A Way Forward: Enabling Operating System Innovation in the Cloud

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

1 Introduction

Cloud computing has not resulted in a fundamental change to the underlying operating systems. Rather, distributed applications are built over middleware that provides high-level abstractions to exploit the cloud's scale and elasticity. This middleware conjoins many general purpose OS instances.

Others have demonstrated that a new operating system built specifically for the cloud can achieve increased efficiency, scale and functionality [11, 14]. However, this work does not take into account the way applications are being deployed in cloud environments. In particular, entire physical or virtual machines are being dedicated to run a single application, rather than concurrently supporting many users and multiple applications.

In this paper we introduce a new model for distributed applications that embraces a reduced role of the OS in the cloud. It allows for the construction of application-driven compositions of OS functionality wherein each application can employ its own customized operating system.

2 Role of the OS

For security and auditability, Infrastructure as a Service (IaaS) providers isolate their tenants at a very low level as physical or virtual compute *nodes*. Individual tenants own and manage their compute nodes, software stack, networks and disks within an IaaS cloud.

Typically, scale-out cloud applications run across a set of compute nodes solely dedicated to that application. In such an environment, three of the major objectives that general purpose operating systems were designed to meet are relaxed or eliminated entirely.

First, the burden to support multiple users is removed from the operating system. In this environment, the isolation enforced by the IaaS provider eliminates the need for many system level security checks and accounting, and reduces the requirement for internal barriers between trusted and untrusted code.

Second, it becomes the responsibility of the IaaS provider to arbitrate and balance competitive resource usage. In a deployment where entire nodes are assigned to a single application, much of the complexity of existing operating systems (e.g., scheduling, memory management, etc.) is redundant.

Third, a symmetric structure is unnecessary in a large-scale distributed application. Many cloud applications are already composed of multiple services run across a set of compute nodes; As a result, OS functionality can be provided asymmetrically, where only some nodes need full OS functionality, while other nodes can be much simpler.

Given these observations, it is apparent that distributed cloud applications built on top of general purpose systems are comprised of unnecessary software functionality with the risk of reduced performance and added complexity.

3 A Way Forward

The reduced role of the operating system in the cloud suggests a new way forward for providing OS functionality and further optimising application performance. Since we do not require the same OS functionality on all nodes, it becomes possible to combine general purpose operating systems with specialized operating systems. Since we do not have to support multiple users or multiple applications on a single node, new OS functionality can be provided by application-specific library OSs [3] linked directly into the application's address space.

We propose that operating system functionality be structured in a model we call *MultiLibOS*. A cloud application adopting this model is distributed across a mix of general purpose OSs and specialized library OSs. The general purpose OS nodes support complete OS functionality and legacy compatibility, while the rest of the nodes execute simple, customized, library operating systems.

With the MultiLibOS model, we exploit the on-demand nature of resource management in the cloud to allow applications to allocate dedicated nodes for a particular task. The hardware acquired for this purpose, as well as the libraries used by the application on that hardware, can be focused on aspects unique to that application's task. 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.

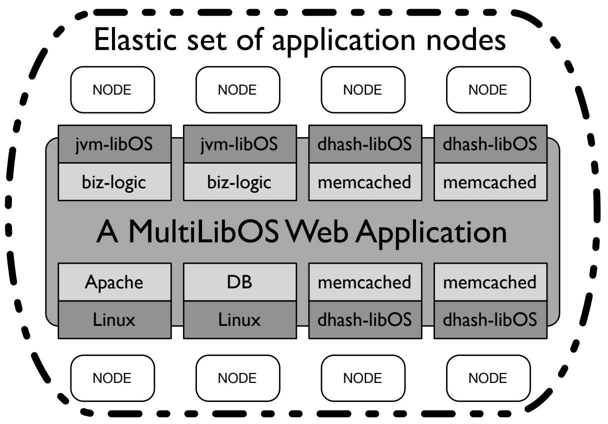

Figure 1: A web app structured as a MultLibOS Application.

For example, consider a typical web application comprised of four standard components: front-end Apache [4] servers, Java business logic, memcached [5] instances, and a database server. A typical deployment might distribute these components across multiple nodes that each run the same underlying operating system.

As illustrated in Figure 1, a MultiLibOS model can run Apache and the database on nodes running Linux. Meanwhile, the Java business logic and memcached servers can run on their own highly specialized library operating systems. The nodes dedicated to these tasks can be rapidly added and removed in response to application demand.

One might structure the memcached nodes to run on a specialized library OS designed for distributed hash tables (dhash-libOS in the figure) that directly identifies and processes cache requests at interrupt level. Such a system would have no need for virtual memory, scheduling, or rich interfaces. By avoiding expensive kernel/user level context switches, it is feasible that a full memcached request can be processed in a few hundred CPU cycles. In contrast, memcached running on top of Linux is likely to require tens of thousands of cycles to service a request.

As our past work, along with others, have demonstrated, specialized library OSs can result in substantial advantages for Java applications [1, 12] by removing redundant OS functionality (e.g., scheduling), and allowing the JVM direct control over system memory and page tables. A MultiLibOS, as illustrated, would allow such a JVM library OS (jvm-libOS in the figure) to be naturally integrated and used in a complex heterogeneous web application.

4 Implications

The MultiLibOS model introduces an intuitive way to asymmetrically distribute OS functionality across an application while preserving legacy compatability. In this section we describe the key implications of such a model as they apply to modern cloud applications.

Simplicity

Intuitively, it is simpler to provide an application with a special purpose feature, such as a high speed messaging service built directly on top of a hardware supported RDMA system, rather than implementing a fully functional general feature, such as sockets and the associated software to support TCP/IP and arbitrary networking hardware. Following this intuition, we expect that the library OSs in the MultiLibOS model will be lightweight and simple. In particular, it may be that a library OS would not need to provide complex protection logic.

In addition, simplicity is critical to allow experimental system techniques to be introduced and explored. With other OS models, new features and interfaces need to be integrated into a general purpose system for widespread adoption. Plumbing a new innovative feature through a complex general purpose OS is an enormous challenge. A key characteristic of our model is that new features can have an instant impact and be directly applicable to real world applications.

Application Specialization

The MultiLibOS model allows libraries to be written that are specialized for a set of applications. As noted, applications that do not benefit from specific OS functionality need not include the library that provides it.

Previous work has shown that applications benefit from low-level optimizations. For example, applications and managed code environments that have control over page tables have achieved greater efficiency [2, 12]. Specialized support for message passing, locks and event driven systems have gradually been incorporated into various operating systems. In a network centric system, low level control over the networking hardware can have a dramatic effect on multi-core performance [10].

Providing specialized functionality is critical to meet the challenges of scale, elasticity and fault tolerance. Our experience in building high-performance system software for large-scale shared memory multiprocessors [6, 7] is that there is no one answer for accelerating parallel applications: to achieve high performance, the operating system needs to be customized to meet the specific needs of that application.

With the MultiLibOS model, we believe 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 OSs, each addressing different concerns for different classes of applications.

Hardware Specialization

Just as the MultiLibOS model allows for customization to the needs of an application, library OSs 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 properties, e.g., networking properties like scatter gather and RDMA, or different compute accelerators like GP-GPUs. Illustrating this trend is HP's Moonshot [8] which embraces heterogeneity in the cloud infrastructure by constructing a system out of server cartridges, each with 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, custom OS functionality can enable hardware developers to provide radically different hardware [13] that could not easily support general purpose software due to the lack of hardware features such as virtual memory or privileged domains.

Hardware that achieves major gains for even a small set of applications can be usefully deployed. Gains in the hardware will drive improved system functionality that, in turn, will allow the computational power of today's clouds to be accessible to a broader range of 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 through elasticity, that is, having the resources used by the application vary depending on what the application's demands are.

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. A library OS can be customized to include support for only the specific devices needed by the application, reducing boot time and, thus, permitting fast component instantiation and execution.

Full functionality

The integration of general purpose OSs into our model implies that optimizing existing applications and introducing new system-level support for scale, elasticity and fault-tolerance can be done incrementally.

With the MultiLibOS model, the system libraries do not need to replace the full OS functionality, instead, they augment it. Applications written against legacy interfaces can incrementally exploit the new features of the library only when valuable. We found the ability to incrementally exploit new OS functionality critical in previous operating system projects [7]. In the MultiLibOS model, this advantage is achieved without the huge investment we had previously made to reproduce legacy compatibility. For example, in Libra [1], we implemented a Library OS for a JVM where most POSIX system calls were function-shipped to a colocated VM running a general purpose OS.

In this model, we have the same advantage that we had in past OS research projects which reproduced the functionality of general purpose OSes [7]. 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 general purpose operating systems in the MultiLibOS model is not just important for existing applications. Entirely new applications written to the MultiLibOS model may benefit from the ability to exploit the interoperability and tools provided by the general purpose OSs. These applications can integrate with existing system tools and primitives, and address the needs for protocol and API compatibility by integrating with the large body of software written for general purpose operating systems.

5 Research Questions

In the previous section we highlighted several implications to the MultiLibOS model as a framework for future cloud applications. While we focused on the advantages of the model, we acknowledge that there are also significant research challenges.

The MultiLibOS model advocates for a proliferation of OS libraries. As with application libraries, this introduces known issues of configuration, compatibility, administration, and coping with "versionitis." While there exist methods and tooling to cope with these problems at user level, it remains to be seen how these apply to the construction of operating system functionality. An additional disadvantage is the inherent fragmentation of the OS development community. Accordingly, it seems less likely that each individual library OS will be as tested, secure and reliable as our current operating systems. Language level techniques may help address these problems [9].

To avoid the development of custom OS libraries for every application, there must exist libraries that can be reused. This will require the definition of interfaces that can remain compatible with a wide range of other libraries. We foresee it being a significant challenge to prevent a collection of compatible libraries from itself becoming a new operating system, and losing the advantages of the MultiLibOS model.

The practical applicability of this model is dependent on whether or not the value gained by building specialized OSs is worth the added cost of development and administration. In the past, the construction of specialized operating systems has largely been inhibited by issues of hardware and software compatibility. However, the discussed implications of both our model and the deployment of modern cloud applications suggests that it is time for this trade-off to be reevaluated.

6 Concluding Remarks

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.

We have proposed a model for introducing new operating system functionality into the cloud while preserving legacy compatibility. In the MultiLibOS model, an application is distributed across nodes running general purpose operating systems and nodes with library operating systems. Particular tasks of an application can be partitioned onto different nodes each with an accompanying library OS. The operating system functionality of each library OS can be customized to the needs of the application and the characteristics of the underlying hardware.

We are developing an instance of a MultiLibOS, called EbbRT [1] 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 general purpose operating systems in the model, we have been able to focus our efforts on more radical research ideas, with greatly reduced effort compared to previous systems we have worked on.

Throughout the development of EbbRT we have come to realize that there exists a vast space of MultiLibOS implementations and designs. It has become clear to us that the research questions generated by the MultiLibOS model exceed the scope of any one research project. We hope that many applications will be developed to explore the implications and challenges of the MultiLibOS model.

[1]http://www.github.com/sesa/ebbrt

Acknowledgements

This material is based upon work supported by the National Science Foundation under Grant Numbers; CNS-1347525, CNS-1012798, and CNS-1254029.

References

[1] G. Ammons, J. Appavoo, M. Butrico, D. Da Silva, D. Grove, K. Kawachiya, O. Krieger, B. Rosenburg, E. Van Hensbergen, and R. W. Wisniewski. Libra: A library operating system for a jvm in a virtualized execution environment. In *Proceedings of the 3rd International Conference on Virtual Execution Environments*, VEE '07, pages 44–54, New York, NY, USA, 2007. ACM.

[2] A. Belay, A. Bittau, A. Mashtizadeh, D. Terei, D. Mazières, and C. 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.

[3] 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.

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

[5] B. Fitzpatrick. Distributed caching with memcached. *Linux J.*, 2004(124):5–, Aug. 2004.

[6] 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.

[7] O. Krieger, M. Auslander, B. Rosenburg,

R. W. Wisniewski, J. Xenidis, D. Da Silva, M. Ostrowski, J. Appavoo, M. Butrico, M. Mergen, A. Waterland, and V. 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.

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

[9] A. Madhavapeddy, R. Mortier, C. Rotsos, D. Scott, B. Singh, T. Gazagnaire, S. Smith, S. Hand, and J. Crowcroft. Unikernels: library operating systems for the cloud. *SIGPLAN Not.*, 48(4):461–472, Mar. 2013.

[10] A. Pesterev, J. Strauss, N. Zeldovich, and R. 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.

[11] 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.

[12] G. Tene, B. Iyengar, and M. 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.

[13] R. F. van der Wijngaart, T. G. Mattson, and W. Haas. Light-weight communications on intel's single-chip cloud computer processor. *SIGOPS Oper. Syst. Rev.*, 45(1):73–83, Feb. 2011.

[14] 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.