2012-12-10

MultiLibOS: an OS architecture for cloud computing

MultiLibOS: An OS architecture for Cloud Computing

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

Abstract

Cloud computing allows consumers on-demand access to massive computational capacity. Researchers have argued that new operating systems are needed to address the challenges of scale, elasticity, and fault tolerance inherent to the cloud.

The cloud not only introduces new challenges, but also simplifies the role of the operating system. We describe how these simplifications enable a new model for developing and deploying OS functionality that we call *MultiLibOS*. This model can be used to enable many, potentially domain specific, runtimes and operating systems to address the challenges and broaden the applicability of cloud computing. The model allows OS, hardware and application researchers to innovate unconstrained by the requirements of backwards compatibility while still providing a strategy to enable full compatibility with commodity operating systems.

1 Introduction

Cloud computing has not resulted in fundamental changes to operating systems. Instead, distributed applications that could utilize its unique opportunity for scale and elasticity are built on top of middleware that stitches together multiple commodity operating systems to provide higher-level abstractions.

Others have argued that new operating systems built for the cloud can achieve much greater efficiency [29, 33]. However, what has been generally overlooked is that not only does the cloud introduce new challenges, but also it comes with major simplifications to the role of the operating system.

All the security concerns, and much of the complex resource management is provided by the cloud Infrastructure as a Service (IaaS) management and isolation layer rather than by the OS that runs within the nodes [1]. A cloud operating system does not necessarily need to support multiple applications or multiple users. In fact, since the application is distributed, full functionality does not need to be supported on every node.

Cloud applications are a heterogeneous composition of functions spread across libraries and distributed collections of computers of a datacenter scale system. We suggest that OSes should reflect this structure. Rather than running multiple instances of a single general purpose OS, we should compose applications using a collection of many special purpose OSes tuned to the needs of the application and nature of the hardware.

In this paper, we do not advocate for a single new operating system. Rather we propose a model for constructing application driven compositions of OS functionality. This will permit application requirements for scale, elasticity and fault-tolerance to be addressed in a more precise and effective manner.

We next discuss in more detail the simplified role an operating system has in the cloud. We present the proposed model in the following section. Finally we discuss the implications for meeting the challenges of cloud computing.

2 Reduced Role of the OS

For security and auditability, IaaS providers isolate their tenants at a very low level (either physically or virtually). Individual tenants own and manage logical computers, networks and disks within an IaaS cloud. Typically, for scale-out cloud applications, each application runs on a set of nodes dedicated to that application. In this environment three of the major objectives that existing commodity operating systems were designed to meet are either relaxed or eliminated.

Firstly, the burden to support multiple users is removed from the operating system. In this new environ-

[1] Throughout this paper we use the term *node* to refer to either a virtual machine or a physically isolated machine

ment, users are each given their own nodes each running their own applications and operating systems. This isolation 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.

Secondly, the IaaS management software assigns entire nodes to a single application. Hence, the functionality associated with balancing and arbitrating competitive use of resources across multiple applications is the responsibility of the provider. Much of the complexity of existing operating systems (e.g., scheduling, memory management, etc.) is redundant.

Thirdly, a symmetric structure is unnecessary in a large-scale distributed application. A distributed application that runs across many nodes does not require full OS functionality on all of its nodes. This applications can instead partition some nodes that only have specialized functionality while other nodes are designated to have full functionality.

We believe that these three simplifications: 1) not having to support multiple users on a node, 2) not having to support multiple applications on a node, and 3) not having to support full OS functionality on every node, simplifies the required system functionality on each node.

3 A Way Forward

Today, in the cloud, users deploy general purpose operating systems. Issues of scale, fault tolerance, and elasticity are addressed by middleware. If the cloud is the computer of the future, it is the responsibility of the OS community to provide the abstractions to allow that computer to be more easily and efficiently harnessed by distributed applications.

The natural inclination, and the approach taken by most distributed OS projects [9, 10, 17, 26, 29, 30, 33] is to build a single general purpose OS, where the same OS functionality is replicated on every node. This introduces two challenges. First, most efforts take a clean slate approach that often sacrifices the rich functionality and interfaces of existing general purpose OSes. Second, it is our experience, that issues of scale and fault tolerance (even on a relatively modest machine), are very difficult to address in a general way [16, 19]. The proliferation of domain specific middleware services and libraries for the cloud [11, 18, 22, 23, 24] that provide very different programming models supports this concern.

The observation of the reduced role of the operating system, discussed previously, suggests a way forward. Since we do not require the same functionality on all nodes, it becomes possible to augment a general purpose OS running on some nodes with new functionality running on other nodes. Since we don't have to support multiple users or multiple applications on a single node, new OS functionality can instead be provided by application specific library OSes linked into the application's own address space.

We therefore propose that operating system functionality be structured in a model we call *MultiLibOS*. A cloud application adopting this model is distributed across both nodes running full function OSes and bare hardware nodes. The full function OS nodes support complete OS functionality and legacy compatibility. The rest of the hardware executes a distributed library operating system where the system functionality is provided by libraries linked directly into the application address space that can be highly customized to the characteristics of the hardware or the application [2].

The intuition behind this model is that it is much simpler to solve many small problems rather than trying to provide general solutions. The complexity of traditional operating systems is driven by sharing of system resources across multiple applications and users and the fact that one body of operating system code is used for all purposes. With the MultiLibOS model, we exploit the on-demand nature of resource allocation and deallocation in the cloud to allow applications to acquire dedicated resources for a particular sub-function, such as processing a stream of images for a particular feature, and then releasing the resources. The hardware acquired for this purpose, as well as the libraries used by the application on that hardware, can hence be focused on aspects unique to that single application sub-function running in isolation. Issues of protection, fairness and general multiplexing are eliminated. Rather, application-centric aspects of system software can take a front seat: application specific APIs, light-weight hardware abstraction, distributed primitives, etc.

Examples

We believe that the MultiLibOS model has broad applicability, not only offering value for standard scale-out cloud applications, but also potentially enabling applications that are poorly suited to today's cloud. To concretize the discussion, we describe two example applications that follow in the MultiLibOS model. The first being a standard web application that receives incremental benefits from the MultiLibOS model. The second describes a new application not feasible with the standard model of deploying OS functionality in the cloud.

First, consider a typical web application comprised of four standard components: front-end Apache [13] servers, Java business logic, memcached [14] instances,

[2] Unlike previous library OS researchers [12], the model does not require a specialized exokernel to allow multiple applications to share a machine. Instead, the provisioning and isolation provided by the IaaS provider is sufficient for the large scale applications targeting cloud computing.

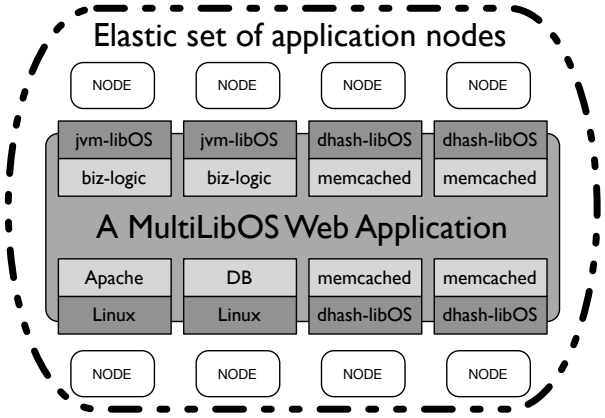

Figure 1: How a typical web app might be restructured as a MultLibOS Application.

and database servers. A typical deployment may distribute these components across many nodes each running on top of the same operating system. With the MultiLibOS model, new OS functionality can be introduced to incrementally optimize this application.

As illustrated in Figure 1, in a MultiLibOS model, Apache and the database can run on standard Linux instances with full network protocol compatibility. Meanwhile, the Java business logic and memcached servers can run on their own custom library operating systems. The nodes dedicated to each subfunction can be added and removed rapidly depending on the demand on the application. Previous work has demonstrated that substantial advantages in Java performance results from specialized library OSes [2, 31]. The memcached library OS, while maintaining its API, could be implemented using a hardware tuned distributed hash table that in a fine-grain fashion exploits the features of the underlying platform, e.g., [3].

Next, consider an online neural imaging service that provides complex image analysis. This application has two components: a web front-end that provides an interface to the application, and a computationally intensive imaging library. These components can be separated by providing the web accessibility on Apache Linux nodes while the core computation is run on nodes loaded with a custom library OS. The library OS can exploit the characteristics of the hardware and application to optimize for scale and elasticity in isolation from the protocol compatibility provided by Linux nodes. The simplicity of the back-end library OS allows new nodes to be spun up for seconds at a time, enabling the extreme elasticity necessary to allow massive computational resources to be exploited for short bursts of time. This *interactive supercomputing* can allow for computationally intensive workloads, currently restricted to scheduled batch computation, to run as part of a daily work flow.

4 Implications

In the previous section we described the MultiLibOS model, and suggested some of the advantages in the context of two simple examples. In this section we describe key implications of the model.

Full functionality

The integration of commodity OSes into our model implies that optimizing existing applications and introducing new system support for scale, elasticity and fault-tolerance can be done incrementally. Additionally, the development of new applications can greatly benefit from the ability to exploit the interoperability and tools provided by the commodity OSes.

With the MultiLibOS model, the baremetal libraries do not need to replace the full OS functionality, instead they augment it. We have the same advantage that we had in past OS research projects which reproduced the functionality of general purpose OSes [19]. Applications written against legacy interfaces can incrementally exploit the new features of the OS only where relevant. However, we get that advantage without the huge investment done to reproduce that legacy compatibility. In our previous research, we found that 90% of our time was spent on the last 5% of compatibility with commodity operating systems.

The use of commodity operating systems in the MultiLibOS model is not just important for existing applications. When writing entirely new applications, the commodity operating systems still provide many benefits. These applications can integrate with existing system tools and primitives such as signals, pipes, scripts, etc. Furthermore, applications can address the needs for protocol and API compatibility by integrating with the large body of software written for commodity operating systems.

Simplicity

While library OSes can be very sophisticated [28], in the MultiLibOS model we expect them to be very simple [3] for two reasons. First, they don't require complex scheduling, resource management or security. Second, it is much easier to produce special purpose code to support a specific application than to write general purpose OS code.

Libra [2] demonstrated that a simple library OS is sufficient to support complex middleware like a JVM. We

[3] Simplicity in this context is not just limited to development cost. Since a library OS is deployed and configured for a single application both the testing and the distribution are hugely simplified. The testing only needs to be concerned with the functionality of the one application, and the library can be distributed with the application.

expect this to be the case for other managed code environments and for middleware services, like MPI libraries which can be designed to be highly efficient [25].

Simplicity has further implications to elasticity and specialization, as discussed later. In addition, simplicity is critical to allow radical system techniques to be introduced and explored. OS research projects have explored new ideas in the context of clean-slate approaches because only in a simple project is it practical to explore new ideas. Plumbing innovation through a complex general purpose OS is an enormous challenge. In many cases, the innovation, if successful, requires integration into a general purpose system for widespread adoption. A key characteristic of our model is that we expect that simple library OSes can have an instant impact and be directly applicable to real applications.

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. One key way to achieve efficiency is elasticity; having the resources used by the application vary depending on what the applications demands are.

The MultiLibOS model encourages the design of lightweight systems. Only the software necessary to support the application is deployed on the baremetal nodes. A small system image allows for fast deployment on the cloud. Additionally, due to limited functionality (e.g., no need for device discovery), the system is able to boot quickly.

The design of a general purpose operating system is often times at odds with the goals of an elastic application. A general purpose operating system is designed to boot once, initialize, and run for a long period of time. In contrast, an elastic system may lazily initialize components. The resource usage model is also different. A general purpose OS will discover all devices and use what resources are provided to it. An elastic system can exploit the ability of a hypervisor to add or remove cores or memory as necessary. This system can more accurately reflect the applications demand for resource consumption.

In the above discussion, we have discussed elasticity from the perspective of deploying, booting, and managing the resources of an individual node. More speculatively, we believe that the MultiLibOS model has additional implications for elasticity across nodes. Since there is no separation between the application and the OS, full information is available on the load and demands on different components to determine when nodes should be added or removed. Also, since the model encourages decomposing an application into subfunctions that are deployed on different nodes, it will be easier to understand the impact of changing the resources of a specific subfunction.

Application Specialization

The MultiLibOS model allows libraries to be written that are specialized to a subset of applications. Applications that do not benefit from specific OS functionality need not use the library that provides it. In contrast, to get a low-level optimization into a general purpose operating system it has to be justified for general use.

Previous work has shown that applications benefit from low-level optimizations. Applications and managed code environments having control over page tables have achieved greater efficiency [6, 31]. Specialized support for message passing, locks and event driven systems have gradually been incorporated into various operating systems [4, 5, 15]. In a network centric system, low level control over the networking hardware can have a dramatic effect on multi-core performance [27].

Providing specialized functionality is, in our opinion, critical to meet the challenges of the cloud. Our experience in building high-performance system software for large-scale shared memory multiprocessors [16, 19] 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. We believe that the same holds true for fault tolerance and elasticity.

In addition to being necessary to meet the cloud challenges, our experience with a customizable operating system is that it reduces code complexity. It is much simpler to write efficient special purpose code to solve a specific problem than it is to write general purpose OS code [1]. 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.

Hardware Specialization

The MultiLibOS model is a natural fit for the heterogeneity of the cloud. Just as the model allows for customization to the needs of an application, library OSes can be optimized to the characteristics of specific hardware.

IaaS datacenters are intrinsically heterogeneous. Non-uniform latencies and bandwidth exist between different parts of the datacenter. Large datacenters may have many generations of hardware, each with different quantities and characteristics of processing, memory, and networking. Different systems may have different prop-

erties, e.g., networking properties like scatter gather and RDMA, or different compute accelerators like GPGPUs. HP's Moonshot [21] embraces heterogeneity in the cloud by constructing a system out of server cartridges which have a wide variety of configurations for different applications.

We hypothesize, that the MultiLibOS will enable even greater heterogeneity in the cloud. In the MultiLibOS model, full functionality is not provided on all nodes and, therefore, there may be opportunities for hardware developers to provide radically different hardware [32] which would not support general purpose software.

Integrated Systems

The MultiLibOS model implies a path for exploiting the natural trend towards higher degrees of integration in cloud infrastructure.

We hypothesize that datacenters will increasingly be constructed using highly-integrated large scale systems. Rather than building large systems out of commodity network, compute and storage, the parts of an integrated system are designed for large scale operation. These systems are optimized for aggregate price, performance and manageability as opposed to per-node performance. This shift towards integrated datacenter scale systems can be observed in systems such as HP's Moonshot [21], IBM's BlueGene [8], and SeaMicro [20].

Highly integrated systems will require the kinds of specialization described above. As we have seen in high performance computing, much better efficiency is achieved using specialized kernels that exploit all the features of the hardware while stripping out the majority of general OS functionality (e.g., multi-user, multi-program support and general purpose paging [7].)

Low-level optimizations become more important with highly-integrated systems. With clusters of commodity hardware, the network latency is typically high enough that the performance of general purpose system software is adequate. However, in highly-integrated systems, where the relative latency of the network is typically much lower, the system software will have greater impact on application performance.

In this section we highlighted several implications to the MultiLibOS model that suggest its utility as a framework for future cloud OS research. While we focused on the advantages of the model, we acknowledge that there are significant challenges in adopting this model. Challenges range from very pragmatic tooling and configuration, to more conceptual questions regarding the granularity of decomposition appropriate for different systems.

5 Concluding Remarks

The cloud is changing the computing platform in fundamental ways. The assumptions that our current operating systems were designed for are no longer valid. These operating systems do not provide the 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, especially as highly integrated datacenter scale systems become more common.

We have proposed a model for introducing new operating system functionality into the cloud while preserving full OS functionality. In the MultiLibOS model, an application is distributed across nodes running full function operating systems and nodes with library operating systems. Subfunctions of an application can be partitioned onto different nodes each with their own library OS. The operating system functionality of each library OS can be customized to the needs of the application and the characteristics of the hardware.

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

We believe that this model can enable additional innovation by both application developers and hardware architects. Hardware that achieves major gains for even a subfunction of an application can be usefully deployed. Gains in the hardware will drive improved system functionality that, in turn, will allow the massive computational power of today's clouds to be accessible to a broader range of applications.

We are developing an instance of a MultiLibOS, called EbbRT [4] which explores some of the implications discussed above as well as the challenges of tooling, configuration and decomposition. We have found the ability to take an application focused approach powerful, where we can build functionality only as needed. By exploiting commodity operating systems in the model, we have been able to focus our efforts on some of the more radical research ideas, with greatly reduced effort compared to previous systems we worked on.

[4] http://www.github.com/sesa/ebblib

References

[1] 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.

[2] 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.

[3] Jonathan Appavoo, Amos Waterland, Dilma Da Silva, Volkmar Uhlig, Bryan Rosenburg, Eric Van Hensbergen, Jan Stoess, Robert Wisniewski, and Udo Steinberg. Providing a cloud network infrastructure on a supercomputer. In *Proceedings of the 19th ACM International Symposium on High Performance Distributed Computing*, HPDC '10, pages 385–394, New York, NY, USA, 2010. ACM.

[4] G. Banga, J.C. Mogul, P. Druschel, et al. A scalable and explicit event delivery mechanism for unix. In *USENIX Annual Technical Conference*, pages 253–265, 1999.

[5] A. Baumann, P. Barham, P. Dagand, T. Harris, R. Isaacs, S. Peter, T. Roscoe, A. Schüpbach, and A. Singhania. *The Multikernel*. SOSP '09. ACM Press, New York, New York, USA, 2009.

[6] Adam Belay, Andrea Bittau, Ali Mashtizadeh, David Terei, David Mazières, and Christos Kozyrakis. Dune: safe user-level access to privileged cpu features. In *Proceedings of the 10th USENIX conference on Operating Systems Design and Implementation*, OSDI'12, pages 335–348, Berkeley, CA, USA, 2012. USENIX Association.

[7] R. Brightwell, A. B. MacCabe, and R. Riesen. On the appropriateness of commodity operating systems for large-scale, balanced computing systems. In *Proceedings of the 17th International Symposium on Parallel and Distributed Processing*, IPDPS '03, pages 68.1–, Washington, DC, USA, 2003. IEEE Computer Society.

[8] 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.

[9] D. R. Cheriton. The V Kernel: A Software Base for Distributed Systems. *IEEE Softw.*, 1(2):19–42, April 1984.

[10] D.R. Cheriton, M.A. Malcolm, L.S. Melen, and G.R. Sager. Thoth, a portable real-time operating system. *Communications of the ACM*, 22(2):105–115, 1979.

[11] Jeffrey Dean and Sanjay Ghemawat. Mapreduce: simplified data processing on large clusters. In *Proceedings of the 6th conference on Symposium on Opearting Systems Design & Implementation - Volume 6*, OSDI'04, pages 10–10, Berkeley, CA, USA, 2004. USENIX Association.

[12] 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.

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

[14] Brad Fitzpatrick. Distributed caching with memcached. *Linux J.*, 2004(124):5–, August 2004.

[15] H. Franke, R. Russell, and M. Kirkwood. Fuss, futexes and furwocks: Fast userlevel locking in linux. In *AUUG Conference Proceedings*, page 85. AUUG, Inc., 2002.

[16] 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.

[17] Graham Hamilton and Panos Kougiouris. The spring nucleus: A microkernel for objects. Technical report, Mountain View, CA, USA, 1993.

[18] Michael Isard, Mihai Budiu, Yuan Yu, Andrew Birrell, and Dennis Fetterly. Dryad: distributed data-parallel programs from sequential building blocks. In *Proceedings of the 2nd ACM SIGOPS/EuroSys European Conference on Computer Systems 2007*,

EuroSys '07, pages 59–72, New York, NY, USA, 2007. ACM.

[19] Orran Krieger, Marc Auslander, Bryan Rosenburg, Robert W. Wisniewski, Jimi Xenidis, Dilma Da Silva, Michal Ostrowski, Jonathan Appavoo, Maria Butrico, Mark Mergen, Amos Waterland, and Volkmar Uhlig. K42: building a complete operating system. In *Proceedings of the 1st ACM SIGOPS/EuroSys European Conference on Computer Systems 2006*, EuroSys '06, pages 133–145, New York, NY, USA, 2006. ACM.

[20] G. Lauterbach. Recovery act: Seamicro volume server power reduction research development. Technical report, SeaMicro, 2012.

[21] Kevin Lim, P. Ranganathan, Jichuan Chang, C. Patel, T. Mudge, and S.K. Reinhardt. Server designs for warehouse-computing environments. *Micro, IEEE*, 29(1):41 –49, jan.-feb. 2009.

[22] Yucheng Low, Danny Bickson, Joseph Gonzalez, Carlos Guestrin, Aapo Kyrola, and Joseph M. Hellerstein. Distributed graphlab: a framework for machine learning and data mining in the cloud. *Proc. VLDB Endow.*, 5(8):716–727, April 2012.

[23] Grzegorz Malewicz, Matthew H. Austern, Aart J.C Bik, James C. Dehnert, Ilan Horn, Naty Leiser, and Grzegorz Czajkowski. Pregel: a system for large-scale graph processing. In *Proceedings of the 2010 ACM SIGMOD International Conference on Management of data*, SIGMOD '10, pages 135–146, New York, NY, USA, 2010. ACM.

[24] Sergey Melnik, Andrey Gubarev, Jing Jing Long, Geoffrey Romer, Shiva Shivakumar, Matt Tolton, and Theo Vassilakis. Dremel: interactive analysis of web-scale datasets. *Commun. ACM*, 54(6):114–123, June 2011.

[25] José Moreira, Michael Brutman, José Castaños, Thomas Engelsiepen, Mark Giampapa, Tom Gooding, Roger Haskin, Todd Inglett, Derek Lieber, Pat McCarthy, Mike Mundy, Jeff Parker, and Brian Wallenfelt. Designing a highly-scalable operating system: the blue gene/l story. In *Proceedings of the 2006 ACM/IEEE conference on Supercomputing*, SC '06, New York, NY, USA, 2006. ACM.

[26] 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.

[27] Aleksey Pesterev, Jacob Strauss, Nickolai Zeldovich, and Robert T. Morris. Improving network connection locality on multicore systems. In *Proceedings of the 7th ACM european conference on Computer Systems*, EuroSys '12, pages 337–350, New York, NY, USA, 2012. ACM.

[28] 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.

[29] 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.

[30] 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.

[31] Gil Tene, Balaji Iyengar, and Michael Wolf. C4: the continuously concurrent compacting collector. In *Proceedings of the international symposium on Memory management*, ISMM '11, pages 79–88, New York, NY, USA, 2011. ACM.

[32] Rob F. van der Wijngaart, Timothy G. Mattson, and Werner Haas. Light-weight communications on intel's single-chip cloud computer processor. *SIGOPS Oper. Syst. Rev.*, 45(1):73–83, February 2011.

[33] 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.