International Journal of Advanced Research Methodology in Engineering & Technology,
Volume 1, Issue 1, December 2016

The Role of MultiLibOs in Cloud Computing

Archana Karnik K.M[1], Jayapadmini Kanchan [2]

[1]Assistant Professor,Dept. of Computer Science,AIMS&R

[2]Assistant Professor, Dept. of ISE, Sahyadri College of Engineering and Management,Mangalore

[1]archanayenni@yahoo.co.in, [2]jayapadmini@gmail.com

ABSTRACT

Cloud computing means the use of computing resources that are delivered in the form of service over the internet. Cloud computing is resulting in fundamental changes to computing infrastructure. These changes in the computing infrastructure have not resulted in corresponding changes to operating systems. A new operating system called MultiLibOs is built specifically for the cloud to achieve increased efficiency, scale and functionality. Later in this paper MulitLibOS architecture is explained. If this architecture is adopted will result in many families of libraries, each addressing different concerns for different classes of applications and systems.

Key words: Cloud Computing, MultiLibOs, Baremetal.

1. INTRODUCTION

Cloud computing refers to the delivery of computing resources over the Internet. It is a practical approach to experience direct cost benefits and it has the potential to transform a data centre from a capital-intensive set up to a variable priced environment. The idea of cloud computing is based on a very fundamental principal of reusability of IT capabilities.

In an era when the focus is on cloud computing [1] why is the operating system more important than ever? The answer is that in order for end users to benefit from a cloud platform that supports balanced workloads that can scale in a secure manner; the operating system has to be designed to enhance that cloud platform.

Cloud operating system plays an important role in Internet Browsing and storing data. Cloud and many core systems share several challenges with respect to the operating system. Cloud is a new thought in the area of network computing. Traditional operating system cannot fulfil all the requirements of cloud computing.
Some key factors influence the computing infrastructure and applications of IaaS systems to change [2].In this paper the need for the MidLib operating system is explained along with the architecture of the proposed operating system.

2. LITERATURE SURVEY

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 at-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.

L.A. Barroso and U. Hlzle[1] propose that the trend toward server-side computing and the exploding popularity of Internet services has created a new class of computing systems that we have named warehouse-scale computers, or WSCs. Moreover, datacenter economics allow many application services to run at a low cost per user. The computation itself may become cheaper in a shared service. Finally, servers and storage in a datacenter can be easier to manage than the desktop or laptop equivalent because they are under control of a single, knowledgeable entity.

Judith Hurwitz & Marcia Kaufman [2] propose that the cloud environment has to protect the identity of users and information from external threats. Even more importantly, security has to be managed across a diverse set of resources, each with its own way of managing security. To support the needs of most organizations, cloud security requires an integrated management

capability within the operating system that can track all IT assets in the context of how they are being used.

Dan Schatzberg, James Cadden, Orran Krieger, Jonathan Appavoo [3] defines a model for introducing new operating system functionality into the cloud while preserving legacy compatibility.

D. R. Engler, M. F. Kaashoek, and J. O'Toole [4] have proposed Exokernel design guidelines for exokernels. The main task of the exokernel is to securely expose machine resources to applications. The exokernel employs access control and secure bindings to achieve this goal safely. To allow effective application-level resource management, the exokernel uses physical names and visible resource revocation. An abort protocol is used to protect against uncooperative applications.

D. Wentzla_, C. Gruenwald, N. Beckmann, K. Modzelewski, A. Belay, L. Youse_, J. Miller, and A. Agarwal [5] explains that how inter-process communication and virtual memory can be implemented efficiently and directly at application level. Frequently, the performance differential between ExOS and Ultrix is more than an order of magnitude.

J. Hamilton [7] has proposed, a key opportunity for reducing the cost of cloud service data centers is to eliminate expensive infrastructure. Turning geo-diversity into geo-redundancy requires that the critical state for data center applications be distributed across sites, and frameworks to support this remain a non-trivial systems and networking research problem.

3. IMPORTANCE OF OS IN CLOUD

One of the most significant requirements for companies adopting cloud computing is the need to adopt a hybrid approach to computing [6]. To do so, most organizations will continue to maintain their traditional data center to support complex mixed workloads. In order for end users to benefit from a cloud platform that supports balanced workloads that can scale in a secure manner, the operating system has to be designed to enhance that cloud platform.

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.

3.1 KEY FACTORS

In this section important factors that are needed in IaaS infrastructure and applications are discussed.

Datacenter-Cost Reduction

Data center costs are concentrated in servers, infrastructure, power requirements, and networking, in that order. Though costs are steep, utilization can be remarkably low. Several approaches are adopted to significantly improve data center efficiency [7]. First, we need to increase internal data center network agility, to fight resource fragmentation and to get more work out of fewer servers – reducing costs across the board. Second, we need to pursue the design of algorithms and market mechanisms for resource consumption shaping that improve data center efficiency. Finally, geodiversifying data centers can improve end to end performance and increase reliability in the event of site failures. To reap economic benefits from geo-diversity, it is needed to design and manage data center and network resources as a joint optimization, new systems are needed to manage the geo-distribution of state.

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 a hypervisor allows the application to add and remove cores and memory as the demands on the application changes [4]. Direct interconnect access allows programmers to explore the trade o_ 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.

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 [7]. This increased focus on efficiency is driving IaaS providers, hardware architects, system 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'[9] systems where the aggregate power consumption is more directly and smoothly proportional to the use of the system.

Scalable Applications

The 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 today's system software is adequate. However, in highly-integrated systems, where the relative latency of the network is typically much lower, the system software has much more of an impact on application performance. The general availability of highly elastic IaaS services [3] and the characteristics of highly integrated systems may well enable new classes of more demanding scale-out applications.

4 THE MultiLibOS

The MultiLibOS model introduces an intuitive way to asymmetrically distribute OS functionality across an application while preserving legacy compatibility [3]. A MultiLibOS is a single tenant; single application distributed operating system composed of per-node library operating systems. The component of the application/OS that runs on a node is called as 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.

4.1 The Library Os Architecture of MultiLibOs

In this section we discuss Architecture of MultiLib in figure 1.

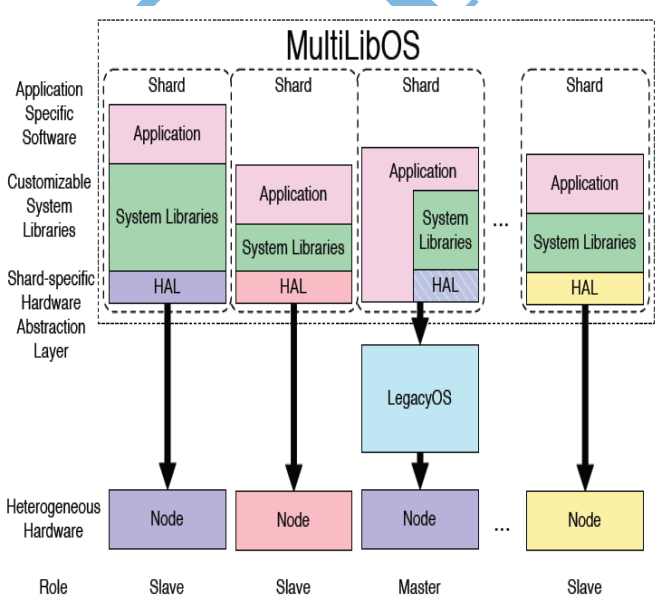

Figure 1: Architecture of MultiLibos

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 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 [2] 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 [4], specialized exokernel do 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.

The use of process based shards provides us with a model that is very di_erent 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 speci_c library OS for that functionality not done better by the underlying general purpose OS.

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 system[8], use the native networking stack, and use the rich set of libraries and middleware provided by a fully functional 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. 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, which runs directly on a physical (or virtual) node. A baremetal 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 baremetal shard can allow low level networking capabilities of the hardware to be exploited by an 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 baremetal 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 baremetal) are composed by the master shard (typically process based), on the y, 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. 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.

5. CONCLUSION

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.
With the MultiLibOS architecture 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 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.
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. This 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.

REFERENCES

[1] The Design and Implementation of a First-Generation CELL Processor, 2005.
[2] The Role of the Operating System in Cloud Environments Judith Hurwitz, President Marcia Kaufman, COO.
[3] A Way Forward: Enabling Operating System Innovation in the Cloud, Dan Schatzberg, James Cadden, Orran Krieger, Jonathan Appavoo Boston University.

[4] 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 _fteenth ACM symposium on Operating systems principles, 1995.

[5] D. Wentzla_, C. Gruenwald, N. Beckmann, K. Modzelewski, A. Belay, L. Youse_, 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.

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

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

[8] D. R. Cheriton. The v kernel: A software base for distributed systems. IEEE Softw., 1(2):1942, April 1984.

[9] J. Hamilton. Cost of power in large-scale data centers. 2008.
[10] A. S. Tanenbaum and S. J. Mullender. An overview of the amoeba distributed operating system. SIGOPS Oper. Syst. 1981.