If the operating system is able to run on thousands of nodes, the mean time to failure implications will make it important for many applications to have an OS that is resilient to failures. In addition, the OS should provide services to applications to allow them to be resilient to those failures.

Heterogeneity: An application should be able to exploit different accelerators and exotic systems for different parts of the application. This will include support for multiple ISAs as well as low level optimizations that depend on different hardware characteristics such as latency, bandwidth of different resources, and networking characteristics.

Customizability: Our experience in building high-performance system software for large-scale

shared memory multiprocessors [13, 16] is that there is no right answer for parallel applications. To achieve high performance, the operating system needs to be customized to meet the needs of that application. As the performance characteristics of networking improves, those optimizations will need to happen at very low levels, e.g., in the exception path of a network packet.

Legacy support: A model that requires wholesale changes to existing software will cover only a tiny fraction of the relevant applications. Hence, it is critical to support existing software while providing a path for those applications to be incrementally modified to exploit more features of the underlying platform. The operating system should allow existing applications to run unmodified. Only those parts of the application that are likely to benefit should be modified to exploit the new capabilities of the cloud.

While the OSes for datacenter scale systems have many challenges, it turns out that three of the major objectives that existing legacy operating systems were designed to meet are either relaxed or eliminated.

Firstly, distributed applications that run across many nodes do not need to fully support traditional OS functionality on all the nodes. These applications can instead partition some nodes to run full legacy operating systems while designating other nodes to run low-level computational tasks that only require basic OS functionality.

Secondly, much of the burden to support multi-tenant applications is alleviated. Existing operating systems need to support many tenants on a single instance of the operating system. In this new environment, security is provided outside of the operating system, isolating applications on different nodes each running their own instances of the operating system. This eliminates the need for many low level checks and accounting in the software, and reduces the requirement for internal barriers between trusted and untrusted code, since the entire system is isolated. As we will see, this has both complexity and performance advantages.

Thirdly, much of the burden associated with balancing and arbitrating competitive use of resources across multiple applications is removed from the operating system. We expect many nodes to be dedicated to a single application/OS, rather than running many applications on a single instance of the OS. Much of the complexity of existing operating systems in scheduling, memory management, etc. are hence eliminated. This, again, results in both complexity and performance advantages.

These three simplifications: 1) not having to support full OS functionality on every node, 2) not having to support multiple tenants on a node, and 3) not having to support multiple applications on a node, make it possible to adopt a vastly different architecture for a datacenter scale operating system that can meet the major challenges/requirements of a modern hardware platform.

Traditional concerns about competition and fairness are largely eliminated while optimizing the operation per dollar cost of a single critical application run on an IaaS platform.

4 The MultiLibOS

A MultiLibOS is a single tenant, single application distributed operating system composed of per-node library operating systems. We call the component of the application/OS that runs on a node a *shard*. Normally, most of the application code runs just on one or more *master* shards; *slave* shards act as accelerators for application code that needs to be executed in a distributed fashion. Shards may be *process based*, where they run as a process on an existing OS. They may also execute *bare-metal*, where they control a node without any underlying legacy OS.

In this section we discuss each part of MultiLibOS architecture, then discuss how they help us meet our requirements of datacenter scale system software.

The Library OS model

The per-shard system functionality is provided as a library or set of libraries directly linked into the per shared application code address space. That is, the

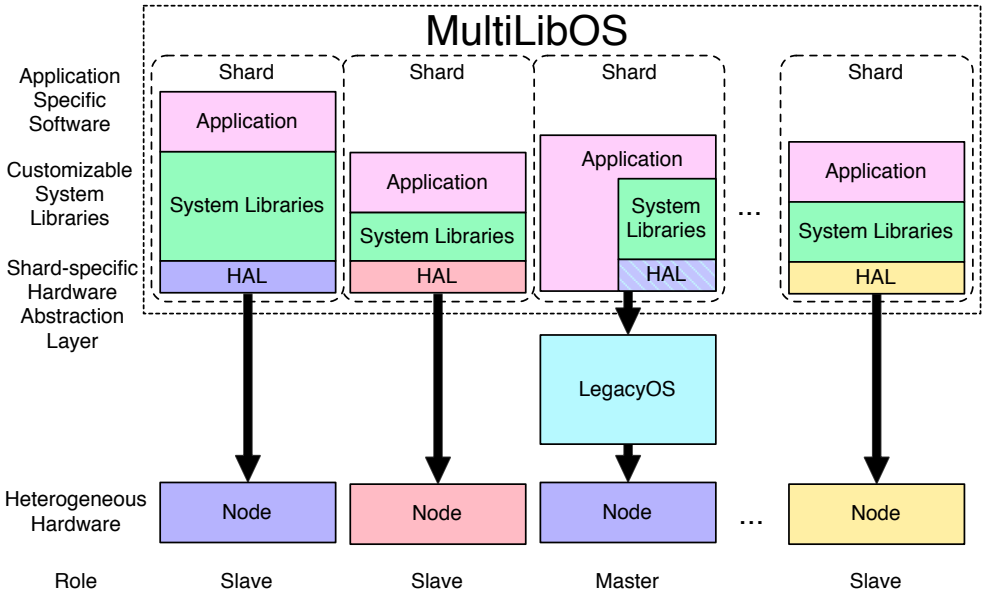

Figure 1: The MultiLibOS Architecture

operating system on a node is really just part of the application.

Just as with previous library OSes, the application can customize the OS functionality by selecting libraries, extending libraries, or parameterizing the OS libraries it is linked to. Also, just like with previous library OSes, the operating system functionality supports just a single tenant and a single application on a node.

In contrast with some previous library OSes [11], we don't depend on a specialized exokernel to allow multiple applications with their own library OSes to share a machine. We either: 1) assume that a node is dedicated to executing the shard and no sharing beyond low level partitioning is enabled, or 2) the shard is running as a process on an existing legacy OS.

Much like legacy OSes, the lowest layer of a shard is a hardware abstraction layer (HAL), which provides a consistent interface that most of the OS library functionality depends on. Also, just as with existing OS HALs, a specialized library can reach past the HAL interface to exploit unique features of a platform.

It is possible that over time the library OSes embedded in shards may grow to be very sophisticated. For example, Microsoft Research recently demonstrated a library OS that can support unmodified windows applications [19]. However, our expectation is that the library OSes will typically be very simple; more on the order of toy operating system functionality than fully functional OSes. There are a number of reasons why we believe that simplicity will be possible, three are discussed below.

Firstly, as demonstrated in our previous work [5] a very simple library OS is sufficient to support complex middleware like a JVM. We expect that we will find this to be the case for other managed code environments and for middleware services, like MPI libraries, that are designed to be highly portable as has been illustrated in the MPI Compute Node Kernel software environment[22] that IBM provides for Blue Gene.

Secondly, as we will discuss below, the MultiLibOS model augments rather than replaces existing legacy OSes. Hence, the application functionality that depends on the library OS is typically a small part of the whole application.

Thirdly, our experience is that it is much simpler to write efficient special purpose code to solve a specific problem than it is to write general purpose OS code [4]. The choice of libraries to execute a specific application computation on a specific hardware platform can all be made when a library OS is composed.

Process based shards

A HAL can be written to support a shard running as a process within a legacy operating system. The application code in a process based shard can directly interact both with the interfaces defined by the library OS, and with the interfaces exposed by the underlying OS. For example, it can access the native file system, use the native networking stack, and use the rich set of libraries and middleware provided by a fully functional OS.

While we say that the MultiLibOS model is single-tenant and single-application, with a process-based shard, the shard can directly interact with other applications (of other tenants) on the same node using the interfaces provided by the underlying operating system. This shard is also protected by the security and isolation provided by the legacy OS.

The use of process based shards provides us with a model that is very different from previous library OSes. Rather than providing an alternative OS architecture to implement the same functionality, we are developing a OS model that augments the existing legacy OSes. The application only uses the shard specific library OS for that functionality not done better by the underlying general purpose OS. In this fashion process shards provide a path for handling legacy function and a gateway to acceleration function of bare-metal shards.

Bare-metal shards

Some shards may run on bare-metal HALs, that runs directly on a physical (or virtual) node. A bare-metal shard is designed to be highly elastic. A shard may grow or shrink in memory or cores as demands

on the application changes.[1] More importantly, the bare-metal shards can be created and destroyed very quickly to increase or decrease the number of nodes being used by the application.

While the HAL in a bare-metal shard will allow much of the library OS code to be re-used across different types of shards, a library OS can be customized with libraries that exploit specific characteristics of the hardware. This means, for example, that a bare-metal shard can allow low level networking capabilities of the hardware to be exploited by an application. The barrier to introducing such functionality in a toy library OS is much smaller than modifying a complex fully functional multi-tenancy multi application OS.

Master slave relationship

While it is not fundamental to the MultiLibOS model, our current exploration assumes a master slave relationship. The slave shards (typically bare-metal) are composed by the master shard (typically process based), on the fly, to meet its computational requirements, or to match the characteristics of the available nodes.

The master/slave pattern we exploit has become common in many domains. For example, IBM's Cell Broadband Engine [1] has a general purpose core that acts a master to SIMD SPU slaves. Similarly GP-GPUs are used as accelerator's to x86 cores. In other contexts special purpose accelerators have been used for executing Java (e.g., from Azul Systems [15])). Much like how we don't have a general purpose OS on a Cell processor SPU, a GP-GPU, or a Java accelerator, there is no need to have a general purpose OS running on the accelerator nodes. All the complex control, coordination, forking, synchronization with other processes, authentication, process management happens on the master shard. We can exploit extremely simple OS functionality on the slave shards to scale out a computation to other nodes.

The simplicity of the functionality of slaves makes it possible for us to relatively quickly introduce new OS functionality that is relevant to real applications. It is much easier for us to introduce new, low level code specific to applications or node hardware.

[1]While today not all hypervisors support elastic characteristics, nor does all hardware support partitioning, we believe this functionality will be ubiquitous as elasticity becomes increasingly important.

5 Example Application

The following is a hypothetical scenario in which the computational capacity of a HPC system is exploited in a novel way using the MultiLibOS architecture.

Using the facilities of an IBM Blue Gene/P system[9] and the commodity apache[12] web-server the *neuro-cgi* MultiLibOS provides a building block for an online high-performance neural imaging service that allows for complex image analysis programs (that are today only used by researchers) to exist as part of a regular clinical work flow.

The *neuro-cgi* MultiLibOS is run across the nodes of a IBM Blue Gene/P system. The master shard is a process shards on a Linux (legacy) OS. In this way, the legacy OS acts as a platform for a front-end web server as well as a gateway to communicate onto the Blue Gene interconnects.

This master shard has been written to conform to apache's CGI module specification such that it can be invoked in response to a html request serviced by the apache server running on the front-end. Slave shards run as bare-metal on additional nodes.

On the arrival of a particular http request, the master-shard spools image data (submitted via http) to the associated storage system of the Blue Gene system[2]. Based on the size of the image data and requested operation the master shard launches an analysis work by allocating the required number of slave shards.

The slave shards are constructed elastically and consist of general MultiLibOS functionality libraries, a Blue Gene specific HAL, and custom Blue Gene

[2]Blue-Gene systems typically are configured with an external IBM GPFS storage system that provides high-bandwidth file I/O to the applications run on the Blue Gene system along with any other commodity hosts of the installation. In our example we assume our front-end is one such host with access to the GPFS storage and Blue Gene control system. In this way it can both store data and launch jobs on the Blue Gene system.

implementations of neuro-imaging functions that exploit the extensive Blue Gene matrix libraries and hardware functions. The slave shards spool results back to a file in the storage system of the master process shard. Using this data that master shard composes an html response that contains the results of the operation and feeds it back to the Apache process according to the CGI specification.

In this way, a html request sent from a client application can utilize the power of a supercomputer. In our example we can imagine that this application is run on a tablet computer operated by a doctor who has requested a test to be run on the scans of a patient. Once submitted to the *neuro-cgi* MultiLibOS, the job is scaled across many nodes of the Blue Gene/P system. Instead of having to wait days, the doctor could hypothetically receive the patient's test results within minutes.

6 Concluding remarks

Technology trends and the economics of providers acquiring datacenter-scale computers are changing the computing platform in fundamental ways. The assumptions that legacy operating systems where designed for are no longer valid. These operating system do not provide critical services that large scale distributed applications require. Although some of these services can be provided by middleware, we believe that this comes at the cost of performance as fully integrated datacenter scale systems become more common. Moreover, the legacy operating system architecture imposes high overhead and makes performance optimization difficult.

We have proposed a new architecture for operating systems that can be used to address the challenges of the cloud. A operating system constructed using this architecture would have library operating systems, i.e., shards, running on both legacy operating systems and bare hardware. The process shards would normally be used as master nodes. The bare-metal shards would be used as slaves to accelerate performance critical operations.

Other's have also argued that new operating systems are required [24, 6] for the cloud. These researchers are generally trying to solve a much tougher problem than we are addressing with the MultiLibOS architecture; they are generally trying to meeting the scalability, elasticity, availability... challenges of the cloud, while also having to reproduce the full functionality of legacy operating systems. With our architecture all the issues of multi-tenancy and resource management across applications are side stepped by exploiting the fine grain elastic abilities of the underlying platforms that OSes have not been exploiting. Also, we have separated the requirement of supporting legacy interfaces (i.e., process shards) from the need to address new hardware challenges including heterogeneity, elasticity and scalability (i.e., bare-metal shards). This allows for systems programmers to innovate in cloud environments without the burden of supporting the legacy functionality. The performance sensitive parts of the application can be incrementally ported to use the library code.

Innovation has been much more rapid in application level libraries than it has in lower level system software for a number of reasons. First, a library can be focused on the needs of a specific set of applications rather than having to be general purpose. Second, an application devleoper can often modify or extend a library. Third, an application programmer can choose libraries to meet their needs. Fourth, anyone can distribute a library which other's can then use, there is no need to dedicate hardware or other resources to that library except when the application is actually using it.

With the MultiLibOS architecture we believe that that operating systems will have as much room for innovation as application level libraries do today. In contrast to todays world, where there is a small number of operating systems, we believe that the MultiLibOS architecture, if adopted will result in many families of libraries, each addressing different concerns for different classes of applications and systems.

We are exploring one particular operating system, EbbOS, that adopts this architecture and targets a particular class of application and some of the challenges of cloud computing. While the architecture is important to the design of this OS, equally important are other ideas borrowed from the rich distributed OS research literature [10, 23, 18, 7]. In de-

veloping this OS we have discovered that the Multi-LibOS architecture simplifies our design and provides a model that usefully frames our discussion and reasoning. A number of the observations we have made in this paper where actually discoveries in pursuing this work rather than something we realized a-priori. We believe that other system programmers will find this architecture equally applicable. EbbOS has process shards GNU/Linux and MacOS, and bare-metal shards for AMD64 and PowerPC32, and is available from:

`https://github.com/SESA/EBBlib.`

References

[1] *The Design and Implementation of a First-Generation CELL Processor*, 2005.

[2] Vblock powered solution for vmware view. 2010.

[3] Hp project moonshot: Changing the game with extreme low-energy computing. 2012.

[4] J. Appavoo, K. Hui, C. A. N. Soules, R. W. Wisniewski, D. M. Da Silva, O. Krieger, M. A. Auslander, D. J. Edelsohn, B. Gamsa, G. R. Ganger, P. McKenney, M. Ostrowski, B. Rosenburg, M. Stumm, and J. Xenidis. Enabling autonomic behavior in systems software with hot swapping. *IBM Syst. J.*, 42(1):60–76, January 2003.

[5] J. Appavoo, K. Hui, C. A. N. Soules, R. W. Wisniewski, D. M. Da Silva, O. Krieger, M. A. Auslander, D. J. Edelsohn, B. Gamsa, G. R. Ganger, P. McKenney, M. Ostrowski, B. Rosenburg, M. Stumm, and J. Xenidis. Libra : A Library Operating System for a JVM in a Virtualized Execution Environment Libra Libra Hypervisor. *System*, 2007.

[6] M. Armbrust, A. Fox, R. Griffith, A. D. Joseph, R. H. Katz, A. Konwinski, G. Lee, D. A. Patterson, A. Rabkin, I. Stoica, and M. Zaharia. Above the clouds: A berkeley view of cloud computing. (UCB/EECS-2009-28), Feb 2009.

[7] Amnon Barak and Ami Litman. Mos: a multicomputer distributed operating system. *Softw. Pract. Exper.*, 15(8):725–737, August 1985.

[8] L.A. Barroso and U. Hlzle. *The Datacenter as a Computer: An Introduction to the Design of Warehouse-Scale Machines*. 2009.

[9] D. Chen, J. J. Parker, N. A. Eisley, P. Heidelberger, R. M. Senger, Y. Sugawara, S. Kumar, V. Salapura, D. L. Satterfield, and B. Steinmacher-Burow. The IBM Blue Gene/Q Interconnection Network and Message Unit. *Proceedings of 2011 International Conference for High Performance Computing, Networking, Storage and Analysis on - SC '11*, page 1, 2011.

[10] D. R. Cheriton. The v kernel: A software base for distributed systems. *IEEE Softw.*, 1(2):19–42, April 1984.

[11] D. R. Engler, M. F. Kaashoek, and J. O'Toole, Jr. Exokernel: an operating system architecture for application-level resource management. In *Proceedings of the fifteenth ACM symposium on Operating systems principles*, 1995.

[12] R. T. Fielding and G. E. Kaiser. The apache http server project. *IEEE Internet Computing*, pages 88–90, 1997.

[13] B. Gamsa, O. Krieger, J. Appavoo, and M. Stumm. Tornado: Maximizing Locality and Concurrency in a Shared Memory Multiprocessor Operating System. In *Proceedings of the third symposium on Operating systems design and implementation*, OSDI '99, pages 87–100, Berkeley, 1999. USENIX Association.

[14] J. Hamilton. Cost of power in large-scale data centers. 2008.

[15] Azul Systems Inc. Supercharging the java runtime. 2011.

[16] O. Krieger, M. Mergen, A. Waterland, V. Uhlig, M. Auslander, B. Rosenburg, R. W. Wisniewski, J. Xenidis, D. Da Silva, M. Ostrowski, J. Appavoo, M. Butrico, D. Da, S. Michal, and

O. Jonathan. K42: Building a Complete Operating System. *ACM SIGOPS Operating Systems Review*, 40(4):133, October 2006.

[17] V. Morozov. Blue gene/p architecture.

[18] J. K. Ousterhout, A. R. Cherenson, F. Douglis, M. N. Nelson, and B. B. Welch. The sprite network operating system. *Computer*, 21(2):23–36, February 1988.

[19] D. E. Porter, S. Boyd-Wickizer, J. Howell, R. Olinsky, and G. C. Hunt. Rethinking the library os from the top down. In *Proceedings of the sixteenth international conference on Architectural support for programming languages and operating systems*, ASPLOS '11, pages 291–304, New York, NY, USA, 2011. ACM.

[20] A. Rao. Seamicro technology overview. 2010.

[21] B. Rhoden, K. Klues, D. Zhu, and E. Brewer. Improving per-node efficiency in the datacenter with new os abstractions. In *Proceedings of the 2nd ACM Symposium on Cloud Computing*, SOCC '11, pages 25:1–25:8, New York, NY, USA, 2011. ACM.

[22] C. Sosa and B. Knudson. Blue gene/p application development. 2009.

[23] A. S. Tanenbaum and S. J. Mullender. An overview of the amoeba distributed operating system. *SIGOPS Oper. Syst. Rev.*, 15(3):51–64, July 1981.

[24] D. Wentzlaff, C. Gruenwald, N. Beckmann, K. Modzelewski, A. Belay, L. Youseff, J. Miller, and A. Agarwal. An operating system for multicore and clouds: Mechanisms and implementation. In *Proceedings of the 1st ACM symposium on Cloud computing*, SoCC '10, pages 3–14, New York, NY, USA, 2010. ACM.