Performance Comparison of Virtual Machines and Containers with Unikernels

Nagashree N, Suprabha S, Rajat Bansal
VMware Software India Private Limited, Bengaluru.
{nagashreen, suprabhas, rajatb}@vmware.com

In a typical cloud environment that uses software-defined data center (SDDC), most of the virtual machines are deployed to run single application for back-end services resulting in the under-utilization of virtual resources. With virtual machines, legacy applications are run on a full-fledged OS. This overhead is avoided by containers using OS-level virtualization where the applications are ‘caged’ to provide secure isolation. However, this adds another layer to the software stack. Unikernels, leveraging the benefits of virtualization, remove these redundant layers by enabling us to build applications with only the required libraries. These applications run as virtual machines on a hypervisor in Ring 0 (kernel) mode. Consequently, the attack surface is minimal ensuring very high security as compared to containers. Also, the privileges are reduced building potentially stronger walls between disparate components. The applications can scale largely because of a very small memory footprint.

In this paper, we use OSv – a unikernel, Cloud OS, to build and deploy applications such as Apache, MySQL, etc. We have done the performance analysis of these applications running in virtual machine, container and unikernel. We have observed that the unikernel VM size is the smallest, the boot time is much faster and the read/write time is significantly low.

1. INTRODUCTION

Cloud computing arose from the realization that virtualization facilitates efficient use of hardware resources by providing a greater degree of abstraction between hardware and software. A virtual machine is a self-contained computer with a standard operating system running applications. It helps with server consolidation as there is lesser hardware to maintain. It is scalable and secure. However, each VM runs a full copy of an operating system and a virtual copy of all the hardware that the OS requires. Containers provide an alternative to this by virtualizing the operating system which relatively reduces the resource consumption. A container wraps up applications and all their dependencies in a complete file system. Containers, running as isolated processes in the host operating system’s user space, share the kernel that makes RAM usage and disk usage more efficient as compared to VMs [7]. Despite these benefits, containers are not as secure as virtual machines. The attack surface is comparatively larger because if there is vulnerability in the OS’s kernel, it can make its way through the containers. Also, they lack to offer the same level of isolation as hypervisors. The next logical approach from VMs and containerization is the “Unikernels”.

Unikernels are specialized, single address space machine images constructed by using ‘library operating systems’. A developer selects the minimal set of libraries which correspond to the OS constructs required for their application to run. These libraries are then compiled with the application and configuration code to build sealed, fixed-purpose images (unikernels) which run directly on a hypervisor or hardware without an intervening OS. [8] (Fig. 1)
Virtualization, in spite of many of its advantages, adds many layers and this overhead consumes more resources. Unikernels significantly reduce this overhead without compromising on any of the advantages of virtualization. Some of the key advantages are:

2. Security — The application is the kernel. Unikernels provide extremely tiny and specialized runtime footprint that is less vulnerable to attack.
3. Scalability — The image size is very small (less than 100MB), enabling flexible deployment on a large scale.

As with many technologies, unikernels also face a chasm between establishing the technical value and market adoption. With this paper, we attempt to bridge this gap and help unikernels be adopted widely across various industries.

In Section 3 and Section 4, we present evidences for the above by comparing virtual machines and containers (Docker) with unikernels.

2. RELATED WORK

The architecture of library operating system (libOS) has several advantages over more conventional designs [1]. “Unikernels: Rise of the Virtual Library Operating System”, details the limitations of current operating systems and how library operating systems overpower them. It illustrates the design of MirageOS which aims to restructure VMs into more modular components for increased benefits.

There are several unikernels or library operating systems such as OSv, ClickOS, Clive, HaLVM, LING, MirageOS and Rump Kernels. In “OSv — Optimizing the Operating System for Virtual Machines”, Avi Kivity et al. explain that OSv is a better OS for cloud than traditional operating systems such as Linux.

J(VM)2 [2], a pure Java-based VM encapsulated inside a specialized hypervisor-aware framework, provides many substantial advantages over the traditional JVM. unikernels.

The copyright of this paper is owned by the author(s). The author(s) hereby grants Computer Measurement Group Inc. a royalty free right to publish this paper in CMG India Annual Conference Proceedings.
3. PROOF-OF-CONCEPT

Unikernels, by design, will not have open-vm-tools running inside them, as only one process runs within each unikernel. However, each of the unikernel implementations has an instrumentation API to get similar data.

OSv is the unikernel OS used for building and deploying applications. OSv [4] is the open source operating system designed for the cloud. Built from the ground up for effortless deployment and management, with superior performance.

3.1 Legacy and Cloud-Native Applications

There are not many applications that are ready to be run as unikernel images. Building a unikernel image for an application requires detailed knowledge of the application’s runtime and its dependencies, so that only the relevant parts of the OS library can be compiled into the image. A few legacy applications like Java, MySQL and Cloud-Native applications like Apache, Cassandra are compiled to an OVA. Later, they are deployed as virtual machines on a hypervisor. The virtual machine boots the application automatically in an instant as soon as it powers on. However, the VM shuts down if the application crashes or fails to start. The automated process of building an OSv application is depicted in Fig. 2.

Fig. 2: OSv Application Build Process

3.2 Performance

Considering the above applications, a performance evaluation is done for unikernels against virtual machines and containers (Docker). A standard benchmark tool is chosen for each of the applications which are configured consistently across VM, container & unikernel. The stats are provided in the next section.

3.3 Security

The two major flaws in containers are isolation and security. Since, unikernels are deployed as virtual machines on a hypervisor, these are not a concern. Containers do not provide resilient, multi-tenant isolation. A malicious code inside the container can attack the host operating system and other containers. Docker daemon requires root privileges to run containers (applications) which means that the underlying OS is vulnerable. The packaging of applications is still tricky because of the uncertainty in their dependencies [3]. Unikernels are more secure because the OS is only the libraries relevant for the application. There is no support for device drivers which are a potential risk. Unikernels offer the highest level of isolation as there is no privilege switching between user & kernel modes.

BTC Piñata, a MirageOS unikernel, is an evidence for the level of security that a unikernel can provide. It is written in OCaml, runs directly on Xen, and uses native OCaml TLS and X.509 implementations.

3.4 Which Applications to build as Unikernels?

Every application will not be suitable for implementation as a unikernel. Like other technologies, unikernels also have
their limits. The key limitations are [14]:

1. Lack of multiple processes (Single Process, Multiple Threads)
2. Single User
3. Limited Debugging
4. Impoverished Library Ecosystem

An application that doesn’t violate these limits can be considered for compiling as a unikernel. The application shouldn’t fork new processes in the same machine. Whenever it demands for a new process, a new unikernel VM can be created as it starts up in sub seconds. Debugging can be improved by instrumenting it internally. Unikernel specifically targets security and scalability (applications that may need to scale into very high numbers). Unikernel would be appropriate for something that will be exposed to the Internet and therefore needs the highest levels of security [14]. There is no specific set of criteria for whether an application can be built as a unikernel or not. This is due to the fact that the technology is new. However, with time they can be widely adopted and many applications can be built as unikernels.

5. RESULTS & DISCUSSION

Unikernels run as conventional virtual machines on a hypervisor. We measure the performance of applications such as Apache HTTP server & MySQL running in virtual machine, container (Docker) and unikernel.

We have performed all the tests on a hypervisor with twelve 1-2 GHz Intel Xeon E5-2620 processors for a total of 12 cores, Hyper-Threading and 64 GB RAM. This is the configuration of the mainstream server to deploy virtual machines, containers and unikernels. We have used Ubuntu 15.10 (Wily Werewolf) 64-bit with Linux kernel 2.6.27.56 as base image for all Docker (v1.8.0) containers and cloud image for all virtual machines. We run a single instance of each of Apache Web Server & MySQL applications in VM, container and unikernel.

a. Apache Web Server

The performance of Apache Hypertext Transfer Protocol (HTTP) server is measured with a benchmarking tool called ApacheBench (ab). This tool especially shows how many requests per second the Apache installation is capable of serving.

We have configured Apache HTTP server and ApacheBench on a single virtual machine with 4 vCPUs and 2 GB RAM. We run the Apache Benchmark Docker image in a single container hosted on another VM. We launch the server in unikernel and run the benchmarking tool in a VM. For consistency, all of these connect to "https://www.yahoo.com".

The total number of bytes received from the server which is essentially the number of bytes sent over the wire is 524210. The total number of document bytes received from the server excluding bytes received in HTTP headers is 460120. The number of requests is 1000. The number of concurrent clients used during the test is 100.

The performance of the server running in each of VM, container and unikernel is measured across three parameters as shown in TABLE I. Unikernel performs better than VM & container as the time taken (mean) to serve a request across all concurrent requests is the least. However, the rate at which requests are returned per second is better than VM and close to container.

<table>
<thead>
<tr>
<th>Performance Metric</th>
<th>VM</th>
<th>Container</th>
<th>Unikernel</th>
</tr>
</thead>
<tbody>
<tr>
<td>Requests/Second (Higher is better)</td>
<td>16.45</td>
<td>30.12</td>
<td>28.12</td>
</tr>
<tr>
<td>Time/Request (ms)</td>
<td>5480.987</td>
<td>3200.13</td>
<td>3087.29</td>
</tr>
<tr>
<td>Total Test Time (s)</td>
<td>54.23</td>
<td>31.230</td>
<td>28.53</td>
</tr>
</tbody>
</table>

TABLE I: Performance Analysis of Apache HTTP server with VM, Container & Unikernel
Fig. 3 shows the max. time taken to connect to the server, process the request and send the response to the clients. With VM, the fastest request was 55 ms and the slowest was 10900 ms. In case of container and unikernel, the fastest was 71 ms and 58 ms respectively. The slowest was almost half the time taken as compared to VM. Fig. 4 shows the percentage of the requests served within a certain time. The server running inside the VM takes double the time to serve last 50% of the requests. Unikernel performs fairly well for 1000 requests compared to others. In real-time where the server has to satisfy thousands of requests, VMs and containers take a very long time. However, unikernels offer predictive performance.

![Fig. 3: Maximum Time Taken to Connect, Process Request & Wait for response](image1)

![Fig. 4: Time Taken to Service the Requests](image2)
b. MySQL

The MySQL performance is measured with sysbench [13], a benchmark suite which allows to quickly get an impression about system performance while running a database under intensive load.

We have configured MySQL and sysbench on a single VM with 4 vCPUs and 2 GB RAM. We have used Percona v5.6 Server which is a fork of the MySQL relational database management system created by Percona. We follow the same configuration in container. We run only MySQL in unikernel and run sysbench inside another VM. This simulates a more "real world" use case. We run sysbench OLTP for Mixed OLTP (R/W) and Read-Only OLTP tests with single thread and table size equal to 20,000,000.

Fig. 5: Read-Only and R/W Requests per Second

Fig. 6: Time Taken to service 95% of Read-Only and R/W Requests

Fig. 5 shows the Read-Only and R/W requests per second. The number of Read-Only requests per second is higher in
case of unikernel and lower in case of VM & container. However, with R/W requests per second, VM beats the others. Fig. 6 shows the time taken to serve 95% of Read-Only and R/W requests. Comparatively, unikernel takes nearly half the time to satisfy 95% of the Read-Only requests. However, this is not the case with the latter as the time taken is relatively higher. This is due to the network latency caused by connecting from a different host. Regardless, unikernel did not kill performance all that much & is more suitable for read-intensive applications.

c. Performance Assessment

The performance analysis is done for database and web server via applications such as MySQL and Apache Web Server. These applications running in each of VM, container and unikernel are measured across standard parameters as shown in TABLE II. With Apache Web Server, unikernel performs better than VM & container as the average time taken to serve a request across all concurrent requests is the least. However, the rate at which requests are returned per second is better than VM and close to container. With MySQL, the read-only (R) requests are served faster by unikernel. Also, the rate at which read-only requests are returned per second is better than VM & container. The read/write (RW) requests per second and the time taken to serve them are almost the same across all the three technologies.

<table>
<thead>
<tr>
<th>Scenario</th>
<th>Application</th>
<th>Requests/Second (Higher is better)</th>
<th>95% Service Time (ms)</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td>VM</td>
<td>Container</td>
</tr>
<tr>
<td>Database</td>
<td>MySQL (sysbench)</td>
<td>7398.52 (R), 5588.97 (RW)</td>
<td>7301.21 (R), 4771.01 (RW)</td>
</tr>
<tr>
<td>Web Server</td>
<td>Apache (ab)</td>
<td>16.45</td>
<td>30.12</td>
</tr>
</tbody>
</table>

TABLE II: Performance Analysis of Applications with VM, Container & Unikernel

Unikernels might not be the panacea, however, it performs much better than containers and virtual machines in some cases, and on par with them in the other cases.

6. CONCLUSION

In our work, we have compared unikernels with virtual machines and containers to show that they perform better for certain applications. Unikernels allow applications that demand predictable performance to access the hardware resources directly. Specialized implementations can be provided by including only the required set of libraries, drivers and interfaces. In addition to performance, security, scalability, isolation and feasibility, it allows to provide Disaster Recovery (DR) solution as a unikernel service. As unikernels boot in an instant, a new unikernel can be summoned whenever an application crashes. It also provides Transient Micro services in the Cloud. Nowadays, the hardware is fast, compact and cheap, allowing us to come out of the assumption that VM is persistent. A single VM need not be multifunction. A new unikernel VM can be summoned to handle each request over the network. Unikernels, with all these advantages, can serve as a better operating system for SDDC.

7. ACRONYMS & ABBREVIATIONS

<table>
<thead>
<tr>
<th>Acronym/Abbreviation</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>ab</td>
<td>ApacheBench</td>
</tr>
<tr>
<td>BTC</td>
<td>Bitcoin</td>
</tr>
<tr>
<td>DR</td>
<td>Disaster Recovery</td>
</tr>
<tr>
<td>HaLVM</td>
<td>Haskell Lightweight Virtual Machine</td>
</tr>
<tr>
<td>libOS</td>
<td>library Operating System</td>
</tr>
<tr>
<td>OLTP</td>
<td>OnLine Transaction Processing</td>
</tr>
<tr>
<td>OVA</td>
<td>Open Virtual Appliance or Application</td>
</tr>
<tr>
<td>SDDC</td>
<td>Software-Defined Data Center</td>
</tr>
<tr>
<td>VM</td>
<td>Virtual Machine</td>
</tr>
<tr>
<td>VMDK</td>
<td>Virtual Machine Disk</td>
</tr>
<tr>
<td>VMX</td>
<td>Virtual Machine Configuration File</td>
</tr>
</tbody>
</table>

The copyright of this paper is owned by the author(s). The author(s) hereby grants Computer Measurement Group Inc. a royalty free right to publish this paper in CMG India Annual Conference Proceedings.
ACKNOWLEDGEMENT

We would like to extend our gratitude to VMware for allowing us to carry out the research work by providing complete support and adequate resources.

REFERENCES