# Web Service Appliance Based on Unikernel

Kai Yu, Chengfei Zhang, Yunxiang Zhao
Science and Technology on Parallel and Distributed Laboratory, College of Computer
National University of Defense Technology, Changsha, Hunan
Email: yukaigoforit@126.com, chengfeizh@gmail.com, zhaoyx1993@163.com

***Abstract*—Mini-OS is a tiny OS (operating system) kernel distributed with Xen Project Hypervisor. It is mainly used as an OS for stub domain aimed at Dom0 disaggregation and also a stepping stone for Unikernel development. We implemented a simple http server on Mini-OS, and built Mini-OS into a web service appliance. We evaluated its performance compared with the same implemented server on Ubuntu PV (para-virtualization) DomU, and achieved about 39% performance improvement. The results show that Mini-OS can be a web service appliance and has a good performance**

***Index Terms*—Unikernel, Mini-OS, Web service appliance**

## I. INTRODUCTION

As one of the key technologies on building cloud computing infrastructure, virtualization has got great attention from researchers and developed a lot over the past few decades.[7] With virtualization, multiple virtual machines can be built on only one physical machine, and each virtual machine can be a standalone terminal in the cloud distributed system. Compared with using physical platforms directly, using virtualization has a huge advantage in many aspects, such as high-efficiency, high-reliability, and dynamic scheduling of computing resources.

Unikernel is a relative newcomer among virtualization technologies, raised by Madhavapeddy et al of Cambridge University. Observing that standalone application usually relies on only a small number of services provided by operating system, unikernel strips away most unnecessary components of operating system and becomes a single address space system. Unikernel packs an application with its needed system components into a single-purpose, sealed and highly-specialized image, which can then be run on the hypervisor or on hardware [1].

Xen is a well-known virtual machine monitor (hypervisor) and can be a platform most unikernels run on. It comes with Mini-OS, a tiny OS kernel mainly used as an OS for stub domains aimed at Dom0 disaggregation. Mini-OS is also a stepping stone for unikernel development. Several examples, including Rump kernels and ClickOS, have been formed on the base of Mini-OS [2]. Inspired by this, we try to build Mini-OS into a unikernel by implementing several applications on Mini-OS, and we want to know whether the applications on Mini-OS get better performance. So firstly we implemented a reduced http server on Mini-OS and built Mini-OS into a web service appliance. Compared with the implemented server on Ubuntu PV, the web service appliance has about 39% performance improvement, which means that Mini-OS can be a web service appliance (unikernel) and unikernel indeed is more efficient than general-purpose OS. The main contributions of this paper are summarized as follows:

1. We give Mini-OS a simple http server and run it on Xen, make Mini-OS a web service appliance (unikernel).
2. We make an empirical study on http servers' performance of web service appliance and linux. Our results suggest that our web service appliance has an improved performance, which can be an evidence about unikernel's high-effiency.

## II. BACKGROUND

General-purpose OSs, like some flavor of Windows, Linux, were designed to be run on hardware, they have all the complex functionality needed for an assortment of hardware drivers [4]. Since it tends to configure virtual machine to run a single application, such as a DNS server or a database in the cloud today. Using these fully functionalized OSs as virtual system will result in inefficiencies and resources consuming.

After eliminating the unnecessary parts between application and hardware, and treating the final VM image as a single-purpose appliance, unikernel is a lightweight, efficient and quick booting image. Unikernel also is highly-specialized by simplifying and optimizing the kernel into what we want, offers improved security because it cannot be modified after being deployed on a cloud platform [1]. As shown in Fig.1, unikernel has reduced many layers of soft-ware stack compared with general-purpose OS.

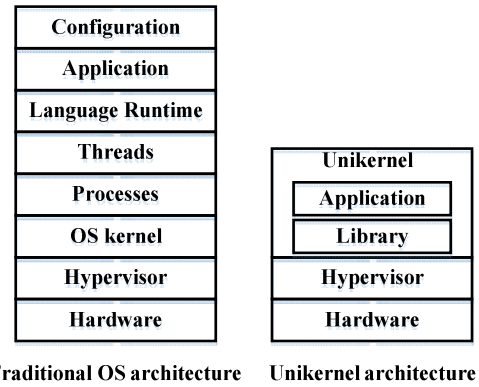



Fig.1 Traditional OS architecture VS Unikernel architecture

Unikernels, such as MirageOS [1], OSv [5], have been developed these years. They were designed for different purpose

2332-5666/17 $31.00 © 2017 IEEE
DOI 10.1109/ICDCSW.2017.38

and were implemented in a variety of high-level programming languages. MirageOS and OSv both have implemented many popular applications on them, including http server. Experiments show that their web server appliance exceeds apache2 on linux[1,5].

Because of unikernel's these advantages on image size, efficiency and security versus general-purpose OSs, unikernel is more and more popular in cloud computing. For example, some developers put their websites on unikernels for saving source of remote servers [8].

## III. HTTP SERVER ON MINI-OS

Xen is a popular hypervisor for most unikernels. Xen's many characteristics make it can be regarded as a whole operating system, and the unikernels on Xen become processes in OS. Mini-OS implements all the basic function needed to run on Xen, it has a tiny size with no more than one megabyte. Like unikernel Mini-OS is a single-address space image and developers can write applications on it. Mini-OS is equipped with some standard libraries, lwIP as TCP/IP stack, newlib as C standard library, zlib as compression library, and pciutils for PCI devices management. Since there is a ready-made tiny OS kernel on Xen, we use Mini-OS for unikernel development. We wrote an http server on Mini-OS and built it into a web service appliance.

We used C language to write our http server because it was used in Mini-OS as well. Our purpose in this paper is not to implement an excellent http server which can be used in production environment, so we implement a reduced http server for Mini-OS which is so plain that it just served a single static page. As most experimental http servers do, our server listens to request from clients all the time with a listening socket, when a request comes in, a new client socket is created to handle the request and send the page. In the beginning, we implemented a single-process-single-thread http server with non-blocking IO which handles only one request every time. This server had poor performance on Mini-OS. Mini-OS is single-address space OS with only one process, while it can implement multi-threads. So we add the server with thread using the multi-thread interfaces provided by Mini-OS, but it didn't work well either. Finally, we equipped the http server with IO multiplexing mechanism.

IO multiplexing also is referred as event-driven model. Mini-OS provided *select* as high-performance event interface. IO multiplexing makes one single process can handle many networking IO simultaneously. After user process calls *select* function, OS kernel monitors all the sockets in the charge of *select*, including listening socket and all client sockets, when any socket is ready to output data, *select* returns and notifies process to accept connection or read data. With the help of IO multiplexing, our http server worked well on Mini-OS.

## IV. EVALUATION

### A. Experiment Configuration

Fig.2 showed our experiment configuration. We used Xen 4.4 as our hypervisor, and the Mini-OS kernel distributed with Xen 4.4 for our tests. We installed Xen from source on an Ubuntu 12.10 physical machine, and the Ubuntu OS became Xen Dom0. We also built Ubuntu 14.04 PV DomUs on Xen, and the implemented http server run on Ubuntu PV as a comparison.

We used weighttp 0.4 as our test client, which was a light and small benchmarking tool for webservers [6]. Weighttp could provide the number of how many requests the server responses per second (reqs/s), we took it as the performance evaluation of http server. Weighttp client run on another Ubuntu 12.10 physical machine, which was connected with Xen Dom0 via our lab's private network. Xen Dom0 and weighttp client machine were both assigned static IP addresses, so did all DomUs.

All Xen DomUs were allocated one vcpu. In the original configuration file, Mini-OS were allocated 64M (megabytes) of memory. To find whether memory size has influence on server performance, we increased the memory size of Mini-OS from 64M to 1024M, doubled each step. However, the *reqs/s* number remained relatively consistent, which means that 64M is enough for Mini-OS. In the final test, we allocated 512M of memory to Mini-OS. We did the same memory test on Ubuntu PV as well, and allocated 512M of memory to it.

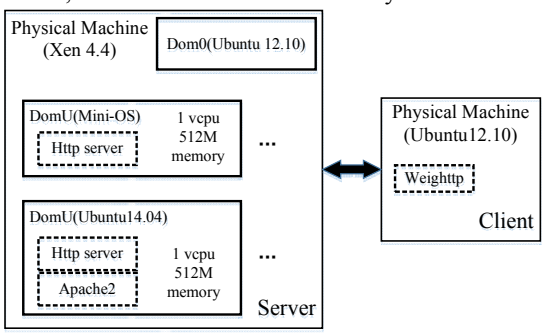



Fig.2 Experiment configuration

As Mentioned above, we implemented three kinds of http servers on Mini-OS, and the event-driven http server had the best performance. We run all the servers on Ubuntu PV too, but the response rate (*reqs/s* number) of event-driven http server was much smaller than single-process-single-thread http server. We speculated it is because Ubuntu as general-purpose OS has more software stack layers. Therefore, we chose the single-process-single-thread http server for Ubuntu PV. In evaluation part, we also tested Apache HTTP Server 2.4 [3] on Ubuntu PV for comparison.

### B. Experiment Result

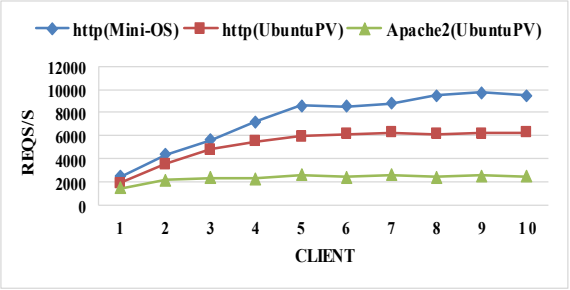
*1) Multi-client concurrence*

Fig.3 (a) compares the performance of http servers when client number increases. The results show that the response rate of all servers tends to be stable when concurrent clients increase. Obviously, http server performance on Mini-OS is better than that on Ubuntu PV, about 39% performance improvement. Since the http server we implemented leaves out many complicated processes. It exceeds Apache2 which can be used in production environment.

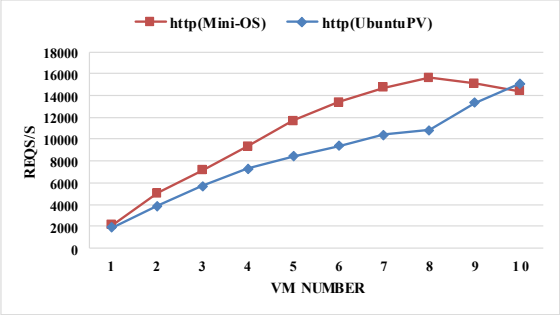
*2) Multi-VMs scaling out*

We scaled out the VM number, by running the http server on multiple DomUs and testing them simultaneously, then we

summed their *reqs/s* numbers. As shown is Fig.3 (b), the total response rate of both http servers increases as the VM number grows. While when the VM number gets larger, the increase speed gets slower.




(a) Multiple clients




(b) Multiple VMs

Fig.3 Experiment results

## V.CONCLUTION

We implemented and tested a reduced http server on Mini-OS, and also tested it on Ubuntu PV for comparison. The results showed that http server performance on Mini-OS is better than that on Ubuntu PV. It is clear that Mini-OS can be a web service appliance and has a good performance. Meanwhile, since unikernel is a single-purpose appliance with a tiny image size and quick booting, we can run many unikernels simultaneously to improve performance. In contrast, Ubuntu is a traditional and general-purpose OS, scaling out its VM number to run a single application will result in resources consuming.

## VI.FUTURE WORK

The http server we implemented can only be used for testing, far from being in practical use. In future we will optimize the http server so that it can be used in production environment. Meanwhile, we will implement other applications on Mini-OS, such as DNS server and database[9]. Next step we are going to modify the implementation of redis to run it on Mini-OS, and make Mini-OS a redis appliance.

## REFERENCES


1. Madhavapeddy A, Mortier R, Rotsos C, et al. Unikernels: library operating systems for the cloud[C]// International Conference on Architectural Support for Programming Languages and Operating Systems. 2013:461-472.
2. Xen Project community. Mini-OS wiki [EB/OL]. https://wiki.xenproject.org/wiki/Mini-OS.
3. Apache HTTP Server Team. Apache HTTP Server Project [EB/OL]. https://httpd.apache.org/.
4. Madhavapeddy A, Scott D J. Unikernels: the rise of the virtual library operating system[J]. Communications of the Acm, 2014, 57(1):61-69.
5. Kivity A, Laor D, Costa G, et al. OSv: optimizing the operating system for virtual machines[J]. 2014.
6. Lighty Labs. Weight project of lighttpd [EB/OL]. http://redmine.lighttpd.net/projects/weighttp.
7. Gong C, Liu J, Zhang Q, et al. The characteristics of cloud computing[C]//Parallel Processing Workshops (ICPPW), 2010 39th International Conference on. IEEE, 2010: 275-279
8. Shi J, Yang Y, He J, et al. Design of a comprehensive virtual machine monitoring system[C]//Cloud Computing and Intelligence Systems (CCIS), 2014 IEEE 3rd International Conference on. IEEE, 2014: 510-513.
9. Zhang Y, Li D, Sun Z, Zhao F, Su J, Lu X. CSR: Classified Source Routing in DHT-Based Networks. IEEE Transactions on Cloud Computing. 2015 Jun 2. doi: 10.1109/TCC.2015.2440242.