# Bare-Metal Container

--- Direct execution of a container image on a remote machine with an optimized kernel ---

Kuniyasu Suzaki, Hidetaka Koie, Ryousei Takano
National Institute of Advanced Industrial Science and Technology
{ k.suzaki | koie-hidetaka | takano-ryousei }@aist.go.jp

***Abstract*— Container technology has become popular in HPC applications because it offers easy customization and quick execution. However, container technology does not allow kernel-level optimization. It does not allow changing a kernel as well as loading/unloading a kernel module, thus it prevents the extraction of full performance on a target machine. In addition, container technology is designed for multi-tenancy and makes difficult to measure exact performance and power consumption for an application on a target machine.**

**Bare-Metal Container (BMC) solves these problems by a mechanism that offers a suitable kernel for a container image and boots it on a remote physical machine. The application can occupy the physical machine and utilize the full performance. It is implemented by remote machine management technologies (WakeupOnLAN, Intel AMT, and IPMI), and a network bootloader. The power consumption is also measured, and the user knows the good combination among application, kernel and target machine. Although BMC requires the overhead of kernel booting, HPC applications can improve performance, which surpasses the overhead. The experiments showed the effect of kernel optimizations for CPU (Hyper Threading), memory (Transparent Huge Pages), and network (Receive Flow Steering) from a low power CPU (Celeron) to a high speed CPU (Xeon).**

***Keywords*— container, kernel optimization, power measurement, application-centric architecture**

## I. INTRODUCTION

As the popularity of Docker [10] shows, container technology [33] is becoming widely used because it enables easy customization and can be shared on hub servers. It also is popular for HPC applications because it offers quick execution[14,21]. However, the container technology represented by Docker has limitations for HPC because it does not allow the Linux kernel to be customized to extract the highest level of performance. For example, Docker does not permit loading/unloading of the kernel module. This issue directly impacts applications, with the most famous case being the Data Plane Development Kit (DPDK) [11]. DPDK is a user-space tool for fast packet processing that bypasses the Linux networking stack but requires "igb_uio" and "rte_kni" kernel modules on the Linux host. This results in a Docker image for DPDK that does not run on all Docker environments. Furthermore, kernel options that are passed through /sys or /proc are not effective because Docker uses union file system (AUFS or DeviceMapper), preventing these options from reaching the kernel. The reason is that container technology is based on system-centric architecture which is led by providers and does not allow changes to the kernel and its settings.

Even if a user wants to run a Docker image on a real or virtual machine, it is not so easy because normal Docker image does not have a Linux kernel and a bootloader. Although the user can install these two components on a Docker image, it will not boot on a real or virtual machine because it does not offer a physical disk image and the bootloader cannot setup the MBR (Master Boot Record). The user has to make a proper root file system on a physical or virtual disk and setup a bootloader and Linux kernel correctly with the contents of Docker image.

On the other hand, the overhead caused by the kernel becomes a noticeable problem. For example, Arrakis[27] showed that nearly 70% of network latency was spent in the network stack in a Linux kernel. To address this problem, technologies that bypass the network stack in a Linux kernel have been proposed (e.g., mTCP[15], Direct Code Execution[36]). Also, there is a trend to create Library OSes (e.g., OSv[18], Unikernels[20], Drawbridge[28]) that reduce kernel overhead for an application by eliminating unnecessary functions. However, these technologies require customization of applications and are not easy to use. The customization of the Linux kernel for an application is a realistic alternative in many cases [1,37].

The overhead problem is also caused by general purpose kernels of Linux distributions that are compiled with trouble-free configurations and set up with kernel options that are stable for most applications. However, some applications do not achieve their expected performance by using the default kernel and options. In order to solve this problem, Liquorix[19] offers an alternative kernel for multimedia on Ubuntu. This alternative kernel is configured with hard kernel preemption, budget fair queue, preemptible tree-based hierarchical RCU, smaller TX net queues, etc.

Hadoop is one example that requires suitable kernel options. Transparent Huge Page (THP) is a memory optimization to mix huge page (2MB) and normal page (4KB) and is designed to reduce the demands on TLB for good performance, but this is application dependent. Performance degradation of Hadoop with THP was found on Red Hat Enterprise Linux 6.2. Some home pages that include Hadoop vendor "Cloudera" warned users to disable THP [7]. Performance degradation was also caused by THP on other applications (e.g., MongoDB, Splunk) on other distributions.

@ 2016 IEEE
DOI 10.1109/HPCC-SmartCity-DSS.2016.7

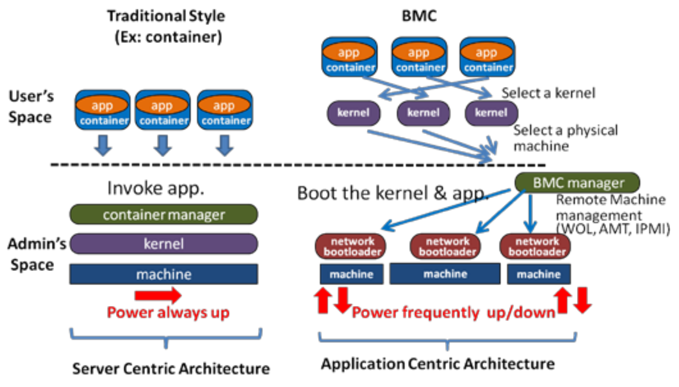


Fig.1. Invocation styles.

Power consumption is also a big problem in HPC. Administrators and device vendors have focused on reducing power consumption; however, users cannot readily know the impact of power consumption and cannot customize their computing environments. For example, some organizations offer Power Usage Effectiveness (PUE) data, but this only shows the rate of IT devices' power consumption against the total infrastructure. There is no data regarding the optimal system configuration that is the best for a given user's applications. Even if users want to set up their computing environment for their application, it is not allowed. One reason for this is that most organizations aim to host many applications on a server, which is called multi-tenancy, server consolidation or system-centric architecture. The servers have a particular kernel with fixed configuration that is unalterable, even if application performance would be improved by kernel customization.

In order to solve these problems, we developed Bare-Metal Container (BMC). BMC offers an environment to boot a container image (i.e., Docker image) with a customized kernel on a suitable remote physical machine, giving an application full performance benefits. It is implemented by remote physical machine management technologies (WakeupOnLAN, Intel AMT, and IPMI), and a network bootloader. BMC also offers information about power consumption on each machine, allowing users to select a machine with a suitable kernel if low power consumption is a requirement.

Although, BMC has increased overhead due to booting a kernel for each application and does not allow multi-tenancy on a machine, BMC offers a better environment for an application. BMC allows the full power of the machine to be used and a customized kernel for each application, both of which are advantageous features for HPC. Moreover, our results show the improvement caused by kernel optimization outweigh the overhead caused by BMC.

Figure 1 shows the difference in application invocation between traditional style and BMC. The traditional style is a system-centric architecture as it assumes the system software is fixed, running all the time, and unalterable by users. In contrast, BMC allows users to change the kernel and the machine for each application, which we describe as an application-centric architecture.

The contributions of this paper are the following:

- BMC compensates for shortcomings of Docker, and allows an application (i.e., Docker image) to select the suitable kernel and machine. It recognizes kernel as a part of an application and encourages application-centric architecture.
- BMC supplies power consumption data to the user. The user can select a Linux kernel and physical machine to achieve the desired balance of performance and power consumption for an application.
- BMC offers high usability because it is compatible with Docker. BMC downloads a Docker image from the Docker Hub. BMC also offers Hub for Linux kernels, and the users can share the optimized Linux kernel.
- BMC is designed for used on LAN and the Internet. It has two modes to obtain a root file system: NFS mode for secure LAN and RAMFS mode for insecure Internet.
- The experiments showed the performance improvement by replacing kernel optimizations which were for CPU (Hyper Threading), memory (Transparent Huge Pages), and network (Receive Flow Steering) on 4 type of CPUs(Celeron, Core2Duo, i7, and Xeon). The improved performance could surpass the overhead caused by BMC.

This paper is organized as follows. Section II gives a brief description of works related to BMC. Sections III, IV, and V describe the element technologies used by BMC, including remote machine management mechanism, network bootloader, and power consumption. Section VI outlines the design of BMC. Section VII presents the current implementation and reports experiments on BMC. Section VIII discusses future works, and Section IV summarizes our conclusions.

## II. Related Work

Docker is popular for HPC and there are some special customizations. For example, Lawrence Berkeley National Lab has developed Shifter[14] for Cray XC and XE. IBM also offers LSF(Load Sharing Facility)-Docker for HPC [21]. They have mechanisms to modify kernel settings; however, the mechanisms are limited and do not allow changes to the kernel itself because it assumes the management software runs on the target machines. BMC, in contrast, offers a suitable Linux kernel for a Docker image and boots it on a remote machine.

There are many researches that boots remote machines for testing system software on a bare metal environment. For example, most network testbeds (Emulab[12], CloudLab[8], and Chameleon[5], etc.) are used for this purpose, and LinuxBIOS/BProc Cluster[6] and Onesis[23] are designed to test operating systems. These systems require a bootable disk image, which does not separate user space and kernel clearly. On the other hand, BMC is designed to test the affinity between a user space application (i.e., Docker image) and kernel focusing on performance and power consumption.

The hosting style of BMC resembles bare-metal cloud technologies (e.g., NoHype[17], OpenStack Ironic[24], SoftLayer's bare-metal servers[32]), but a key difference is that BMC allows an application to select a suitable remote machine and kernel, which is a temporary computing environment for container technology. Bare-metal cloud technologies assume that an OS is installed on a remote machine's disk and is reused for a long period of time. BMC also resembles OS deployment technologies (e.g., OS stream deployment[4], BMCast[22]), but these technologies do not allow kernel customization.

## III. Remote Machine Management

Current machines have a remote machine management mechanism that allows power-on and power-off from a remote machine. BMC uses three remote machine management mechanisms; WakeOnLAN (WOL), Intel Active Management Technology (AMT) [14], and Intelligent Platform Management Interface (IPMI). Table I shows the differences between these technologies.

TABLE I. Remote Machine Management Technologies.

| | **WOL** | **AMT** | **IPMI** |
|---|---|---|---|
| Protocol | Magic Packet<br>MAC | HTTPS<br>IP | RMPC<br>IP |
| Power-On | ✔ | ✔ | ✔ |
| Power-Off | × | ✔ | ✔ |
| Security | × | Password | Password |
| Comment | Most PCs have WOL. | High level Intel machine | Server Machine (Slow BIOS) |

WOL is the most popular remote machine management mechanism because most BIOS support it. WOL is based on a protocol called "magic packet" that is broadcasted on the data link layer, and the corresponding NIC powers on the machine. Most Linux distributions offer this management tool named "etherwake". Unfortunately, WOL does not support physical machine shut down and status confirmation. Thus, the sender of WOL cannot know if power-on is successful or not. Therefore BMC has no recovery mechanism using WOL. Moreover, the lack of authentication in WOL makes it less secure. In addition, the use of data link layer protocol makes it difficult to extend to the Internet, and BMC uses WOL on secure private LAN only.

Intel Active Management Technology (AMT) [16] is a part of Intel vPro technology equipped in high level Intel machines. Intel AMT is activated through BIOS, requiring both a password for user authentication and a static IP address for HTTPS communication. If the IP address is reachable via the Internet, the machine can be controlled worldwide. Most Linux distributions offer this management tool named "amttool". The power-off mechanism in this tool does not account for the status of the running OS. It terminates the machine immediately and may cause trouble in the normal OS installed in a hard disk. However, BMC loads an OS image in memory and uses it only for the duration of the application. BMC does not need to account for the OS after the application finishes.

The Intelligent Platform Management Interface (IPMI) is a remote machine management mechanism used on server machines. The implementation of IPMI is different from Intel AMT, but the function is similar from the BMC perspective. Most Linux distributions offer this management tool named "ipmitool". In order to set up IPMI, the BIOS requires a password and a static IP address, similar to Intel AMT. The static IP address is used for communication with Remote Management Control Protocol (RMCP). The only difference between Intel AMT and IPMI is that the BIOS with IPMI is generally slower because it is equipped with a server machine and has many monitoring functions.

## IV. Network Bootloader

Network boot is an essential technology for diskless machines. PXE (Preboot eXecution Environment) is the most popular method, but depends on TFTP (Trivial File Transfer Protocol) which is insecure. Instead of PXE, BMC uses iPXE[13] which is an open source network bootloader forked from gPXE. iPXE can download a kernel and initrd (i.e., initial ramdisk to prepare a root file system) with HTTP/HTTPS protocols. BMC uses HTTPS for unsecure environment (i.e., Internet) and HTTP for secure environment (i.e., private LAN).

In order to use HTTP/HTTPS, iPXE requires an IP address. Although iPXE can obtain an IP address dynamically by DHCP, the BMC manager must know which IP address is used for iPXE. Therefore, a static IP address is assigned to the iPXE on a remote machine. Furthermore, in order to use HTTPS, iPXE must include a server certification offered by a Certification Authority. BMC also uses the iPXE scripting language to control of the boot procedure by changing the Linux kernel and initrd for each machine.

## V. Power Consumption

Power consumption is one of the important topics in HPC, but most administrators focus on the effectiveness of IT devices' energy usage against the total facility energy, including air conditioning. As mentioned previously, many organizations are evaluated by PUE, which is from an administrator perspective. From the user perspective, accurate measurement of the power consumption for an application is important, because the amount of power consumed directly impacts the cost of running their applications. Unfortunately, measuring power consumption for an application is very difficult because of the multi-tenant computing model, where many applications run simultaneously on multi-core servers. Although the workload manger SLURM [31] has a mechanism to measure and cap the power consumption, SLURM uses the function offered by CRAY system.

Fortunately, some super computer centers lend hardware on a per time basis. In this scenario, an application occupies the physical machine and allows the user to measure the power consumption for the application. Since BMC assigns one application to one machine, this type of energy accounting scenario is applicable. The BMC manager measures the power consumption for each remote machine during power-on and power-off caused by remote machine management technology.

## VI. Bare-Metal Container

As shown in Figure 1, the BMC manager controls the Docker image, kernel, and remote machine with a remote machine management mechanism and the network bootloader iPXE. This section outlines the important functions of BMC.

### A. Seamless from/to Docker

BMC utilizes Docker for container technology because most Docker images include the procedure to boot a Linux kernel. However, these Docker images do not include Linux kernel and bootloader (e.g., GRUB). The reason is that most Docker images are created from application packages. For example, Docker offers the mechanism to create a Docker image using a "dockerfile", which is a blueprint of the image. The following is one example of a dockerfile based on CentOS packages:

```
FROM centos:7
RUN yum -y install openssh-server openssh-clients
```

The created Docker image has /etc directory and descriptions to boot daemons because each application package has each description to invoke. BMC utilizes these descriptions to boot specific components of Linux, such as the SSH server daemon for making remote procedure calls.

The Docker images are slightly customized for BMC, and these changes are applied dynamically. The customization sets some configuration files under /etc and adds SSH's authorized_keys for a remote procedure call. This modification does not affect Docker, and the customized images are executable on Docker and BMC. The customized Docker images are uploaded to or downloaded from Docker Hub to be shared by Docker and BMC users.

### B. NFS Mode and RAMFS Mode

BMC is designed for private LAN and the Internet, with two modes: NFS mode and RAMFS mode.

NFS mode is designed to be used in a private LAN. Most private LANs are firewall protected and assumed to be a secure network environment. BMC uses HTTP to get a kernel and initrd with iPXE and NFS to mount a root file system. NFS mode can reduce network traffic at boot time because the booting OS does not require all files; however, NFS mode does not scale well because of the capacity of the NFS server.

RAMFS mode is designed to be used for the Internet. Due to the insecure nature of the Internet, RAMFS mode uses the secure protocol HTTPS to get the Linux kernel and initrd with iPXE. RAMFS mode copies the Docker image to the machine's ramfs because the latency of the Internet is greater than private LAN making it unsuitable for using a remote file system (e.g., NFS, SSHFS). Unfortunately, time is needed to copy the whole Docker image, and memory resources are used to keep the root file system. Despite these drawbacks, the OS runs in a standalone mode, has no server dependency, is scalable, and is suitable for the Internet. Either rysnc or scp can be used to copy the image, but they require a password, and the password management on iPXE is not easy. The current implementation uses "wget" and "tar" commands, because the Docker image is compressed by tar and gzip on BMC manager and downloaded through HTTPS.

### C. Customizing initrd

Many Linux distributions create initrd with the "mkinitramfs" command for their kernel package. The created initrd has the option to treat NFS and the local disk as a root file system. The option is passed by the kernel argument "boot=nfs|local". BMC uses the default nfs for NFS mode but customizes local for RAMFS mode. On the customized initrd, the local does not use local storage. If "boot=local" is designated, the local mode copies the Docker image to the ramfs using "wget" and "tar" commands. After copying, the initrd mounts the ramfs as the root file system and passes the control to the boot procedure in the Docker image.

### D. BMC Hub

One of the main reasons for the popularity of Docker is the use of Docker Hub to share images. Docker Hub offers many useful images, and users can select one that suits their needs. It avoids installation and facilitates the sharing of the same settings for an application.

BMC also offers the BMC Hub to share kernels and initrds. The BMC Hub helps users avoid the chore of customizing a kernel and initrd. Owing to the BMC Hub, normal users only select suitable kernel and intrd for their applications on their target machine. The current BMC Hub uses BitBucket.

### E. Power Measurement

As previously stated, power consumption is generally not recognized by users. BMC offers power consumption data for an application. It is important data when evaluating the effect of kernel optimization on the application. The current implementation uses WattChecker offered by Osaki Electric[25], to measure power consumption of a remote node.

Some data about power consumption are not related to an application. For example, BIOS, the boot procedure, and the terminate procedure are extra power consumption values. In order to concentrate on the application, the BMC manager is informed about the start and end of an application. A few daemons are still running, but their power consumption is considered to be negligible.

### F. Early Boot

On AMT or IPMI, early boot (speculative execution) is used by default. Early boot enables a remote machine to boot before all materials are prepared because these remote machine managements allow the termination of the remote machine and roll back when preparation fails. If the node uses WOL, the power-on must wait until all materials are prepared because WOL lacks a termination mechanism.

### G. Application Invocation Procedure on BMC

Figure 2 shows the stages that launch an application on a remote machine with BMC. Each node machine uses three stages to boot Linux: power control, network bootloader, and Linux boot procedure (stage ③, ④, and ⑤, respectively). The three stages are distinguished by MAC address or IP address to manage the machine and its status. The boot with WOL uses one MAC address and two IP addresses. The boot with Intel

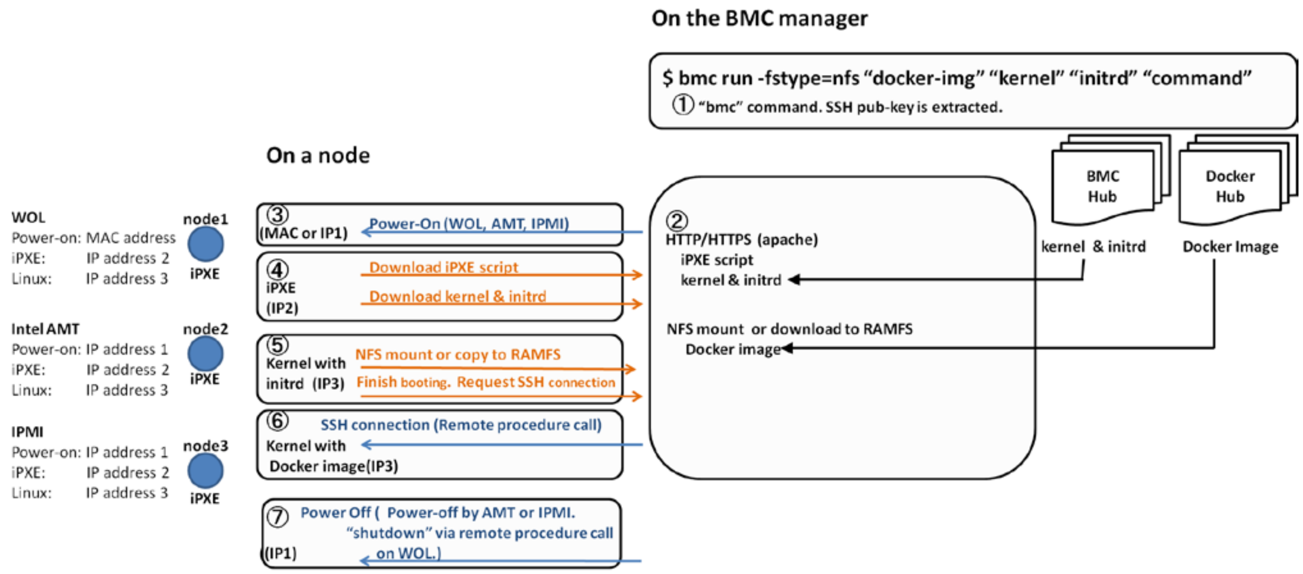


Fig 2. Process invocation procedure on BMC.

AMT or IPMI uses three IP addresses. The information is registered in the BMC manager in advance.

Stage ① is where the "bmc" command invokes the BMC manager. The command has several arguments, following the Docker style. The arguments are file system mode (NFS or RAMFS), "Docker image", "Linux kernel", "initrd" , and "command". If the user wants to designate a node machine, the node number is added. If not, an anonymous node is assigned by the BMC manager. The BMC manager also retrieves the user's public SSH key for remote procedure calls.

In stage ②, the BMC manager checks the availability of "Docker image", "Linux kernel", and "initrd". The Docker image is downloaded from DockerHub if not previously done. The Docker image is exposed for NFS mode or compressed by tar and gz for RAMFS mode. If the target node uses Intel AMT or IPMI, this stage is skipped, and it moves to stage ③ immediately as an early boot, in order to reduce boot overhead. This stage is performed in the background until stage ④.

Stage ③ is called the power control stage, where the BMC manager powers up a remote node machine by WOL using MAC address or Intel AMT or IPMI using IP address. After that, the BIOS runs and launches the network bootloader iPXE. If the power-on fails with Intel AMT and IPMI, the status is returned to the BMC manager, and it attempts to boot again.

At stage ④, the network bootloader stage, iPXE boots with a static IP address and gets an iPXE script form the HTTP server. The iPXE script designates the Linux kernel and initrd. iPXE executes the script and downloads the Linux kernel and initrd using HTTP or HTTPS. The iPXE script also includes kernel arguments to control the boot process. One of the kernel arguments designates the NFS or RAMFS mode.

In stage ⑤, the Linux boot stage, the booting Linux kernel sets up a static IP address, which is passed by the kernel arguments. It selects NFS or RAMFS for the root file system, according to the designated mode. After the mounting of the root file system, initrd passes the control to /etc/init of the root file system (Docker image), which is Linux boot procedure. At the end of /etc/init, it sends the boot-up message to the BMC manager.

At stage ⑥, the BMC manager gets the boot-up message and subsequently confirms the opening of the SSH port by the "nc" command. If the port is not opened, it retries. If the port is opened, it runs the designated "command" with the remote procedure call of SSH.

In stage ⑦, after the remote procedure call is finished, the BMC manager terminates the node machine. If the machine uses Intel AMT or IPMI, the machine is terminated by either one. If the machine uses WOL, the BMC manager uses the remote procedure call to shutdown the machine.

## VII. Implementation and Evaluation

The current BMC manager is implemented by the shell script. The code size is about 2400 LOC. The BMC manager requires Docker, Apache for HTTP or HTTPS server, and remote machine management tools (etherwake, amttool, and ipmitool for WOL, Intel AMT, and IPMI, respectively). The BMC manager knows the MAC address for WOL, and the static IP addresses for Intel AMT, IPMI, iPXE, and Linux. A remote node must be set up with WOL, Intel AMT, or IPMI. iPXE must be installed on the first boot device.

In order to show the feasibility of BMC, we evaluated the overhead caused by BMC and the application's performance improved by kernel optimizations. The target machines are Lowpower PC (Celeron), Note PC (i7), Desktop PC (Core2Quad) and Server PC (Xeon) as listed in Table II. Although the issue dates and configurations are different, it is sufficient to demonstrate the applicability of BMC.

TABLE II. SPECIFICATION OF MACHINES.

| | Remote machine management | CPU,Core/thread,Clock (Burst time), Power | Logical performance GFLOPS (Burst time) | Issue date | Memory | NIC (queue) |
|---|---|---|---|---|---|---|
| Low Power Intel NUC 5CPYH | WOL | Celeron (N3050), 2/2, 1.6 (2.16)GHz,8W | 6.4 (8.6) | 2015 | 8GB | RealTek r8169 (1) |
| NotePC Lenovo ThinkPAD T430s | Intel AMT | i7 (3520M) 2/4, 2.9(3.6)GHz, 35W | 46.4 (57.6) | 2012 | 16GB | Intel e1000 (1) |
| DesktopPC Dell Optiplex 960 | Intel AMT | Core 2Quad (Q9400) 4 /4, 2.66GHz,95W | 42.656 | 2008 | 16GB | Intel e1000 (1) |
| Server Dell PowerEdge T410 | IPMI | Xeon (X5650) 6/12, 2.66(3.06)GHz,95W | 63.984 (73.44) | 2010 | 8GB | Broadcom NeXtreme II (8) |

Our analysis consisted of measuring the main overheads of BMC (time, network traffic, and power consumption at boot time) and analyzing the boot sequence. After that, three types of kernel optimizations were applied to the applications. The kernel optimizations were for CPU (Hyper Threading), memory (Transparent Huge Pages), and network (Receive Flow Steering). Every test case, except for boot time, was measured five times, and the results present the average for each test case.

## A. *Overhead of BMC*

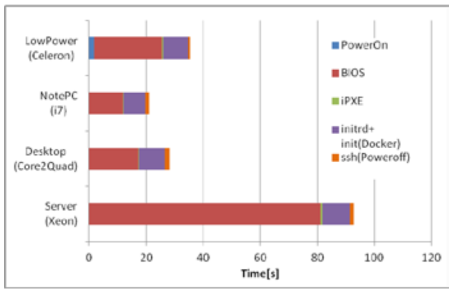
The elapsed time, network traffic, and power consumption at booting stage were measured on four machines. The Linux kernel was 3.13.11, and the initrd was customized for the kernel. The size of the Linux kernel and initrd was 5.93MB and 8.35MB, respectively. The Docker image was based on CentOS created by the dockerfile which is described in Section VI-A. The tar.gz file for RAMFS mode was 61.97MB. The command was “date”, and the execution of the remote procedure call terminated immediately.

### *1) Elapsed Time*

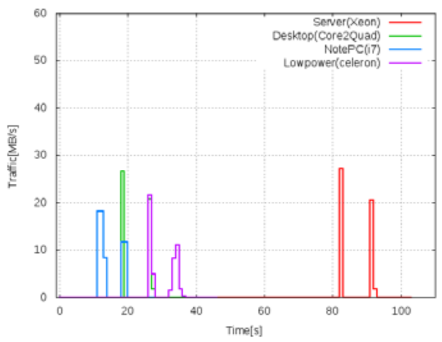
Figure 3(a) and 3(b) show the elapsed time at booting stage on NFS and RAMFS, respectively. The elapsed time consists of six parts: ①Power On, ②BIOS, ③iPXE, ④initrd, ⑤boot procedure “init” on Docker image, and ⑥ command executed by remote procedure call with SSH and termination time. The timing of NFS mount is not distinguished by the BMC manager, and the time for “init” is merged to initrd (“initrd+init” in the figure).

The results showed that the majority of elapsed time was consumed by the BIOS and initrd (initrd+init on NFS), especially for the BIOS on the server configuration. The reason for this is that the sever BIOS had many monitoring functions. The initrd on RAMFS included copying the root file system from the HTTPS server. The initrd+init on NFS included reading a file from NFS. Thus, both initrd and initrd+init required network traffic and took additional time.

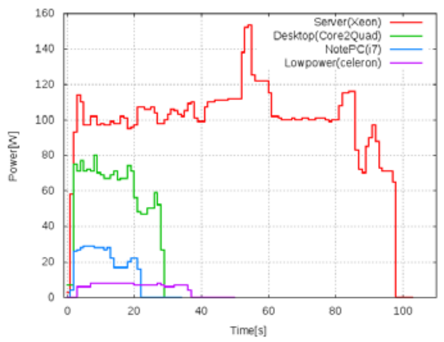
iPXE on RAMFS took significantly longer time than on NFS because of the slow downloading by HTTPS. This is further supported by evidence in the following section on



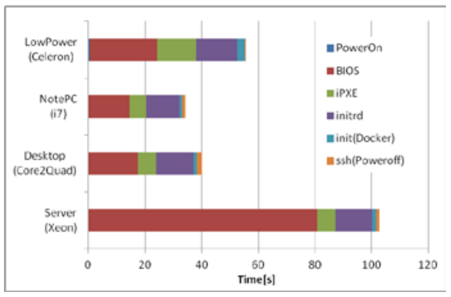
(a) NFS



(a) NFS

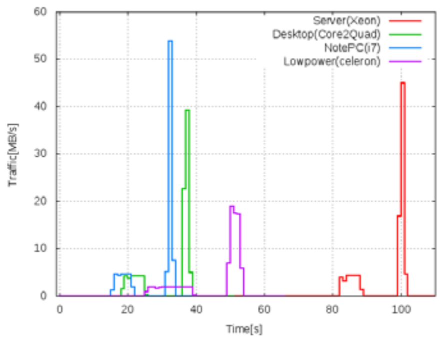


(a) NFS



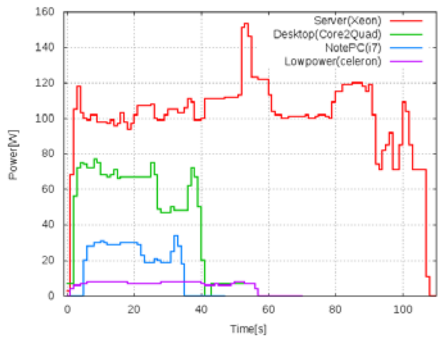
(b) RAMFS

Fig. 3. Elapsed Time at Boot Time.



(b) RAMFS

Fig. 4. Network Traffic at Boot Time.



(b) RAMFS

Fig.5. Power Consumption at Boot Time.

network traffic. The time consumed by power-on on low power PC with NFS took a few seconds because WOL did not allow early boot and the NFS needed time to prepare.

### 2) *Network Traffic*

Figure 4(a) and 4(b) show the network traffic at booting stage on NFS and RAMFS respectively. Each machine has 1Gbps NIC. The figures show two traffic peaks for each configuration on boot. The first peak was caused by the downloading of the kernel (5.93MB) and initrd (18.35MB) at iPXE. The kernel and initrd were downloaded by HTTP on NFS mode and by HTTPS on RAMFS mode. The results showed that the downloading on RAMFS took a longer time than NFS. The throughputs were nearly 20 MB/s on NFS, but it was less than 5MB/s on RAMFS. The difference was caused by the ability of protocol stack of iPXE for HTTP/HTTPS.

The second peaks on RAMFS were caused by HTTPS with the "wget" command, and the throughputs were more than 20MB/s. The results showed that the implementation of HTTPS on iPXE was not as mature as wget. As mentioned in the previous section, this is the reason for the long elapsed time at iPXE on RAMFS. The throughputs of the second peak on NFS were consistently less than the first peak. The reason is that the traffic was caused by multiple file accesses on NFS at the boot procedure. Each file access was short and prevented high throughput.

### 3) *Power Consumption*

Figure 5(a) and 5(b) show the power consumption at booting stage on NFS and RAMFS respectively. These times were longer than the elapsed times in Figure 3 because the time for power consumption included time for the power to return to nearly 0 W.

These results illustrated the feature of each machine configuration. The server took the most power (>100W), and the low power PC took the least power (<10W). However, the direct comparison between NFS and RAMFS is difficult because the performance potentials are different. The case is shown in the "Optimization for CPU" Section.

### 4) *Summary*

Table III shows the total figures for elapsed time, power consumption and network traffic. Each result on NFS was smaller than RAMFS, suggesting that NFS was more efficient than RAMFS. However, NFS mode is limited to private LAN to maintain security and scalability. Additionally, the performance of the application is affected because NFS mode is not standalone and all access to the root file system must go through the network. For these reasons, the following performance measurements were done in RAMFS mode.

TABLE III. TOTAL CONSUMPTIONS ON BMC BOOT

| | Time(s) | Power (j) | Traffic (MB) |
|---|---|---|---|
| NFS | | | |
| Lowpower | 35.4 | 242 | 49.5 |
| Note | 20.9 | 481 | 49.1 |
| Desktop | 28.1 | 1,773 | 49.3 |
| Server | 92.6 | 9,932 | 49.8 |
| RAMFS | | | |
| Lowpower | 55.6 | 402 | 92.9 |
| Note | 34.3 | 775 | 92.8 |
| Desktop | 40.0 | 2,493 | 92.8 |
| Server | 102.7 | 11,015 | 92.6 |

## B. *Optimization for CPU: Hyper Threading*

Normal Linux distributions utilize hyper threading technology (HTT), but Goto, the developer of GotoBlas, said "Hyper-thread is harmful"[30] and it is common practice to turn off HTT for numerical computation. BMC tried to confirm the effect of HTT on each architecture. The experiment uses the OpenBlas library and calculates the matrix multiplication. The OpenBlas binary is compiled for each architecture and confirmed the performance with and without HTT of Linux kernel. The HTT was changed as follows:

```
Turn on core 1
# echo 1 > /sys/devices/system/cpu/cpu1cpu/online
Turn off core 1
# echo 0 > /sys/devices/system/cpu/cpu1cpu/online
```

The limited memory of the test machines did not allow the calculation of a large matrix. Therefore, the experiment was the result of 10 times of matrix multiplications. The size of the matrix varied from [1600:1600] to [12800:12800]. Celeron and Core2Quad configurations did not have HTT, and the performance measurement for HTT-off was examined on i7 and Xeon. The number of HTT on a core is 2 as described in Table II.

TABLE IV. TOTAL TIME, POWER, GFLOPS, AND EFFICIENCY ON EACH MACHINE WITH/WITHOUT HYPER THREAD TECHNOLOGY AT 10 TIMES MULTIPLICATION OF [12800:12800] MATRIX. THE PARENTHESIS AT GFLOPS SHOWS PERCENTAGE FROM LOGICAL GFLOPS AS SHOWN IN TABLE II.

| | **Time (s)** | **Power (j)** | **GFLOPS** | **Power/ (GFLOPS *time)** |
|---|---|---|---|---|
| Celeron | 12,783.8 | 125,084 | 2.99 (34.7%) | 3.27 |
| i7 HTT-on | 961.4 | 55,315 | 43.8 (76.0%) | 1.31 |
| i7 HTT-off | 827.1 | 45,364 | 50.9 (88.4%) | 1.08 |
| Core2Quad | 1,060.2 | 140,346 | 39.8 (93.3%) | 3.32 |
| Xeon HTT-on | 945.6 | 211,908 | 44.6 (60.7%) | 5.02 |
| Xeon HTT-off | 698.9 | 151,760 | 60.5 (82.4%) | 3.59 |

Table IV shows the figures of the time, power, GFLOPS, and efficiency (Power/(GFLOPS*time)) of 10 times multiplication of [12800:12800] matrix. The column for efficiency shows the power used by 1Giga Floating Point Operations, where a smaller value is better. The results showed that the Xeon with HTT-off had the best total time, but the i7 with HTT-off had the lowest power consumption. Interestingly, the results also suggested that the Celeron was not a low power CPU, as it was only the third lowest in power consumption and had the longest total time. One reason for this was that the OpenBLAS reached 34.7% of logical GFLOPS only, which was less than half of the other CPUs, indicating that it could not extract the full performance from the CPU. On the other hand, the OpenBLAS on Core2Quad reached 93.3% of logical GFLOPS, indicating that the effect of recompilation was the best on the Core2Quad.

TABLE V. OVERHEAD AND IMPROVEMENT CAUSED BY HTT-OFF.

| | **Boot overhead** | **Improvement at [6400:6400]** | **Improvement at [12800:12800]** |
|---|---|---|---|
| Tim e(sec) | | | |
| i7 | 35.4 | 15.9 | 134.3 |
| Xeon | 108.0 | 29.8 | 246.7 |
| Power (joule) | | | |
| i7 | 1,805.3 | 1,150 | 9,951 |
| Xeon | 11,274.5 | 6,792 | 60,148 |

Table V shows the overhead caused by BMC and the improvements caused by HTT-off. The results showed that the improvement on both i7 and Xeon surpassed the overhead for both time and power between [6400:6400] and [12800:12800]. This supports the idea that the kernel optimization used by BMC is useful at larger than 12800 matrix multiplication.

## C. *Optimization for Memory: Transparent Huge Pages*

Transparent Huge Pages (HTP) is a memory optimization which mixes huge page (2MB) and normal page (4KB). Huge pages reduce demands on TLB and achieve good performance. As mentioned in the introduction, the performance is highly application dependent. Some performance degradation was found in Linux 2.6.32, but it is not clear if this is the same on other Linux versions. In this section, BMC attempted to confirm the effect of HTP on Linux 3.13.3 with a Redis benchmark, which is also reported as an affected application. The configuration of HTP was set as follows:

```
HTP-on
# cd /sys/kernel/mm/transparent_hugepage/
# echo always > enabled
# echo 0 > khugepaged/scan_sleep_millisecs
HTP-off
# cd /sys/kernel/mm/transparent_hugepage/
# echo never > enabled
```

The Redis benchmark set the key space length to 50,000 and the total number of requests from 10,000,000 to 80,000,000.

TABLE VI. OVERHEAD AND IMPROVEMENT CAUSED BY HTP-OFF.

| | **Boot overhead** | **Improvement at 20,000,000** | **Improvement at 40,000,000** |
|---|---|---|---|
| Time(sec) | | | |
| Celeron | 56.6 | 12.0 | 26.6 |
| i7 | 31.0 | 26.1 | 56.5 |
| Core2Quad | 37.9 | -5.6 | -27.3 |
| Xeon | 100.8 | 4.4 | 4.6 |
| | | | |
| Power (joule) | | | |
| Celeron | 444.0 | 1279.3 | 2216.0 |
| i7 | 688.7 | 1215.9 | 2681.4 |
| Core2Quad | 2321.1 | 336.0 | -1650.2 |
| Xeon | 10476.1 | 1279.3 | 2216.0 |

Table VI shows the overhead caused by BMC and the improvement caused by HTP-off. The table showed the problem size when BMC's overhead was surpassed by HTP-off. The result on i7 showed BMC's time overhead was surpassed between 20,000,000 and 40,000,000 requests. The power overhead was surpassed before 20,000,000 requests. The results on Celeron also showed the time overhead was surpassed between 40,000,000 and 80,000,000 requests. The power overhead was surpassed before 20,000,000 requests. However, the time and power improvement on Xeon was not as good, and it will take more than 80,000,000 requests to see an improvement. The improvement on Core2Quad was the reverse, indicating that Core2Quad configurations should use HPT-on. These results are especially important for Redis user on each CPU.

## D. *Optimization for Network: Receive Flow Steering*

Current CPUs have multiple cores, but the interrupts from the NIC are assigned to only one core, leading to performance degradation. Some high specification NICs have Receive Packet Steering (RSS), which distributes the interrupts across cores, but it depends on the NIC. A Linux kernel has a similar implementation, called Receive Flow Steering (RFS), to distribute interrupts to cores. However, the effect is not clear because it depends on NIC and applications. In this section, BMC attempted to confirm the effect of RFS on Linux 3.13.3 using an Apache benchmark. The configuration of RFS was set as follows:

```
RFS-on
# cd /sys/class/net/eth0/queues/rx-0
# echo "f" >rps_cpus
# echo 32768 >rps_flow_cnt
# echo 32768 >/proc/sys/net/core/rps_sock_flow_entries
RFS-off
# cd /sys/class/net/eth0/queues/rx-0
# echo "0" >rps_cpus
# echo 0 >rps_flow_cnt
# echo 0 >/proc/sys/net/core/rps_sock_flow_entries
```

The Apache benchmark set the concurrency to 100 and the number of requests from 1,000,000 to 8,000,000.

TABLE VII. OVERHEAD AND IMPROVEMENT CAUSED BY RFS-OFF.

| | **Boot overhead** | **Improvement at 2,000,000** | **Improvement at 4,000,000** |
|---|---|---|---|
| Time(sec) | | | |
| Celeron | 58.5 | 73.7 | 141.3 |
| i7 | 30.9 | 20.1 | 41.9 |
| Core2Quad | 38.8 | 27.5 | 56.2 |
| Xeon | 101.9 | 0.02 | -0.24 |
| Power (joule) | | | |
| Celeron | 459.9 | 341.3 | 662.4 |
| i7 | 706.6 | 842.5 | 1,821.5 |
| Core2Quad | 2,406.5 | 2,291.8 | 4,736.4 |
| Xeon | 10,686.3 | 172.7 | 505.2 |

Table VII shows the overhead caused by BMC and the improvement caused by RFS-off. The results show that the time and power improvements by RFS-off surpassed the overhead at around 2,000,000 requests on Celeron, i7, and Core2Quad. However, for Xeon, there was a paradoxical effect on time and power, which is under investigation at this moment.

## VIII. Discussion

### A. *BMC is more secure than container technogies.*

When some containers are hosed on a physical machine security concerns arise. For examples, Cross-VM attack[29,34,35] and hyper thread attack [26] are famous attacks on multi-tenant servers. One distinct advantage of BMC is that it assigns only one application to one physical machine, mitigating security concerns on multi-tenant servers.

Another important security concern is software aging on non-stop servers. They prevent updating the kernel even if it has vulnerabilities. BMC renews all software for each trial and encourages software rejuvenation.

In addition, the continuous operation of the kernel may cause critical data to remain in memory for a long time [2,3]. The advantage of BMC is that it resets the power and boots a kernel for each application, clearing the data in the memory.

### B. *Limitations on Current BMC*

In comparison to container technologies, current BMC has some limitations for its usage. One limitation is that it has no mechanism to create a cluster environment as “Docker Swarm”. Users need to allocate nodes one by one to create cluster environment. In addition, BMC does not support network isolation for the cluster environment, although a Docker image runs on the secure communication between the BMC manager and a node on RAMFS mode. Another limitation is that RAMFS mode has size limitation to accept a root file system because it uses memory file system. BMC needs to use compression for the memory file system on a small memory machine. They are issues for the next version.

### C. *How to deal with Knowledge of Optimization*

BMC can get much data for optimization, but current BMC has no mechanism to use these types of data. Most of the optimization is performed manually by users. As a next step, we will implement an automatic optimization mechanism on BMC. The mechanism will test the combinations of kernel optimizations on different machines for applications. The knowledge gained from these automatic tests will be applied through machine learning algorithms and used for the selection of kernels and machines for BMC.

### D. *Performance Improvement*

The main overhead on BMC is caused by BIOS. Unfortunately, BIOS is a part of the hardware in general and cannot be customized. However, some machines accept the open source BIOS “coreboot”[9] and that allows BIOS customization. We will utilize coreboot to reduce the BIOS overhead. Furthermore, iPXE [13] is designed to be embedded in NIC, and we will try the NIC embedded iPXE as next step. We hope the combination of coreboot and embedded iPXE will boot within a few seconds on BMC.

### E. *Do we need a traditional kernel?*

Current BMC uses a Linux kernel to run a Docker image on a remote machine. It is useful, but Linux includes many unused functions and causes slow boot because Linux is designed for multi-user and long-run usage. In order to solve this problem, Docker Inc. acquired Unikernel Systems in January 2016. Unikernels [20] is one of the library OS and aims to reduce OS overhead for an application. It is a good trend for application-centric architecture, and we hope BMC treats library OS and becomes a driving force to change from traditional kernel to Library OS.

## IX. CONCLUSION

Bare-Metal Container offers an environment to run a Docker image with a suitable Linux kernel on a remote physical machine. The Docker image is allowed to change the kernel and its settings. As a result, the application extracts the full performance of the physical machine. This mechanism allows changes to both the kernel and machine for optimum performance for a given application. We believe it will encourage the idea of application-centric architecture.

BMC is complementary, not a replacement, to Docker because Docker offers quick execution, which is useful for trials. BMC aims to be used for a high performance computing environment where the booting overhead is compensated by the kernel optimization. The evaluation highlighted the effects of kernel optimization for CPU (Hyper Threading), memory (Transparent Huge Pages), and network (Receive Flow Steering). The results showed the improved performances surpassed the overhead caused by BMC.

BMC encourages the trend of Library OS, which eliminates functions and overheads because the OS launched by BMC is sufficient to host only one application. Future versions of BMC will include Library OS.

## Acknowledgment

We thank our colleagues from Cyber Physical Cloud group of Information Technology Research Institute (ITRI) of the National Institute of Advanced Industrial Science and Technology (AIST). They gave us many comments that assisted the research and improved the manuscript.

This paper is partially based on results obtained from a project commissioned by the New Energy and Industrial Technology Development Organization (NEDO), Japan.

## References

[1] S. Boyd-Wickizer , A.T. Clements, Y. Mao, A. Pesterev, M.F. Kaashoek, R. Morris, and N. Zeldovich, An Analysis of Linux Scalability to Many Cores, USENIX Symposium on Operating Systems Design and Implementation (OSDI), 2010.

[2] J. Chow, B. Pfaff, T. Garfinkel, K. Christopher, and M. Rosenblum, Understanding data lifetime via whole system simulation, USENIX Security, 2004.

[3] J. Chow, B. Pfaff, T. Garfinkel, and M. Rosenblum, Shredding your garbage: reducing data lifetime through secure deallocation, USENIX Security, 2005.

[4] D. Clerc, L. Garces-Erice, and S. Rooney, OS Streaming Deployment, Performance Computing and Communications Conference (IPCCC), 2010.

[5] Chameleon, https://www.chameleoncloud.org

[6] S.E. Choi, E.A. Hendriks ; R.G. Minnich ; M.J. Sottile and A.J. Marks, Life with Ed: a case study of a linux BIOS/BProc cluster, High Performance Computing Systems and Applications (HPCS), 2002.

[7] Cloudera's HP, http://www.cloudera.com/documentation/enterprise/5-2-x/topics/cdh_admin_performance.html

[8] CloudLab, https://www.cloudlab.us/

[9] coreboot, https://www.coreboot.org/

[10] Docker, https://www.docker.com/

[11] J. Eder, Can you run Intel's Data-plane Development Kit (DPDK) in a Docker container? Yep., Redhat deveoper's blog

http://developers.redhat.com/blog/2015/06/02/can-you-run-intels-data-plane-development-kit-dpdk-in-a-docker-container-yep/

[12] Emulab, https://www.emulab.net/

[13] iPXE, http://ipxe.org

[14] D. Jacobsen, and S. Canon, Contain This, Unleashing Docker for HPC, Cray User Group, 2015.

[15] E.Y. Jeong, S. Wood, M. Jamshed, H. Jeong, S. Ihm, D. Han and K.S. Park, mTCP: a Highly Scalable User-level TCP Stack for Multicore Systems, Symposium on Network Systems Design and Implementation (NDSS), 2014.

[16] G. Hofemeier, and L. Atencio, INTEL® ACTIVE MANAGEMENT TECHNOLOGY (INTEL® AMT) Start Here Guide, Intel, 2011.

[17] E. Keller, J. Szefer, J. Rexford, and R.B, Lee, NoHype: Virtualized Cloud Infrastructure without the Virtualization, International Symposium on Computer Architecture (ISCA) 2010.

[18] A. Kivity, D. Laor, G. Costa, P. Enberg, N. Har'El, D. Marti and V. Zolotarov, OSv - optimizing the operating system for virtual machines, USENIX Annual Technical Conference(ATC), 2014.

[19] Liquorix, https://liquorix.net/

[20] 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, International Conference on Architectural Support for Programming Languages and Operating Systems (ASPLOS), 2013.

[21] B. McMillan and C. Chen, High Performance Docking, IBM platform computing technical white paper, June 2014.

[22] Y. Omote, T. Shinagawa, and K. Kato, Improving Agility and Elasticity in Bare-metal Clouds, International Conference on Architectural Support for Programming Languages and Operating Systems (ASPLOS), 2015.

[23] Onesis, http://onesis.org/

[24] OpenStack's Ironic, https://wiki.openstack.org/wiki/Ironic

[25] Osaki Electic, http://www.osaki.com/tabid/61/Default.aspx?scroll=p1

[26] C. Percival, Cache missing for fun and profit, BSDCan, 2005.

[27] P. Peter, J. Li, I. Zhang, D.R.K. Ports, D. Woos, A. Krishnamurthy, and T. Anderson, and T. Roscoe, Arrakis: The Operating System is the Control Plane, USENIX Symposium on Operating Systems Design and Implementation (OSDI ), 2014.

[28] D.E. Porter, S. Boyd-Wickizer, J. Howell, R. Olinsky, and G. Hunt, Rethinking the Library OS from the Top Down, International Conference on Architectural Support for Programming Languages and Operating Systems (ASPLOS), 2011.

[29] T. Ristenpart, E. Tromer, H. Shacham, and S. Savage, Hey, you, get off of my cloud: exploring information leakage in third-party compute clouds, Conference on Computer and Communications Security (CCS), 2009.

[30] Y. Sawa and R. Suda, Auto Tuning Method for Deciding Block Size Parameters in Dynamically Load-balanced BLAS, International Workshop on Automatic Performance Tuning (iWAPT), 2009. http://iwapt.org/2009/slides/Yuta_Sawa.pdf

[31] SLURM, http://slurm.schedmd.com/

[32] SoftLayer, http://www.softlayer.com/bare-metal-servers

[33] S. Soltesz, H. Potzl, M.E. Fiuczynski, A. Bavier, and L. Peterson, Container-based Operating System Virtualization: A Scalable, High0performance Alternative to Hypervisors, European Conference on Computer Systems (EuroSys), 2007.

[34] K. Suzaki, K. Iijima, T. Yagi, and C. Artho, Memory deduplication as a threat to the guest OS, European Workshop on System Security (EuroSec), 2011.

[35] K. Suzaki, K. Iijima, T. Yagi, and C. Artho, Implementation of a Memory Disclosure Attack on Memory Deduplication of Virtual Machines, IEICE TRANSACTIONS on Fundamentals of Electronics, Communications and Computer Sciences, Volume E96-A No.1, 2013.

[36] H. Tazaki, F. Uarbani, E. Mancini, M. Lacage, D. Camara, T. Turletti, and W. Dabbous, Direct code execution: revisiting library OS architecture for reproducible network experiments, Conference on Emerging networking experiments and technologies (CoNEXT), 2013

[37] P. Yuan, Y. Guo, and X. Chen, Rethinking Compiler Optimizations for the Linux Kernel: An Explorative Study, Asia-Pacific Workshop on System (APSys), 2015.