Received December 25, 2020, accepted January 6, 2021, date of publication January 18, 2021, date of current version February 5, 2021.

Digital Object Identifier 10.1109/ACCESS.2021.3052525

A BitTorrent Mechanism-Based Solution for Massive System Deployment

STEVEN J. H. SHIAU[1], YU-CHIANG HUANG[2], YU-CHIN TSAI[1], CHEN-KAI SUN[1], CHING-HSUAN YEN[2], AND CHI-YO HUANG[3]

[1]National Center for High-Performance Computing, National Applied Research Laboratories, Hsinchu 30076, Taiwan
[2]Department of Computer Science, National Chiao Tung University, Hsinchu 300, Taiwan
[3]Department of Industrial Education, National Taiwan Normal University, Taipei 106, Taiwan

Corresponding authors: Steven J. H. Shiau (steven@nchc.org.tw) and Chi-Yo Huang (cyhuang66@ntnu.edu.tw)

This work was supported in part by the Ministry of Science and Technology, Taiwan, under Grant MOST 108-2634-F-492-001, and in part by the National Center for High-Performance Computing in Taiwan.

ABSTRACT System deployment of a computer environment plays a critical role in the daily administration of computer systems, and tasks of massive deployments may take a lot of time for data center administrator(s). The existing solutions for massive deployment normally consist of storage spaces and extra computer servers for running deployment services. Existing solutions of multicast massive deployment are not robust because the overall deployment performance will worsen if one client machine fails. Because of the limitation of the network protocol, a multicast solution is not scalable. Although scholars have proposed solutions based on BitTorrent (BT) to overcome performance and scalability problems, solutions are not good enough because they still require the storage space to save the image file. In this paper, we present a novel mechanism of massive deployment called ''BT deployment mechanism from the raw device'' (BDMfRD), which differs from conventional solutions in that it avoids creating any image file in the deployment process or using external storage for it. The proposed solution was verified by conducting 10 experiments to replicate the 50 GB system of the source machine to 1–32 destination computers. Experimental results showed that the proposed method reduced the total time for deploying 32 computers by 45.289%. The implemented software is the first massive deployment solution that provides light, robust, efficient, and scalable capabilities simultaneously.

INDEX TERMS System deployment, bare-metal provisioning, massive deployment, free software, open source, cloning, imaging, peer-to-peer (P2P), BitTorrent (BT).

I. INTRODUCTION

The activity to enable the uses of an operating system (OS) and applications in computers is called system provisioning or system deployment [1]–[3]. As information is digitized, scientific computing and information education play increasingly important roles in the modern world, and system deployment becomes critical daily because the OS and applications are the bases for computer services. In computer terminology, creating a backup (or saving information) is the process of storing the contents of a disk or a partition to an image file [4]; meanwhile, restoring refers to writing an image to a disk or a partition. The processes of backing up or restoring files are called imaging whereas the process of duplicating the data from one machine's disk partition directly to another disk partition, without saving any image, is called cloning. Disk restoring or cloning can be done one to one or one to many in the local disks on the same computer as well as on multiple destination computers. When the process is executed on numerous destination computers, it is called massive system deployment, or massive deployment for short. In order to reduce system administration's maintenance efforts significantly, an efficient scheme for massive deployment is a must. As the number of machines to be deployed concurrently increases, the performance and functionality of massive deployment techniques have to be enhanced. Many researchers and developers (e.g., ROCKS and OpenGnSys [1], [5]) have tried to reduce the gap between users' needs and the current massive deployment solutions by using solutions that rely on the automatic installation, network booting, and network protocols (e.g., unicast, broadcast, multicast and peer-to-peer [P2P]). Some vendors also

The associate editor coordinating the review of this manuscript and approving it for publication was Zeev Zalevsky.

provide massive deployment solutions, such as Norton Ghost, Acronis True Image [3], Commercial Scale-Out (CSO) [6], the P2P deployment patent by IBM Corporation [7], and the method and apparatus for operating system deployment patented by Sun Microsystems, Inc. [8]. Among these commercially available solutions, the P2P mechanism being proposed in the patent of IBM [7] is described as follows: "All of the computer system will be sorted as different groups. ... For example, if the total computers are classified into three groups, the first group sends a wake-on-LAN (WoL) instruction and boots each of the second group in the beginning. The second group identifies a set of third group that corresponds to each of the selected second group. Following the same mechanism, the second group sends the WoL signal to each of the third group, and the third group is booted over the computer network from the second group." This approach is one of the P2P solutions, and apparently the solution proposed by IBM can provide good scalability for massive deployment. However, an image has to be stored in each level of computers belonging to the group. Thus, more disk space is required, meaning there is still room for improvement.

Some firms also provide different solutions for massive deployment by using hardware. One example is the recovery card [9] from the Taiwan-based company TOPOO Technology, which uses a peripheral component interconnect (PCI)-based interface card and embeds the data replication software in its read-only memory (ROM). Every computer to be deployed should be equipped with this peripheral card to receive the data blocks being forwarded by the server. However, this solution is not flexible as every update in the replication software has to be written to the erasable programmable read-only memory (EPROM), which is not easy for a system administrator. In addition, the expense is a burden as extra costs will be required to add a PCI card in every machine.

Generally speaking, existing solutions for massive deployment normally consist of storage spaces and extra computer servers for running deployment services. Existing solutions for multicast massive deployment are not robust because overall deployment performance will worsen if one client machine fails. Because of the limitation of the network protocol, the multicast solution is not scalable. Our previous studies [3], [10] have revealed that the network-based deployment system using unicast, broadcast, and multicast mechanisms have some drawbacks. Such systems cannot be enhanced to achieve good efficiency and scalability. The BT deployment mechanism from an image (BDMfaI) solution proposed in our previous study [10] was based on the BitTorrent (BT) protocol [11], a P2P [12] file-sharing technology developed approximately 20 years ago. The BDMfaI has provided a robust and scalable scheme for massive deployment. However, some improvements are still required so that the BDMfaI solution can finish massive deployment in a lighter and more efficient way, where lighter means the extra disk space required is reduced. The original BDMfaI solution requires extra storage spaces to store the image files. A lighter solution can resolve this problem. Meanwhile, the original system is inefficient as the disk of the source machine has to be stored as an image and then the image must be converted to the file system blocks transferring (FSBT) format [10]. These storing and converting actions are time consuming and make the BDMfaI solution an imperfect solution when the source machine's hard drive contains huge used file blocks by the installed OS and applications. For example, when the OS and applications have the data size of about 100 GB, the BDMfaI solution might have to spend 30 minutes saving the contents of a disk as an image and then another 30 minutes converting the image to the FSBT format required by the BDMfaI solution.

Nevertheless, the existing massive deployment solutions to be discussed in section II have some problems or limitations, which include the requirement of extra hardware resources, unrobustness, inefficiency, and unscalability. No existing solution(s) can address these issues altogether. To solve these problems and limitations, we propose a novel massive deployment solution that is lighter, more efficient, robust, and scalable. The proposed solution differs from conventional solutions by avoiding the creation of any image file in the deployment process or adopting external storages for the image file. In addition, for efficiency, robustness, and scalability, BT was chosen as the data communications protocol for massive deployment.

The focus in this study is to neglect the time required to save the disk of the source machine as an image and convert the image to the FSBT format. To greatly enhance the performance, the system architecture of available solutions [10] will be modified. Therefore, in Section 3, a novel solution called the BT deployment mechanism from the raw device (BDMfRD) is proposed. A raw device is a special kind of block device file that allows for accessing a storage device directly [13], bypassing the OS buffer. The proposed BDMfRD solution does not need to save the contents of disks of the source template machine as an image and convert these images as the FSBT format.

To achieve the goals of reading directly from and writing directly to the hard drives of the source machines, the open-source programs Partclone [14] and EZIO [15] were modified to provide the required functions that the BDMfRD solution needs. In addition to proposing the solution, the implementation and improvements of existing solutions [10] will also be discussed. To verify the feasibility and demonstrate the efficiency, the proposed BDMfRD solution was tested and verified in experiments by deploying the OS and applications from 1 up to 32 physical personal computers (PCs) in the same place. Based on the proposed BDMfRD solution, the total time of deploying 32 computers was reduced from 2526 seconds (secs) to 1382 secs when compared with the earlier BDMfaI [10] solution, resulting in a time reduction of 45.289%. The experimental results, comparisons, and discussions in this research demonstrate the feasibility, efficiency, and performance of the proposed solution. With the proposed method, a system administrator of cluster computing, for example, can put the software we developed in this research in a USB flash drive, boot the template machine with it, and dispatch its OS and applications in the internal storage device to other machines or clusters. There is no need to create any image file in the process or use any external storage for storing the file. To the best of the authors' knowledge, this is the first to provide a light, robust, efficient, and scalable solution for massive deployment.

The rest of this paper is organized as follows. Section 2 reviews and discusses the system deployment methods and network protocols. Section 3 defines the proposed system architecture for the massive deployment using the BDMfRD solution. The experimental results are described in Section 4. The differences and performance of the proposed BDMfRD solution in this study are compared with our previous BDMfaI solution, as well as other mass deployment programs in Section 5. The rationalities of the enhancements are also discussed. Section 6 concludes this study, and the suggestions for future research are put forward.

II. RELATED WORKS

This section reviews and discusses the related works for massive deployment, which include the system deployment methods and the related network protocols for system deployment.

A. SYSTEM DEPLOYMENT METHODS

The system deployment methods can be classified into three categories [3]: (1) differential update, or restoring previously saved files from a disk [16]; (2) automated installation, or installing OS and applications from scratch with an automated installation and configuration program [17]; and (3) file system imaging, or restoring an image of a previously saved file system [5]. Due to the efficiency in file synchronization, the network bandwidth consumption, and the requirement for a file system to be created prior to system deployment, the differential update method is not adequate for the bare-metal system deployment.

Many applications require massive deployment, such as high-performance computing (HPC) clusters, modern classrooms, and modern offices, as well as numerous computers that should be rapidly deployed to fulfill users' needs. In addition to these applications, cloud computing adopts emerging virtual technology (i.e., hypervisor and container) for rapid system deployment [3], [18]–[21]. Thus, the OSs and applications can be ready for people to use in a very efficient way and the systems can be ready to serve users in a short time. However, cloud computing relies on the OS and applications installed on the physical host machines. In other words, cloud computing cannot provide services without the OS(s) and applications on the host machine(s). Thus, system deployment on the physical machine is essential for cloud computing because massive physical machines are required to deliver cloud-based services when serving many users. Accordingly, system deployment is indispensable in the modern ages as computing power is required in many locations.

TABLE 1. Summary of data transmission protocols for massive deployment.

Protocol	Transmission Type	Bandwidth Consumption	Robustness	Efficiency	Scalability
Unicast	One to one	Average	Average	Worst	Worst
Broadcast	One to all	Least	Average	Average	Average
Multicast	One to many, many to many	Least	Average	Average	Average
P2P	Many to many	Most	Best	Best	Best

Given these identified requirements, many researchers have endeavored to develop massive deployment techniques by using an automatic installation technique or file system imaging scheme. The solutions using an automated installation technique include Heckle [22], xCat [23], Perceus [24], OSCAR [25], ROCKS [1], Chef [26], Puppet [27], Ansible [28], SaltStack [29], Kickstart Installation [1], Fully Automatic Installation (FAI) [30], and metal as a service (MAAS) [3] whereas resolutions using a file system imaging mechanism include Norton Ghost [31], Acronis True Image [32], Partimage, Partclone, Fsarchive, Redo Backup, OpenGnSys, FOG, MS Windows Deployment Services (WDS), Frisbee, and Emulab [3]. Although so many massive deployment tools have been studied and developed, these solutions all need an extra disk repository to store the image. Meanwhile, scalability issues exist due to two reasons: (1) the consumption of network bandwidth—that is, the available network bandwidth between the server and every unicast destination (client) machine—decreases linearly as the number of destination machines increases and (2) the deployment server's high system loading due to too many connections and requests from the client machines. A novel scheme for massive deployment is required to address these two issues, which include the extra disk repository requirement and scalability.

B. UNICAST, BROADCAST, MULTICAST, AND BITTORRENT PROTOCOLS FOR SYSTEM DEPLOYMENT

Many protocols for data transmission in computer networks have been proposed and existed for a long time. Typical examples include the unicast, broadcast, multicast, [33], and P2P networking [34]. Various researchers [3], [5], [10], [33]–[42] have attempted to implement the last three protocols (i.e., broadcast, multicast, and P2P networking) for massive deployments.

The characteristics of these four categories of protocols are summarized in Table 1. A review of the literature [10] indicated that the P2P solution, although consuming the most bandwidth. surpasses the remaining protocols due to its excellence in terms of reliability, efficiency, and scalability from the aspect of massive deployment.

As shown in Table 1, data transmissions between one sender and one receiver are the mechanism for unicast protocol. The broadcast protocol is designed for data transmissions between one sender and all receivers. Furthermore, the multicast protocol improves the efficiency of unicast by transmitting data from one or many senders to many receivers; thus, the multicast protocol has the ability to select the destination receivers instead of sending data to all receivers. Although the multicasting mechanism outpaces the traditional unicasting one in performance enhancement, the multicast protocol has a drawback in packet loss stemming from the user datagram protocol (UDP) [43]. Consequently, the scalability of the multicast-based deployment is very limited. The scalability and performance are acceptable when the destination machines are limited, yet the scalability/performance becomes worse as the number of destination machines increases. Moreover, a multicast-based solution has a limitation. When a client has a packet-receiving issue, the server will be requested to resend the packets. At this moment, other clients have to wait for the overall packet transmission to be completed, which will lower the overall performance of massive deployment. Therefore, the multicast mechanism is not suitable for large-scale deployments.

In order to overcome the weaknesses of the broadcast and multicast mechanisms in massive deployment, the P2P protocol was proposed; its sharing mechanism leverages a distributed network architecture, which enables the participants in the distributed network to serve as both providers and requestors [12]. One of the merits of P2P solutions is that they have the provisions of resources belonging to each peer, which include computing power, storage space, and network bandwidth. Consequently, once the peers obtain part of the data, they can start sharing the data, and the system's total data-sharing capacity also grows.

On the other hand, for the client–server architecture with a fixed set of servers, when more clients are added, the data transmission rates for all clients decreases. Accordingly, this distributed nature of P2P solutions also increases robustness in the case of failures by replicating data over multiple peers [7]. Therefore, the P2P solution is very suitable for data transmission between many-to-many nodes due to the best performance.

P2P sharing is an overall idea, and many protocols have been proposed for this purpose. Typical examples include eDonkey [44], Gnutella [45], FastTrack [46], and BT [47]. Of these, the BT protocol [47] developed in 2001 aims to cut the file to be shared into segments called pieces, where the file size generally varies from hundreds of kilobytes (KBs) to a few megabytes (MBs). Typically, the BT environment consists of three parts: (1) a torrent tracker, which keeps track of senders and receivers; (2) a torrent server, where the torrent metadata file is kept; and (3) peers, which are the instances on the network that can transfer data. The BT protocol has the mechanism to ensure the high availability of resources, meaning that peers can download pieces in a ''rarest-first'' [48] approach, which enables peers to seek the rarest piece available among the peers. Alternatively, peers download pieces in a random approach. Based on these subtle mechanisms of BT, the server, peers, and network can work in an efficient mode when sharing big files. Thus, the BT-based solution can be scalable. In addition, the proposed solution can leverage the advantages of BT while reducing energy consumption in the future. Because a large number of computers is involved in P2P networks, total energy consumption can be high. A good solution (e.g., Marozzo *et al.* [49]) can reduce overall energy consumption in P2P networks. The BitTorrentSW proposed by Marozzo *et al.* [49] can save energy while ensuring good file-sharing performance. This approach is a sleep-and-wake mechanism for BT networks, which allows seeders to switch cyclically between wake and sleep modes. The analytic results demonstrate that in a network with 50% seeders, compared with a standard BT-based solution in which all seeders are always turned on, BitTorrentSW can reduce energy consumption by 20% while increasing the average time required to complete a file download task by 7% only. In addition, in a network with 60% seeders, BitTorrentSW can reduce energy consumption by 28% while just increasing download time by 4%. The BitTorrentSW proposed by Marozzo *et al.* [49] demonstrates effectiveness in reducing energy consumption with limited effects on average download time.

Due to the features of the P2P protocol (i.e., its ability to distribute large files efficiently and be scaled up), researchers have studied related topics, and some typical examples are reviewed herein. For example, Tracey and Sreenan ()[50] proposed a P2P-based approach for fog computing and implemented a prototype to demonstrate how a Holistic Peer-to-Peer (HPP) architecture and application layer protocol can meet the requirements for the Internet of Things (IoT). Viml and Srivatsa [51] proposed a P2P-based Interplanetary File System (IPFS) to share resources or files in a distributed system. The IPFS aims to increase the efficiency of the P2P file-sharing system by using the blockchain. To motivate the miners, who have computing power and verify the transaction on a blockchain, for successful transaction, an IPFS-based incentive solution such as the filecoin has been proposed. The services provided during the file transfer process and the security strength and some of the incentives based on IPFS are discussed in the work by Viml and Srivatsa [51]. Ali *et al.* [52] proposed a blockchain-based decentralized P2P remote health monitoring system, built on the Ethereum blockchain, which serves as a medium for negotiating and record-keeping, along with the onion router (TOR) for delivering data from patients to doctors. The work by Ali *et al.* [52] enables patients to share their biomedical data with doctors without the data being handled by trusted third-party entities.

As for the system deployment based on the BT protocol, Jeanvoine *et al.* [40] developed a BT-based massive deployment solution called Kadeploy3, which has been widely adopted on the Grid'5000 test-bed since the end of 2009. The solution has been deployed more than 117,000 times by 620 different users, where the largest deployment involved 496 nodes [40]. Xue *et al.* [36] proposed a BT-based solution, the Efficiency, Scalability, Independence, and Reliability (ESIR), for image transference in mass deployments. The ESIR has demonstrated higher image distribution performance than multicast solutions. Qadeer *et al.* [53] proposed a flexible framework to automate the process of bringing up an infrastructure for the deployment of several OpenStack distributions as well as resolving dependencies for a successful deployment. To reduce the bare-metal provisioning time, Shestakov and Arefiev [54], [55] introduced a BT solution to the original OpenStack Ironic project. The computing node to be deployed is able to download the image from both peers and the object storage as well as share the downloaded data with other peers. Only 15 minutes were required to deploy a 3 GB image to 90 nodes by the work proposed by Shestakov and Arefiev [54], [55]. In comparison with the standard OpenStack Ironic, which took 1 hour to provide 15 nodes for the same sized image, the performance of Shestakov and Arefiev's work is apparently much better. Although the number of nodes to be deployed is six times greater, the work by Shestakov and Arefiev took only 25% of the deployment time compared to the standard OpenStack solution. However, Shestakov and Arefiev's patch files have not been merged to the OpenStack repository yet due to some concerns [54].

Although scholars have proposed various BT-based solutions, earlier works related to the BT-based system deployment solutions have mainly concentrated on the efficiency, reliability, and scalability of the systems. All BT-based solutions reviewed thus far in this section have implicitly presumed that the image size of the OS and applications to be deployed is smaller than the available random access memory (RAM) size of the designated machine to be deployed. This assumption is not universally applicable for all scenarios because there is always some machine that does not have sufficient RAM to store the image consisting of the OSs and all the required applications. Consequently, our previous work [10] has addressed the issue related to the shortage of temporary storage by proposing a FSBT mechanism. Our work [10] demonstrated the feasibility and efficiency that the FSBT mechanism can solve the shortage problem of temporary storage in the destination machine(s). The robustness of our previous BT and FSBT work is due to the data distribution to multiple peers in the BT mechanism. Thus, the BDMfaI can reduce the possibility of failure that some peers cannot receive the data, which is a drawback in server–client architecture. Furthermore, the BDMfaI solution was conducted with the used blocks data to be deployed from the source machine larger than the size of the destination machine's RAM. In addition, because the BDMfaI solution makes all the destination machines to be deployed and the source machine use BT protocol to transmit the used blocks data, it has excellent scalability, as demonstrated in our previous study [10].

Although the FSBT mechanism can solve the problem related to the shortage of temporary storage, it creates some additional issues—namely, the BT server has to prepare an extra storage space to keep the image and requires extra time to convert the image being saved from Partclone to the FSBT format [10]. These two requirements are the major drawbacks when the hard disk on the source machine contains the OS and many applications where the size of used blocks to be deployed could be more than 100 GB. A few hours may be required to save the contents of the source disk as an image and then convert the image to the FSBT format. Thus, a mechanism must be designed and implemented to ease this pain and make the BT-based solution more efficient.

Based on the reviews in this section, existing massive deployment solutions have some problems or limitations, which include the requirement of additional hardware resources, unrobustness, inefficiency, and unscalability. No existing solution can address these issues simultaneously. Consequently, this study aims to propose a novel scheme, the BDMfRD, for massive system deployment.

III. DESIGN AND IMPLEMENTATION

In our previous study [10], we proposed the BDMfaI scheme that can apply the BT protocol on massive deployment and verified it as feasible. However, as mentioned in Section II, some drawbacks still need to be improved. Among these drawbacks, one of the main concerns is the requirement to save an image from the hard drive of the source machine, and then the image is converted to the files in FSBT format and used for the BT seeder to replicate the data to other machines. This approach requires some time to save an image from the source machine and then convert the image, and the image needs storage space to be preserved. If these two requirements for the image and storage space can be removed, the efficiency of the massive deployment based on the BT mechanism would be increased. Therefore, we proposed the BDMfRD solution to address these two issues in this work. This novel BDMfRD scheme contains four features: (1) lightness; (2) good efficiency; (3) robustness; and (4) good scalability. The software developed in this study, the upgraded Clonezilla live [3], has a lightness feature because it does not need to prepare these three extra resources: (1) no need for storage space to store the image files; (2) no need to prepare an extra server because the source template machine can be booted by Clonezilla live as the server; and (3) no need to insert an extra PCI interface card in each machine for sending and receiving data. The proposed BDMfRD solution has greater efficiency because the image is not required to be stored in the storage and then converted to the FSBT format. Instead, the proposed solution directly reads and writes the used blocks from the disk device. Compared with other BT-based solutions, the BDMfRD architecture proposed in this research can be adopted to deploy the used blocks data to be deployed from the disk of the source machine, which is larger than the RAM size of the destination machine. In addition, for the proposed BDMfRD solution, the data are replicated to multiple peers so the sources are diverse. Consequently, the proposed solution reduces the risk of single point failure, which has commonly been seen in server–client architectures. Hence, the proposed solution is robust. Because all the destination machines to be deployed and the source machine use the BT protocol to transmit the used blocks data, excellent scalability can be achieved, as discussed in Part B of Section II. The feasibility of the proposed solution will be demonstrated in Section IV by deploying

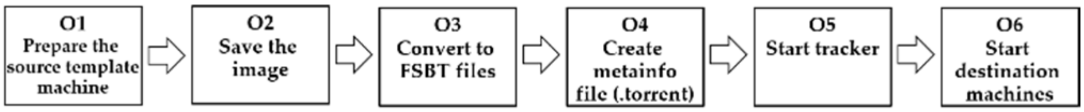

O1
Prepare the source template machine
O2
Save the image
O3
Convert to FSBT files
O4
Create metainfo file (.torrent)
O5
Start tracker
O6
Start destination machines

FIGURE 1. The flowchart for the BDMfaI solution.

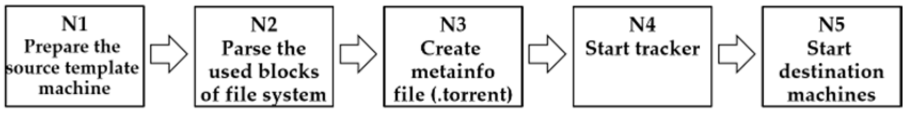

N1
Prepare the source template machine
N2
Parse the used blocks of file system
N3
Create metainfo file (.torrent)
N4
Start tracker
N5
Start destination machines

FIGURE 2. The flowchart for the BDMfRD solution.

a GNU/Linux system from a source template machine to numerous destination machines in a computer classroom.

The flowchart of the original BDMfaI solution is depicted in Figure 1 while Figure 2 presents the flowchart for the newly proposed BDMfRD solution in this study. For the BDMfaI solution, the steps are:

O1: prepare the source template machine (i.e., install the OS and applications).

O2: save the image from the source template machine's hard drive.

O3: convert the image to the FSBT format [10].

O4: create the BT metainfo files (∗.torrent) of the FSBT files.

O5: start the tracker for the BT mechanism and the network booting service.

O6: boot destination machines via network booting and then start the massive deployment via the BT mechanism.

For the BDMfRD solution, the steps can be reduced as follows:

N1: prepare the source template machine (i.e., install the OS and applications).

N2: parse the used blocks of the file system on the source template machine's hard drive.

N3: create the BT metainfo files (∗.torrent) based on the results of step N2.

N4: start the tracker for the BT mechanism and the network booting service.

N5: boot the destination machines via network booting and then start the massive deployment via the BT mechanism.

These two procedures are illustrated in Figures 1 and 2. As shown in the two illustrations, Figure 2 has fewer processes after the source template machine is ready for massive deployment. Although there is such a merit, the BDMfRD solution still has some limitations, as we will discuss in Section V.

To combine procedures O2 and O3 from the BDMfaI solution into a single step (N2), a new mechanism was proposed in BDMfRD. The basic idea is that the data already existed on the hard drive, so there is no need to save the used blocks of the file system on the hard drive as an image and then convert the image to the files in the FSBT format. Apparently, when the BDMfaI solution is implemented, the same data will exist in three different formats: (1) the original used blocks on the hard drive; (2) the image files; and (3) the files in the FSBT format. Therefore, the BDMfaI solution is not efficient because it requires two more storage spaces for the same data but in different formats. Therefore, by introducing the BDMfRD, the image file and the files in the FSBT format are unnecessary. Although we do not have to save the image and convert it to the FSBT format, the used blocks on the file system still have to be parsed by the Partclone [14] so that a mapping file and the metainfo file for the BT mechanism can be created. In addition, the EZIO program [15], a BT-based deployment program, has to be improved to read and write the used blocks directly from and to a raw device. The details of implementing Partclone and EZIO to meet these requirements are described in detail in section A of the Appendix.

Based on the mechanisms mentioned in Appendix A, our earlier works on massive deployment [3], [10] were enhanced so that the system could be deployed directly from a source template machine without any external storage space and an extra server machine. The template machine can be adopted directly as both the source machine and massive deployment server. Figure 3 demonstrates the services on the source template machine and the corresponding functions required by the destination machines during system deployments. Once the source template machine is ready, it is booted by Clonezilla live and enters the "Lite Server Mode" [56]. The services started on the source template machine, including the dynamic host configuration protocol (DHCP), trivial file transfer protocol (TFTP), hypertext transfer protocol (HTTP), EZIO, and BT tracker, are similar to those in our previous work [3], [10]. However, the major difference between this study and the previous one is step D3 for BT deployment—that is, steps O2 and O3 were merged into step N2, as previously mentioned in this section.

Figure 4 presents an example of adopting the mechanism proposed in this study to replicate the OS and applications reading directly from the source template machine's hard drive to three other machines. The major difference between this mechanism and the one proposed in our earlier work [10] is the lack of an image repository requirement. Users can boot the source template machine with Clonezilla live, enter the Lite Server Mode, assign the mode as "BT from the raw device," set the source template machine's hard drive as the source, and then boot the three destination machines via network booting. The remaining jobs, including all deployment tasks and system rebooting, will be finished automatically. The used blocks on the hard drive from the source template machine will be replicated to the three destination machines

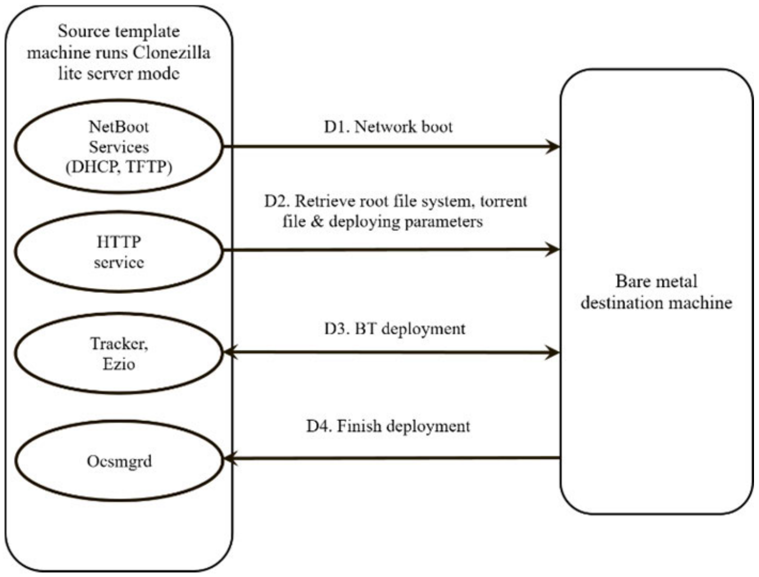

Source template machine runs Clonezilla lite server mode
NetBoot Services (DHCP, TFTP)
HTTP service
Tracker, Ezio
Ocsmgrd
D1. Network boot
D2. Retrieve root file system, torrent file & deploying parameters
D3. BT deployment
D4. Finish deployment
Bare metal destination machine

FIGURE 3. The services provided by Clonezilla lite server in the source template machine and their corresponding functions in system deployment.

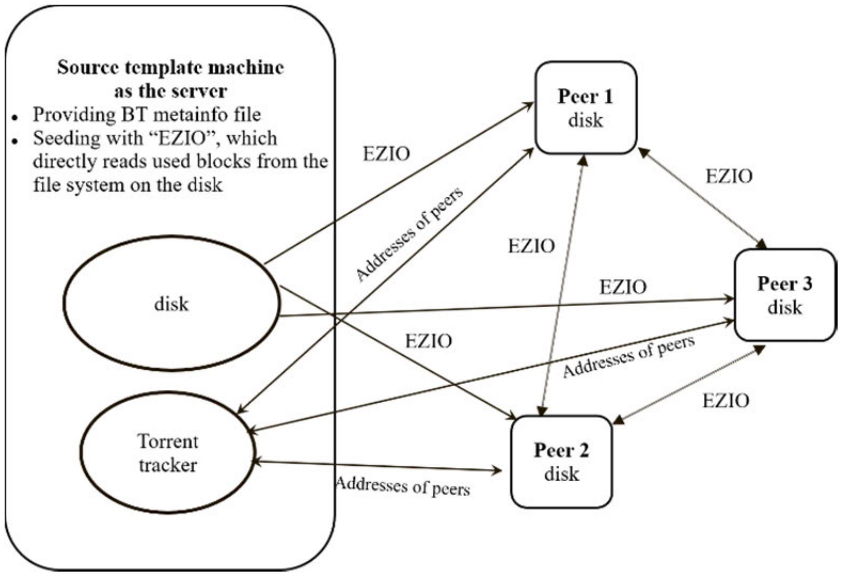

Source template machine as the server
• Providing BT metainfo file
• Seeding with "EZIO", which directly reads used blocks from the file system on the disk
disk
Torrent tracker
Peer 1 disk
Peer 2 disk
Peer 3 disk
EZIO
Addresses of peers

FIGURE 4. The schematic figure for massive deployment. The template machine runs as the source and deployment server, and the three destination machines are deployed.

using the BT protocol by EZIO. In Figure 4, EZIO reads only used blocks from the source template machine (seeder), while both reading and writing the used blocks from and to the hard drives of these three destination machines (leechers).

Based on the mechanism discussed in the section, the software we have implemented has been released as an open-source program for download [56]. Furthermore, a step-by-step document has been released on the website [57].

IV. EXPERIMENTAL PROCESS AND RESULTS

This section presents the experimental process and results of the proposed mechanism for massive deployment. The conditions of experiments are presented in sub-section A. Then, sub-section B demonstrates the process of system preparations and massive deployments. Finally, sub-section C demonstrates the experimental results and compares the results being derived by various deployment methods.

A. CONDITION OF THE EXPERIMENTS

The experiments deployed a Linux system to 1 to 32 destination machines using the newly developed BDMfRD solution in a computer classroom. The experimental environment consisted of the following machines and images:

- The Cisco Catalyst 3560G switch with 48-gigabit ports, was adopted as the network switch. The spanning tree protocol was disabled to avoid the timeout of network booting in the destination machines.
- A Dell T1700 source template machine was adopted, which also served as the server. The central processing unit (CPU) is a 3.3 GHz Intel Xeon E3-1226 processor.

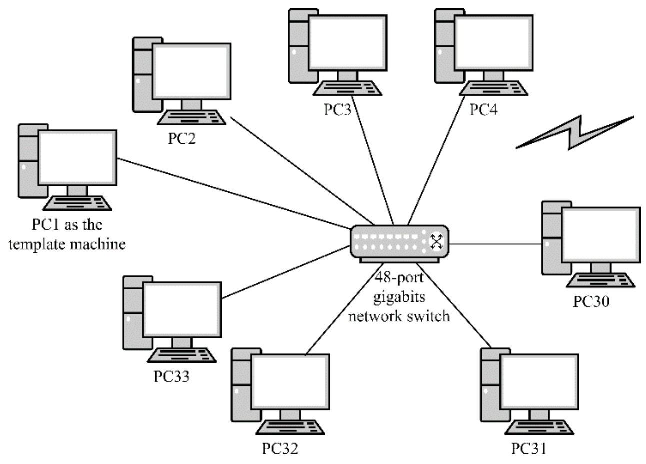


FIGURE 5. Configuration of the experiments. One of the PCs in the computer classroom was chosen as the source template machine. The rest of the 32 computers were connected with a 48-port gigabits network switch.

The size of the RAM is 16 GB. The size of the hard disk is 1 TB. An Ubuntu Linux system with applications and data was installed on this template PC, occupying 50 GB of the hard disk.

- For PC destination machines, Dell T1700 PCs with the same configuration as the one serving as the server were adopted.

In the experiments, 32 PC destination machines were connected to the server using a Cisco Catalyst 3560G switch and the Category 5e network cables. The configuration of the experiments for massive deployments in large-scale computers is presented in Figure 5.

When conducting the experiments, the source template machine was booted with Clonezilla live and entered the lite server mode, which has the function to relay the DHCP requests from the PC destination machines to the existing DHCP service in the LAN. After choosing the hard disk (/dev/sda, i.e., the first disk in the source template machine) as the source, the BT solution was chosen for the massive deployment. The destination machines were booted from the pre-boot execution environment (PXE) [58]. Ten experiments were conducted to replicate the used blocks of the hard drive on the source template machine to 1 to 32 destination machines. During the deployment to some of the destination machines, the remaining destination machines from the 32 machines not deployed were powered off.

B. SYSTEM PREPARATION AND MASSIVE DEPLOYMENTS

When each massive deployment experiment is conducted, the "total time" t is defined as the overall time required to deploy the file system(s). The total time is equal to the summation of the preparation time t_p and the massive deployment time t_d (refer Eq. (1)). Here, the preparation time t_p includes the image saving time and metainfo file creating time

$$t = t_p + t_d \tag{1}$$

TABLE 2. The time, t_p (secs), required for massive deployment preparation by different solutions.

Job type	$t_{SQp}{}^{(1)}$	$t_{MCp}{}^{(2)}$	$t_{BT1p}{}^{(3)}$	$t_{BT2p}{}^{(4)}$
IS(5)	551	551	551	0
CFF(6)	0	0	832	0
CMFUB(7)	0	0	0	291
Subtotal	551	551	1383	291

Remarks: (1) The time required by the sequential solution; (2) the time required by the multicast solution; (3) the time required by the BDMfal solution; (4) the total time required by the BDMfRD solution; (5) IS is the abbreviation for image saving; (6) CFF is the abbreviation for FSBT and metainfo file creations; and (7) CMFUB stands for the creation of metainfo files

TABLE 3. The time, t_d (secs), required to deploy a 50 GB Ubuntu Linux system and applications using the sequential, multicast, BDMfaI, and BDMfRD solutions.

Number of Machine(s)	$t_{SQd}{}^{(1)}$	$t_{MCd}{}^{(2)}$	$t_{BT1d}{}^{(3)}$	$t_{BT2d}{}^{(4)}$
1	474	390	675	483
2	948	474	1273	834
4	1896	638	1331	882
8	3792	980	1412	899
12	5688	1356	1272	936
16	7584	1454	1005	922
20	9480	1660	1000	1008
24	11376	1992	1048	929
28	13272	2131	1067	1074
32	15168	2203	1143	1091

Remarks: (1) The time required by the sequential solution; (2) the time required by the multicast solution; (3) the time required by the BDMfal solution; and (4) the total time required by the BDMfRD solution.

The remaining time required for system booting, partition table creation, image file conversion, and informing the server regarding deployment results is not included.

Table 2 summarizes the preparation time required for the jobs of the four types of solutions for massive deployments (i.e., the sequential [unicast], multicast, BDMfaI and BDMfRD). No matter what the solution is, basically, three types of jobs might be related to the four identified solutions of massive deployments: (1) image saving; (2) FSBT file creations; and (3) the creation of metainfo file. It should be noted that these three types of job are not all required by the four solutions. For example, the "image saving" is required by the sequential and multicast solutions. Therefore, the preparation time for both the sequential (t_{SQp}) and multicast (t_{MCp}) solutions is 551 seconds. In addition, both the "image saving" and "FSBT/metainfo file creation" jobs are required by the BDMfaI solution only. The time (t_{BT1p}) required for the preparation is 1383 secs. For the BDMfRD solution, the time is only required for creating metainfo file. The required time for preparation (t_{BT2p}) is 291 secs.

Table 3 summarizes the time t_d, which is required to deploy a 50-GB Ubuntu Linux system and applications to 1 to 32 destination machines using the sequential, multicast, BDMfaI and BDMfRD solutions. The time required for these four solutions is obtained from the experiments conducted for this study.

TABLE 4. The total time, *t* (sec), required to deploy a 50-GB Ubuntu Linux system and applications using the sequential, multicast, BDMfaI, and BDMfRD solutions.

Number of Machines(s)	t_{SQ} (1)	t_{MC} (2)	t_{BT1} (3)	t_{BT2} (4)
1	1025	941	2058	774
2	1499	1025	2656	1125
4	2447	1189	2714	1173
8	4343	1531	2795	1190
12	6239	1907	2655	1227
16	8135	2005	2388	1213
20	10031	2211	2383	1299
24	11927	2543	2431	1220
28	13823	2682	2450	1365
32	15719	2754	2526	1382

Remarks: (1) The total time required by the sequential solution (secs); (2) the total time required by the multicast solution (secs); (3) the total time required by the BDMfal solution (secs); and (4) the total time required by the BDMfRD solution (secs).

TABLE 5. The average time, t_a (sec), required to deploy a 50 GB Ubuntu Linux system and applications to a destination machine using the sequential, multicast, BDMfaI, and BDMfRD solutions.

Number of Machines(s)	t_{SQa} (1)	t_{MCa} (2)	t_{BT1a} (3)	t_{BT2a} (4)
1	1025	941	2058	774
2	750	513	1328	563
4	612	297	679	293
8	543	191	349	149
12	520	159	221	102
16	508	125	149	76
20	502	111	119	65
24	497	106	101	51
28	494	96	88	49
32	491	86	79	43

Remarks: (1) The average time required by the sequential solution (secs); (2) the average time required by the multicast solution (secs); (3) the average time required by the BDMfal solution (secs); and (4) the average time required by the BDMfRD solution (secs).

C. EXPERIMENTAL RESULTS AND THE COMPARISONS BETWEEN DIFFERENT DEPLOYMENT METHODS

Tables 4 and 5 summarize the total and average time required to deploy the machines using these four solutions, respectively. The values in Table 4 were obtained by aggregating the values from Table 2 and Table 3 using Eq. 1. The total time required to deploy a 50-GB Ubuntu Linux system and applications using the sequential is

$$t_{SQ} = t_{SQp} + t_{SQd} \tag{2}$$

while the time required for the BDMfRD solution is

$$t_{BT2} = t_{BT2p} + t_{BT2d} \tag{3}$$

According the results in Table 4, the BDMfRD solution clearly outperformed the other three solutions in all the deployments to different numbers of destination machines because the time required for the BDMfRD solution to deploy all the files is the shortest, except when two destination machines were deployed. For such a case, the multicast solution (1025 seconds) performed slightly better than the BDMfRD (1125 seconds).

Table 5 summarizes the average time required to deploy a destination machine using the four previously mentioned solutions. The values were obtained from the total time to deploy the whole system divided by the number of destination machines. Basically, the trend is the same as the trend observed in Table 4. When 32 destination machines were deployed (see Table 5), the BDMfRD solution only spent 8.758% of the average time required by the sequential solution, 50% of the average time required by the multicast solution, and 54.430% of the average time required by the BDMfaI solution.

Figures 6 and 7 demonstrate the results of the massive deployments using the multicast, BDMfaI, and BDMfRD solutions. The total time and average time required by these three massive deployment solutions are demonstrated in these two figures. As the sequential solution is not adequate for this kind of massive deployment, the results of

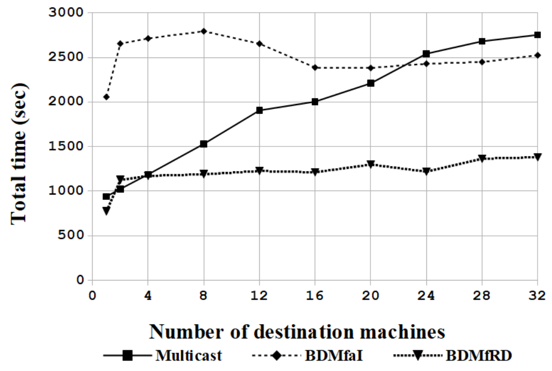


FIGURE 6. The total time required for the massive deployment to various destination machines using the multicast, BDMfaI, and BDMfRD solutions.

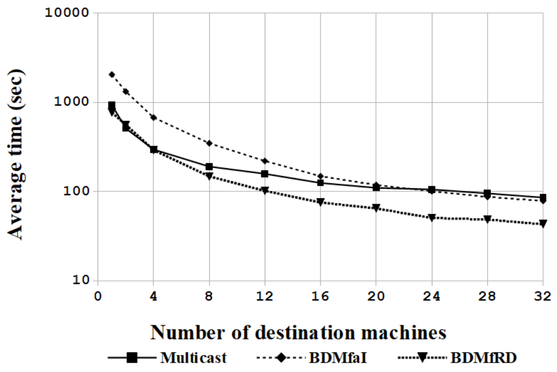


FIGURE 7. The average time required for the massive deployment to various destination machines using the multicast, BDMfaI, and BDMfRD solutions.

deployment when using the sequential solution is not presented in Figures 6 and 7.

Based on the illustrations in Table 4 and Figure 6, the total time required for massive deployment by the proposed BDMfRD solution is lowest compared with the total time required by the multicast and BDMfaI solutions, except when deploying two destination machines. This case is discussed in Section V. In addition, the average time required for

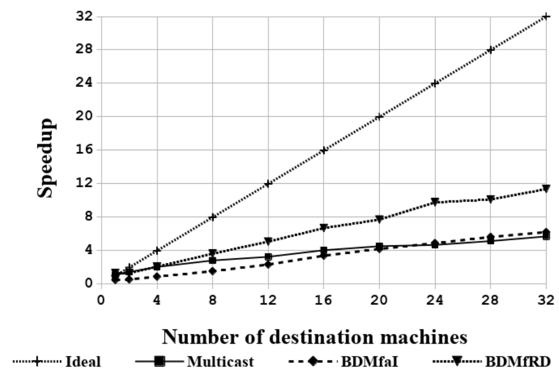


FIGURE 8. The speedup for the massive deployment to various destination machines using the multicast, BDMfaI, and BDMfRD solutions. The ideal speedup is also shown for comparisons.

TABLE 6. Speedups and ratios for BDMfRD to BDMfaI solutions.

Number of Machines(s)	SF_{MC} (1)	SF_{BT1} (2)	SF_{BT2} (3)	SR (4)
1	1.089	0.498	1.324	2.659
2	1.462	0.564	1.332	2.361
4	2.058	0.902	2.086	2.314
8	2.837	1.554	3.650	2.349
12	3.272	2.350	5.085	2.164
16	4.057	3.407	6.707	1.969
20	4.537	4.209	7.722	1.834
24	4.690	4.906	9.776	1.993
28	5.154	5.642	10.127	1.795
32	5.708	6.223	11.374	1.828

deploying a destination machine using the BDMfRD solution decreases as the number of destination machines increases. This phenomenon reveals that the proposed BDMfRD solution has a good scalability in the massive deployment. All of the destination machines deployed by these three massive deployment solutions were verified as bootable and could enter the Ubuntu Linux after being deployed. Therefore, these experiments verified the feasibility, efficiency, and scalability of the proposed BDMfRD solution.

V. DISCUSSION

This section discusses the performance achieved by the proposed BDMfRD solution. The comparisons between the proposed method and other solutions are also discussed. Finally, the limitations of this study are presented.

A. PERFORMANCE OF THE PROPOSED SOLUTION

Based on our earlier definition [3], the speedup of the massive deployment is the ratio of time required to deploy numerous computers sequentially versus the time required to deploy numerous computers through a massive deployment solution. Here, the time is the "total time" required, which includes both preparation time and massive deployment time. Figure 8 and Table 6 demonstrate the speedup being achieved in the experiments incorporating 1 to 32 machines. The ideal speedup case is provided for comparison. In the ideal case, no overhead exists. Meanwhile, the speedup increases linearly as the number of destination machines increases, which is provided for comparison.

TABLE 7. The time-saving ratio by the proposed BDMfRD solution compared to the previous BDMfaI solution.

Number of Machines(s)	Time-saving ratio (%) $\lvert t_{BT2}-t_{BT1}\rvert/t_{BT1}$
1	62.391
2	57.643
4	56.780
8	57.424
12	53.785
16	49.204
20	45.489
24	49.815
28	44.286
32	45.289

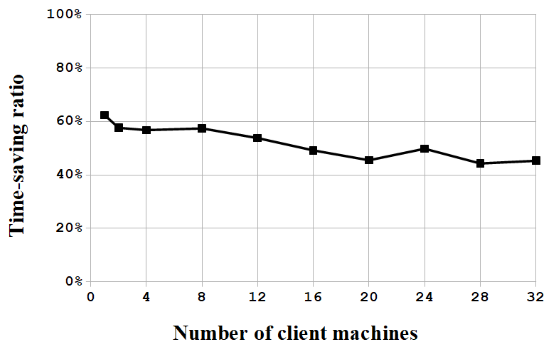


FIGURE 9. The time-saving ratio achieved by the massive deployment solution BDMfRD when compared with the BDMfaI solution.

The results from the real-world experiments show that both multicast and BT solutions achieved limited enhancements of performance compared to the ideal case. Among the multicast, BDMfaI, and BDMfRD solutions, the proposed BDMfRD achieved the best performance from the aspect of massive deployment. The speedup value of the BDMfRD solution is 11.374 when 32 destination machines were deployed, while the values of the multicast and the BDMfaI solutions are 5.708 and 6.223, respectively. The speedup ratio of the BDMfRD versus the BDMfaI solution is 2.659 when there is 1 destination machine. The ratio decreases slightly to 1.828 when 32 destination machines were deployed.

In general, the proposed BDMfRD solution is about two times faster than the BDMfaI one based on the speedup ratios. The time-saving ratios are summarized in Table 7 and demonstrated in Figure 9. The values of the time-saving ratios were derived from the differences between the time required by the BDMfaI and the BDMfRD solutions for the same number of destination machines. The difference between the time required by the two solutions was then divided by the time derived from the BDMfaI solution.

Based on the analytic results, the proposed BDMfRD solution improved 62.391% when one destination machine was deployed. The time-saving ratio was kept as 45.289% when 32 destination machines were deployed. From Tables 2 and 3, the preparation time for the BDMfaI solution is 1383 secs while the time required by the BDMfRD solution is only 291 secs. The preparation time for the BDMfaI is 4.753 times that of the BDMfRD, and it is almost about the same order of time compared with the BDMfaI deployment time for 32 destination machines. This overhead takes too much for the BDMfaI solution; thus, its performance is not as good as that of the BDMfRD solution. As mentioned in the previous section, the time required by the BDMfRD solution is lowest except when two destination machines were deployed, when the multicast solution showed better performance. Although the preparation time for the BDMfRD solution is only 291 secs, which is less than the preparation time for the multicast solution (i.e., 551 secs), during deployment, the BT solution adopted for deploying a small number of destination machines will be choked in the source machine, which is the only source of the data. As more destination machines join, the downloading rate of BT can be achieved when all peers increase [59]. When the number of destination machines is very small, like the 2-destination-machine case in this work, the strength of the BT-based solution cannot be demonstrated. Therefore, the multicast solution performed better in such a scenario.

The deployment time between the BDMfaI and BDMfRD solutions can be compared in Table 3. Based on the comparison results, the deployment time by the BDMfRD solution is smaller than that of the BDMfaI if the preparation time is neglected, no matter what the number of destination machines is. When the BDMfRD solution is adopted, the used block data on the file system to be transferred is read directly from the disks of all destination machines joining in the BT mechanism. Instead, for the BDMfaI solution, the source data is from the FSBT files located in one of the directories on the source template machine (server). Hence, the OS has to spend some time first locating the FSBT files and then sending the files to the destination machine(s) via the BT mechanism. Thus, the BT-based mechanism in the BDMfaI solution is not as efficient as the BDMfRD solution as the EZIO program can directly read the used blocks from the disk of the source template machine in the BDMfRD solution. However, when more destination machines join the BT-based mechanism in the BDMfaI solution, the FSBT files exist only on the server while the rest of the peers will also use the EZIO program to read and write the block data directly from and to the disks on the destination machines. Therefore, as the number of destination machines increases, the impact on the required time to locate the FSBT files on the server will become less when the BDMfaI solution is adopted.

From Table 3 and Figure 6, the scalability of the BDMfRD solution can be demonstrated. When the number of destination machines increases from 8 to 32, the total time to deploy all destination machines increases from 899 to 1091 secs, which is about a 21.357% performance enhancement at the moment when the number of destination machines increases by 400%. Based on the results derived in this study, this trend cannot be guaranteed when the number of destination machines is more than 32. However, based on the nature of BT [47] and earlier works [60], the BT protocol can work well when more than 32 peers exist.

TABLE 8. Comparisons of the BDMfRD and BDMfaI solutions.

Solution	Pros	Cons
BDMfaI	More flexible	Extra disk space required for image repository Lower performance Less scalable
BDMfRD	No extra disk space is required Better performance Better scalability	Less flexible

TABLE 9. Summary of comparisons of characteristics of the BDMfRD and BDMfaI solutions.

Solution	Feasibility	Time Reduction	Realibility	Scalability
BDMfaI	Yes	None	Good	Average
BDMfRD	Yes	52.211%	Good	Good

B. COMPARISONS

Our previous study [10] identified the pros and cons between BT and multicast mechanisms. In this study, although the same BT protocol is applied, the BDMfaI and BDMfRD solutions can still be differentiated. The pros and cons should be discussed, as summarized in Table 8. According to the usage scenarios, the BDMfaI solution showed better flexibility because the user can choose different images to deploy the destination machines. The BDMfRD solution has the pros that no extra disk space is required and it can provide better performance and scalability.

Table 9 compares the work and the previous solutions for BT-based massive bare-metal provisioning. From the aspect of efficiency, the BDMfRD solution has better efficiency than the BDMfaI solution based on the experimental results demonstrated in Section 4 and Table 7. The average time-saving ratio for 10 experiments is 52.211% (see Table 7); thus, the time reduction due to the BDMfRD solution is 52.211% for these 10 experiments when compared with the BDMfaI solution. Considering the number of destination machines, the time reduction can be sustained when the number of destination machines to be deployed is more than 32.

Table 10 compares Clonezilla live and other massive deployment solutions, including open source and proprietary ones. Clonezilla live, which has been improved by using the BDMfRD solution in this work, was released under the open source, free software GNU General Public License (GPL) [61], while Kadeploy [40] is offered under another open source, free software license CEA CNRS INRIA Logiciel Libre (CeCILL) [62]. The remaining programs, including SmartDeploy [63], Acronis Snap Deploy [32], Microsoft Deployment Toolkit [64] and EZ-Back System [65], are all proprietary software. As demonstrated in Table 10, Clonezilla live is superior to other solutions as it supports more features, such as allowing the source to be from an image or a raw

TABLE 10. Comparisons of massive deployment solutions.

Program	Software License	Case(*)				Notes
		1	2	3	4	
Clonezilla live	GPL	Yes	Yes	Yes	Yes	Open source, supports most of the mainstream Oss' deployment.
Kadeploy	CeCILL	Yes	No	Yes	Yes	Open source, provides a set of tools for cloning, configuring (post installation), and managing cluster nodes.
SmartDeploy	Proprietary	Yes	No	No	No	Supports MS Windows deployment only
Acronis Snap Deploy	Proprietary	Yes	No	Yes	No	Supports multicast solution, not BT.
Microsoft Deployment Toolkit	Proprietary	Yes	No	No	No	Supports MS Windows deployment only.
EZ-Back System	Proprietary	Yes	Yes	Yes	No	GNU/Linux deployment is done sector by sector

Remarks: * 1. support source from an image; 2. support source from a raw device; 3. supports both MS Windows and GNU/Linux deployment; and 4. supports the BT solution.

device, having the capability to deploy both MS Windows and GNU/Linux systems, and being able to adopt the BT protocol to replicate the data, resulting in better reliability and scalability than other solutions based only on the unicast or multicast protocols. In light of these factors, the improved Clonezilla live with the newly added BDMfRD solution adopting the BT protocol in this research once again demonstrated superiority for massive deployments.

C. LIMITATIONS

Despite the successful implementation in this study, the proposed BDMfRD solution still has some limitations, including no option for choosing the source image and a fixed device name. Overall, the BDMfRD solution lacks flexibility. These limitations are discussed next.

1) NO OPTION TO CHOOSE THE SOURCE IMAGE

When using the BDMfRD solution, there is no need to save the image of the disk because the source data will be read directly from the template machine's raw device. Hence, only one source is available for deployments. However, when the BDMfaI solution is adopted, the user can choose the image to be deployed from many of the images in the repository. Moreover, the image file is portable; the file can easily be duplicated and transferred to different places. However, raw devices, like the hard drive in the BDMfRD solution, cannot be duplicated and transferred in the same way to different places because the hard disk is a physical item. Hence, the circulation of the hard disk is not as easy as the digitalized image file.

However, the BDMfRD and BDMfaI solutions do not conflict with each other. One can put the image on the storage server as the repository and then use the Clonezilla lite server to mount the image repository to perform massive deployment when using the BDMfaI solution. If the template machine is ready to serve as the source machine, where the OS and applications have been installed, one can boot the source template machine as the Clonezilla lite server. The system can then be massively deployed to other destination machines using the proposed BDMfRD solution. Hence, users still have the option to choose either the BDMfRD or BDMfaI solution.

2) DEVICE NAME

When the BDMfRD solution is applied, the used block data on the file system has to be read and written directly from and to the physical raw device. If the BDMfaI solution is adopted, users can choose the image from the repository. If the destination device is different from the source device due to the differences in the type of hard drives (e.g., the source device is /dev/sda while the destination is /dev/nvme0n1), then Clonezilla can convert the image saved from the /dev/sda to the /dev/nvme0n1 format. However, as previously mentioned, Clonezilla has no way to convert the raw device name if the BDMfRD solution is applied, as the data have to be directly read and written from the physical raw device. Hence, the BDMfRD solution can only be used for the same type of hard drives between the source and the destination machines.

VI. CONCLUSION

This work proposed a novel mechanism of massive deployment, BDMfRD, based on the BT protocol. Previous works have been improved by making the Partclone program scan the file system on the hard drive and then list the used blocks so that the EZIO program can directly read and write the used blocks on the hard drive. This BDMfRD mechanism allows the OS and applications of the source template machine to be massively replicated directly from the hard drive of the source template machine to other destination computers without the preparation and usage of an image file. In addition, the BT-based mechanism proposed is light, more robust, efficient, and scalable. The BDMfRD solution is light because no storage space is required to store the image files. Furthermore, no extra server is required because the source template machine can be booted as the server. Unlike other hardware solutions, the mechanism proposed does not need to insert an extra PCI card in each machine in order to send and receive data. To the best of the authors' knowledge, this is the first solution that can provide such a light, robust, efficient, and scalable solution for massive deployment. Based on the empirical study, the proposed solution outperformed the previous BDMfaI solution with a 45.289% time savings when 32 computers were deployed. Overall, based on the comparisons with the features and performance of prior works as well as some commercial solutions, the proposed solution offers significant advantages and better performance.

As for the future research possibilities, it is valuable to apply the work done by Marozzo *et al.* [49] to our massive deployment solution so that the energy consumption

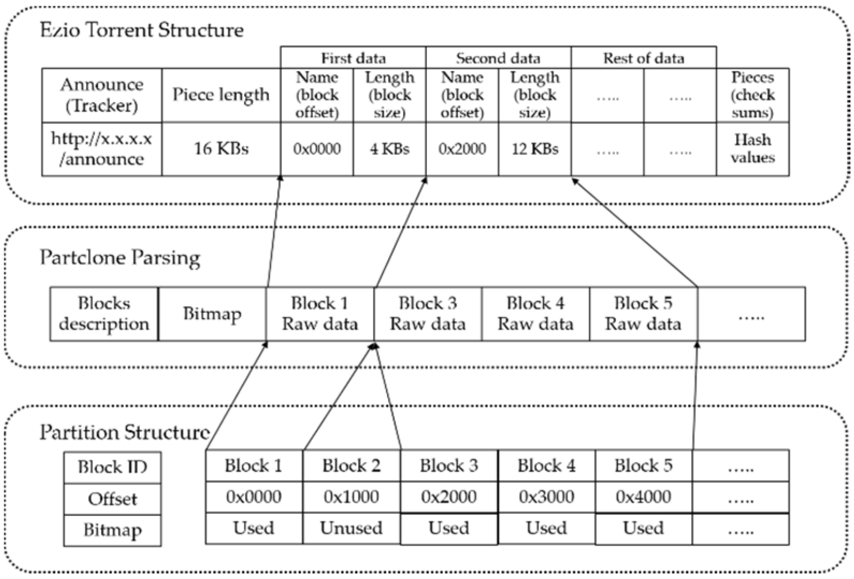


FIGURE 10. The mechanism to parse the used blocks of the file system in the BDMfRD solution.

can be reduced while the impact on the average deployment time can be negligible. Moreover, it is worth extending this research to non-x86 architecture, especially the Advanced RISC Machine (ARM)-based platforms because such solutions are widely adopted in the HPC environment [66], [67] due to their energy-efficient features. Meanwhile, the BT mechanism has been adopted in only system deployments; the network booting process, where every destination machine to be deployed downloads about 300 MB of system booting files, has not been studied. If the number of destination machines increases, the network booting process will become the bottleneck for the whole deployment procedure when all of the destination machines boot at almost the same time. Hence, this improvement is essential when the BDMfRD solution is applied to a deployment with a very large scale. Furthermore, both the BDMfaI and BDMfRD solutions have the potential to be adopted for data replications in multiple sites when the files to be transferred contain numerous small files. In this case, transferring the used blocks on the file system might provide better efficiency than transferring the files for data transmission because, by reading and sending the used blocks on the file system, it does not have to go through the OS file system for every single file. This topic is very suitable for future explorations.

APPENDIX

A. SOFTWARE IMPLEMENTATION

To combine procedures O2 and O3 from the BDMfaI solution (Figure 1) into a single step (N2) in BDMfRD (Figure 2), a new mechanism shown in Figure 10 was introduced. We improved Partclone with the options "-t" or "–btfiles_torrent" for generating the metainfo file required by the BT mechanism in this study. Thus, the used blocks on the file system can be parsed and listed. The new options, "-t" or "–btfiles_torrent," in Partclone are different from the options "-T" or "–btfiles" in our previous study [10]. The option "-t" or "–btfiles_torrent" in Partclone in this study only generates the torrent information file, which lists the offsets and lengths of the used blocks on the file system while the "-T" or "–btfiles" in Partclone in the previous study [10] creates the files in FSBT format based on the used blocks of that file system. Here, we provide an example of using the option "-t" or "–btfiles_torrent" in Partclone. To parse the used blocks of the "ext4" file system on the first partition of the first disk (/dev/sda1), the buffer size for the data being parsed is "16777216" bytes, and the output file "torrent.info" is put in the path /bt-data/, so the command for Partclone is

partclone.ext4 -c -t –buffer_size 16777216 -s /dev/sda1 -o /bt-data/

In this command, the option "-c" is used to enable Partclone to save the information from the file system instead of restoring it.

The content of the generated file "torrent.info" is demonstrated in Figure 11 as an example, where every set of the offset and length on the file system forms the so-called used blocks on the file system. When the used blocks are parsed by Partclone and their size is larger than the buffer, the checksum of the used blocks will be created by the Secure Hash Algorithm 1 (SHA-1) [68], and the checksum will be stored in the "torrent.info" file. Although other modern and better hash algorithms (e.g., SHA-256 or SHA-512) are available nowadays, the checksum is still calculated by the SHA-1 algorithm because the checksum has to be derived using the SHA-1 algorithm according to the BT metainfo definition [47].

Then, a BT metainfo file containing the information, which includes (1) the used blocks list file from the "torrent.info" file, (2) the torrent server, (3) the creator's name, and (4) the partition name, can be created. In this work, a program called "gen-torrent-from-ptcl" was developed to fulfill this purpose. For example, based on the following information: (1) the used blocks from the first partition of the first disk is "sda1"; (2) the torrent server's IP address is "192.168.1.1,"

```
offset: 00000000000000000000000000000000
length: 00000000000000000000000001000000
offset: 00000000000000000000000001000000
length: 000000000000000000000000000a1000
sha1: 9a1cda379847aeae36b41060612ec96f7415a33b
offset: 00000000000000000000000008000000
length: 00000000000000000000000000bc8000
offset: 000000000000000000000000008be0000
length: 00000000000000000000000000019000
offset: 00000000000000000000000008d80000
length: 00000000000000000000000000016000
offset: 00000000000000000000000008da5000
length: 00000000000000000000000000196000
offset: 00000000000000000000000008f3c000
length: 00000000000000000000000000001000
offset: 00000000000000000000000008f3e000
length: 00000000000000000000000000029000
offset: 00000000000000000000000009000000
length: 00000000000000000000000001000000
sha1: 28c6a2cef1b47a44664312cfbae9ca1c1183d6d0
...
```

FIGURE 11. The used blocks list generated by Partclone for the BDMfRD solution; only the first few parts are shown.

with transmission control protocol (TCP) port 6969; (3) the creator's name is "Clonezilla"; (4) the used blocks list file is "/bt-data/torrent.info"; and (5) the desired output BT metainfo file is "/bt-data/sda1.torrent," the following command is executed to combine them and create the metainfo file "/bt-data/sda1.torrent":

gen-torrent-from-ptcl -p sda1 -t http://192.168.1.1:6969/announce -c Clonezilla -i /bt-data/torrent.info -o /bt-data/sda1.torrent

The EZIO program [15] is a BT-based deployment program. In our previous work [10], EZIO can only support the reading of used blocks from an image file. In this work, EZIO has been improved to read and write the used blocks directly from and to a raw device so that we can skip step O2 in Figure 1. The novel BT-based mechanism follows the flowchart in Figure 2. Moreover, with the features being implemented in the BDMfRD solution, there is no need to provide more storage space for extra files in the different formats required in BDMfaI. In the BT mechanism, a peer with all data is called a seeder whereas a peer with only part of the data is called a leecher [10]. In the BDMfRD solution, the seeder only reads the used blocks from the raw device whereas the leecher reads and writes the used blocks directly from and to the raw device. Here, we provide two examples of the improved EZIO serving as the deployment program in the seeder and leecher:

- Seeder:

For the seeder, when the cache size is set as 2459085 KiB (half the size of the free memory on the destination machine; decided before the execution of EZIO), the file containing the used blocks information is "/bt-data/sda1.torrent." The used blocks are loaded from the partition /dev/sda1. No timeout will occur. That is, the program will be executed continually until the process is terminated by the system administrator. The command for EZIO in the seeder is

ezio -U –cache 2459085 -T /bt-data/sda1.torrent -L /dev/sda1

- Leecher:

As for the leecher, when the timeout is defined as one minute, the file containing the information of the used blocks is "/bt-data/sda1.torrent." The blocks are written to and read from the partition /dev/sda1. The leecher will share data if available. So the leecher not only receives data from others but also sends data to them. The command for EZIO in the leecher is ezio -t 1 /bt-data/sda1.torrent /dev/sda1 Here, the one-minute timeout means that, once the leecher finishes deploying the raw device, /dev/sda1, it will continually spend one more minute waiting for other leechers to request data. If no other leechers request data within one minute, the EZIO command will be terminated, and another EZIO command will be started to deploy the rest of the partition(s). After all the partitions are deployed, the post tasks (e.g., boot loader restoring) will be performed.

ACKNOWLEDGMENT

The authors thank those Clonezilla users who provided feedback about the program.

REFERENCES

[1] P. M. Papadopoulos, M. J. Katz, and G. Bruno, "NPACI rocks: Tools and techniques for easily deploying manageable linux clusters," *Concurrency Comput., Pract. Exper.*, vol. 15, nos. 7–8, pp. 707–725, 2003.

[2] A. E. Bruno, S. J. Guercio, D. Sajdak, T. Kew, and M. D. Jones, "Grendel: Bare metal provisioning system for high performance computing," in *Proc. Pract. Exper. Adv. Res. Comput.*, 2020, pp. 13–18.

[3] S. Shiau, C.-K. Sun, Y.-C. Tsai, J.-N. Juang, and C.-Y. Huang, "The design and implementation of a novel open source massive deployment system," *Appl. Sci.*, vol. 8, no. 6, p. 965, Jun. 2018.

[4] J. Xie, Y. Su, Z. Lin, Y. Ma, and J. Liang, "Bare metal provisioning to OpenStack using xCAT," *J. Comput.*, vol. 8, no. 7, pp. 1691–1695, Jul. 2013.

[5] T. J. M. Sanguino, I. F. de Viana, D. A. L. García, and E. C. Ancos, "OpenGnSys: A novel system toward centralized deployment and management of computer laboratories," *Comput. Edu.*, vol. 75, pp. 30–43, Jun. 2014.

[6] D. Daly, J. H. Choi, J. E. Moreira, and A. Waterland, "Base operating system provisioning and bringup for a commercial supercomputer," in *Proc. IEEE Int. Parallel Distrib. Process. Symp.*, Mar. 2007, pp. 1–7.

[7] G. E. Chalemin, I. Naick, C. J. Spinac, and C. L. Sze, "System and method for operating system deployment in a peer-to-peer computing environment," U.S. Patent 11 567 599, Jun. 12, 2008.

[8] J. R. Danielsen, "Method and apparatus for operating system deployment," U.S. Patent 7 546 450, Jun. 9, 2009.

[9] TOPOO. (2020). *TOP CPR Computer Recovery Card*. [Online]. Available: https://www.topoo.com.tw

[10] S. J. H. Shiau, Y.-C. Huang, C.-H. Yen, Y.-C. Tsai, C.-K. Sun, J.-N. Juang, C.-Y. Huang, C.-C. Huang, and S.-K. Huang, "A novel massive deployment solution based on the peer-to-peer protocol," *Appl. Sci.*, vol. 9, no. 2, p. 296, Jan. 2019.

[11] M. Yang and Y. Yang, "Applying network coding to peer-to-peer file sharing," *IEEE Trans. Comput.*, vol. 63, no. 8, pp. 1938–1950, Aug. 2014.

[12] R. Schollmeier, "A definition of peer-to-peer networking for the classification of peer-to-peer architectures and applications," in *Proc. 1st Int. Conf. Peer Peer Comput.*, 2001, pp. 101–102.

[13] Y.-W. Ko, M.-J. Kim, J.-G. Lee, and C. Yoo, "Deduplication TAR scheme using user-level file system," *IEICE Trans. Inf. Syst.*, vol. E97.D, no. 8, pp. 2174–2177, 2014.

[14] Partclone. (2019). *Partclone Project*. [Online]. Available: https://github.com/Thomas-Tsai/partclone

[15] EZIO. (2018). *EZIO Project*. [Online]. Available: https://github.com/tjjh89017/ezio

[16] D. Sampaio and J. Bernardino, "Open source backup systems for SMEs," in *New Contributions in Information Systems and Technologies*. Cham, Switzerland: Springer, 2015, pp. 823–832.

[17] K. Aswani, M. Anala, and S. Rathinam, "Bare metal cloud builder," *Imperial J. Interdiscipl. Res.*, vol. 2, no. 10, pp. 1844–1851, 2016.

[18] B. B. Rad, H. J. Bhatti, and M. Ahmadi, "An introduction to Docker and analysis of its performance," *Int. J. Comput. Sci. Netw. Secur.*, vol. 17, no. 3, p. 228, 2017.

[19] T. Combe, A. Martin, and R. Di Pietro, "To docker or not to docker: A security perspective," *IEEE Cloud Comput.*, vol. 3, no. 5, pp. 54–62, Sep. 2016.

[20] G. M. Kurtzer, V. Sochat, and M. W. Bauer, "Singularity: Scientific containers for mobility of compute," *PLoS ONE*, vol. 12, no. 5, May 2017, Art. no. e0177459.

[21] M. Saleem and J. Rajouri, "Cloud computing virtualization," *Int. J. Comput. Appl. Technol. Res.*, vol. 6, no. 7, pp. 290–292, 2017.

[22] L. Ramakrishnan, A. Liu, P. T. Zbiegel, S. Campbell, R. Bradshaw, R. S. Canon, S. Coghlan, I. Sakrejda, N. Desai, and T. Declerck, "Magellan: Experiences from a science cloud," in *Proc. 2nd Int. Workshop Sci. Cloud Comput. (ScienceCloud)*, 2011, pp. 49–58.

[23] N. Besaw, L. Scheidenbach, J. Dunham, S. Kaur, A. Ohmacht, F. Pizzano, and Y. Park, "Cluster system management," *IBM J. Res. Develop.*, vol. 64, nos. 3–4, pp. 7:1–7:9, 2020.

[24] C. Connor, A. Jacobson, A. Bonnie, and G. Grider, "An innovative approach to bridge a skill gap and grow a workforce pipeline: The Computer System, Cluster, and Networking Summer Institute," *USENIX J. Edu. Syst. Admin.*, vol. 2, no. 1, p. 27, 2016.

[25] V. Holmes and I. Kureshi, "Developing high performance computing resources for teaching cluster and grid computing courses," *Procedia Comput. Sci.*, vol. 51, pp. 1714–1723, Jan. 2015.

[26] M. Younas, I. Ghani, D. N. A. Jawawi, and M. M. Khan, "A framework for agile development in cloud computing environment," *J. Internet Comput. Services*, vol. 17, no. 5, pp. 67–74, Oct. 2016.

[27] W.-T. Tsai, W. Wu, and M. N. Huhns, "Cloud-based software crowdsourcing," *IEEE Internet Comput.*, vol. 18, no. 3, pp. 78–83, May 2014.

[28] S. Thakur, S. C. Gupta, N. Singh, and S. Geddam, "Mitigating and patching system vulnerabilities using ansible: A comparative study of various configuration management tools for iaas cloud," in *Information Systems Design and Intelligent Applications*. New Delhi, India: Springer, 2016, pp. 21–29.

[29] S. Johann, "Kief morris on infrastructure as code," *IEEE Softw.*, vol. 34, no. 1, pp. 117–120, Jan. 2017.

[30] S. Varrette, P. Bouvry, H. Cartiaux, and F. Georgatos, "Management of an academic HPC cluster: The UL experience," in *Proc. Int. Conf. High Perform. Comput. Simulation (HPCS)*, Jul. 2014, pp. 959–967.

[31] D. J. Cougias, E. L. Heiberger, and K. Koop, *The Backup Book: Disaster Recovery From Desktop to Data Center*. Washington, DC, USA: Network Frontiers, 2003.

[32] G. Tiwari, "Automated deployment of windows 7 and application software on large installations of computers having similar hardware," *Int. J. Inf. Electron. Eng.*, vol. 4, no. 3, p. 195, 2014.

[33] K.-M. Lee, W.-G. Teng, J.-N. Wu, K.-M. Huang, Y.-H. Ko, and T.-W. Hou, "Multicast and customized deployment of large-scale operating systems," *Automated Softw. Eng.*, vol. 21, no. 4, pp. 443–460, Dec. 2014.

[34] M. Shojafar, J. H. Abawajy, Z. Delkhah, A. Ahmadi, Z. Pooranian, and A. Abraham, "An efficient and distributed file search in unstructured peer-to-peer networks," *Peer Peer Netw. Appl.*, vol. 8, no. 1, pp. 120–136, Jan. 2015.

[35] A. Baláž and N. Ádám, "Peer to peer system deployment," *Acta Electrotechnica et Inf.*, vol. 16, no. 1, pp. 11–14, Mar. 2016.

[36] Z. Xue, X. Dong, J. Li, and H. Tian, "ESIR: A deployment system for large-scale server cluster," in *Proc. 7th Int. Conf. Grid Cooperat. Comput.*, Oct. 2008, pp. 563–569.

[37] P. Agrawal, H. Khandelwal, and R. K. Ghosh, "MTorrent: A multicast enabled BitTorrent protocol," in *Proc. 2nd Int. Conf. Commun. Syst. Netw. (COMSNETS)*, Jan. 2010, pp. 1–10.

[38] T. N. M. A. Silva, "Bitocast: A hybrid BitTorrent and IP multicast content distribution solution," Ph.D. dissertation, Dept. Inform., Fac. Sci. Technol., New Univ. Lisbon, Almada, Portugal, 2009.

[39] C. M. O'Donnell, "Using BitTorrent to distribute virtual machine images for classes," in *Proc. 36th Annu. ACM SIGUCCS Conf. User Services Conf. (SIGUCCS)*, 2008, pp. 287–290.

[40] E. Jeanvoine, L. Sarzyniec, and L. Nussbaum, "Kadeploy3: Efficient and scalable operating system provisioning for clusters," *USENIX, Login*, vol. 38, no. 1, pp. 38–44, 2013.

[41] C. Neumann, V. Roca, and R. Walsh, "Large scale content distribution protocols," *ACM SIGCOMM Comput. Commun. Rev.*, vol. 35, no. 5, pp. 85–92, Oct. 2005.

[42] S. Luo, H. Yu, K. Li, and H. Xing, "Efficient file dissemination in data center networks with priority-based adaptive multicast," *IEEE J. Sel. Areas Commun.*, vol. 38, no. 6, pp. 1161–1175, Jun. 2020.

[43] F. T. Al-Dhief, N. Sabri, N. M. A. Latiff, N. N. N. A. Malik, M. Abbas, A. Albader, M. A. Mohammed, R. N. Al-Haddad, Y. D. Salman, M. Khanapi, and O. I. O. A. Ghani, "Performance comparison between TCP and UDP protocols in different simulation scenarios," *Int. J. Eng. Technol.*, vol. 7, no. 4.36, pp. 172–176, 2018.

[44] O. Heckmann, A. Bock, A. Mauthe, R. Steinmetz, and M. K. KOM, "The eDonkey file-sharing network," Jahrestagung der Gesellschaft Informatik, Ulm, Germany, Tech. Rep., Sep. 2004, vol. 51, pp. 224–228.

[45] M. Ripeanu, "Peer-to-peer architecture case study: Gnutella network," in *Proc. 1st Int. Conf. Peer Peer Comput.*, 2001, pp. 99–100.

[46] A. Wierzbicki, N. Leibowitz, M. Ripeanu, and R. Wozniak, "Cache replacement policies revisited: The case of P2P traffic," in *Proc. IEEE Int. Symp. Cluster Comput. Grid (CCGrid)*, Apr. 2004, pp. 182–189.

[47] B. Cohen, "Incentives build robustness in BitTorrent," in *Proc. Workshop Econ. Peer Peer Syst.*, vol. 6, 2003, pp. 68–72.

[48] A. Legout, G. Urvoy-Keller, and P. Michiardi, "Rarest first and choke algorithms are enough," in *Proc. 6th ACM SIGCOMM Internet Meas. (IMC)*, 2006, pp. 203–216.

[49] F. Marozzo, D. Talia, and P. Trunfio, "A sleep-and-wake technique for reducing energy consumption in BitTorrent networks," *Concurrency Comput., Pract. Exper.*, vol. 32, no. 14, p. e5723, Jul. 2020.

[50] D. Tracey and C. Sreenan, "How to see through the fog? Using peer to peer (P2P) for the Internet of Things," in *Proc. IEEE 5th World Forum Internet Things (WF-IoT)*, Apr. 2019, pp. 47–52.

[51] S. Vimal and S. K. Srivatsa, "A new cluster P2P file sharing system based on IPFS and blockchain technology," *J. Ambient Intell. Humanized Comput.*, pp. 1–7, Sep. 2019.

[52] M. S. Ali, M. Vecchio, G. D. Putra, S. S. Kanhere, and F. Antonelli, "A decentralized peer-to-peer remote health monitoring system," *Sensors*, vol. 20, no. 6, p. 1656, Mar. 2020.

[53] A. Qadeer, A. W. Malik, A. U. Rahman, H. M. Muhammad, and A. Ahmad, "Virtual infrastructure orchestration for cloud service deployment," *Comput. J.*, vol. 63, no. 2, pp. 295–307, Feb. 2020.

[54] A. Shestakov and A. Arefiev. (2016). *Patch File to Allow User to Provision Image Using Bittorrent Protocol in OpenStack*. [Online]. Available: https://review.openstack.org/#/c/311091/

[55] V. Drok, A. Shestakov, A. Arefiev, P. Shchelokovsky, and S. Kovaleff. (2016). *Cut Ironic Provisioning Time Using Torrents*. [Online]. Available: https://www.mirantis.com/blog/cut-ironic-provisioning-time-using-torrents/

[56] Clonezilla. (2020). *Clonezilla Project*. [Online]. Available: https://clonezilla.org
[57] Clonezilla. (2020). *Massive Deployment Using the BDMfRD Mechanism*. [Online]. Available: https://clonezilla.org/show-live-doc-content.php?topic=clonezilla-live/doc/12_lite_server_BT_from_dev
[58] *Network Booting: Preboot Execution Environment, Bootstrap Protocol, Netboot, Gpxe, Remote Initial Program Load*, LLBooks, Kennewick, WA, USA, 2010.
[59] R. L. Xia and J. K. Muppala, "A survey of BitTorrent performance," *IEEE Commun. Surveys Tuts.*, vol. 12, no. 2, pp. 140–158, 2nd Quart., 2010.
[60] S. Le Blond, A. Legout, and W. Dabbous, "Pushing BitTorrent locality to the limit," *Comput. Netw.*, vol. 55, no. 3, pp. 541–557, Feb. 2011.
[61] R. Stallman, *Free Software, Free Society: Selected Essays of Richard M. Stallman*. Morrisville, NC, USA: Lulu, 2002.
[62] CeCILL. (2013). *Cecill and Free Software*. [Online]. Available: https://cecill.info
[63] SmartDeploy. (2020). *SmartDeploy Solution*. [Online]. Available: https://www.smartdeploy.com/
[64] J. Stokes and M. Singer, *Mastering the Microsoft Deployment Toolkit*. Birmingham, U.K.: Packt, 2016.
[65] S. F. Inc. (2020). *EZ-Backup System*. [Online]. Available: http://www.sanfong.com.tw
[66] D. Yokoyama, B. Schulze, F. Borges, and G. Mc Evoy, "The survey on ARM processors for HPC," *J. Supercomput.*, vol. 75, no. 10, pp. 7003–7036, Oct. 2019.
[67] B. Sorensen, "Japan's flagship 2020 'Post-K' system," *IEEE Ann. Hist. Comput.*, vol. 21, no. 1, pp. 48–49, Jan. 2019.
[68] Q. H. Dang. (2015). *Secure Hash Standard*. [Online]. Available: https://doi.org/10.6028/NIST.FIPS.180-4

STEVEN J. H. SHIAU was born in Hsinchu, Taiwan, in 1971. He received the B.S. and M.S. degrees in nuclear engineering from National Tsing Hua University, Hsinchu, Taiwan, in 1993 and 1995, respectively, and the Ph.D. degree in engineering science from National Cheng Kung University, Tainan, Taiwan, in 2019.

In 1997, he joined the National Center for High-Performance Computing, Taiwan, where he is currently a Research Fellow. His research interests include the development of opensource, free software for diskless systems, massive deployment, and OpenStreetMap.

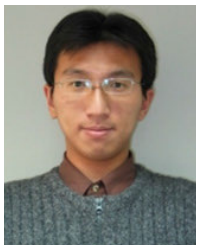
Dr. Shiau won the Public Sector Software Award at the International Free Software Contest, Soissons, France, in November 2007. In 2008, he was recognized for his outstanding contribution in science and technology by the Executive Yuan, Taiwan.

YU-CHIANG HUANG was born in Taichung, Taiwan, in 1994. He received the B.S. degree from the Department of Computer Science and Information Engineering, National Central University, in 2016, and the M.S. degree in computer science and engineering from National Chiao Tung University, in 2018.

He is currently the Senior Engineer with Edgecore Networks, Taiwan, and the Maintainer of the DozenCloud Project and ARM64 VPS Project. His research interests include porting OpenStack to ARM64 and peer-to-peer bare-metal system provisioning in HPC, working on data center networks with open networking technology. He has presented at OpenStack Day Taiwan 2016–2017, Open Source Summit North America 2017, ISC High Performance Project Poster 2018, Hong Kong Open Source Conference 2019, China Open Source Conference 2019, and Open Source Conference Tokyo Fall 2019.

YU-CHIN TSAI is currently a Developer and an Associate Researcher with the Free Software Laboratory, NCHC, Taiwan. He is also a Core Developer of Partclone, a tool for saving file systems on a partition as an image or cloning the file system to another disk. His research interests include middleware and free software research for high performance computing (HPC). His recent research has examined streaming data and AI framework integration systems, and he is also in charge of DAS, a data analysis platform in NCHC.

In November 2007, he won the Public Sector Software Award at the International Free Software Contest, Soissons, France. In 2008, he was recognized for his outstanding contribution in science and technology by the Executive Yuan, Taiwan.



CHEN-KAI SUN is currently a Developer and an Associate Researcher with the Free Software Laboratory, NCHC, Taiwan. His research explores middleware and free software research for high performance computing (HPC). His research interest includes using and developing FOSS. He is also one of the developers for the DRBL, Clonezilla, and DRBL-Winroll used for deploying systems. His recent research has examined data sharing and market platforms. He is also in charge of Scidm, which is a data market platform in NCHC.

In November 2007, he won the Public Sector Software Award at the International Free Software Contest, Soissons, France. In 2008, he was recognized for his outstanding contribution in science and technology by the Executive Yuan, Taiwan.

CHING-HSUAN YEN was born in Pingtung, Taiwan, in 1994. He received the B.S. and M.S. degrees in computer science and engineering from National Chiao Tung University, in 2016 and 2018, respectively.

In October 2018, he joined Appier, a Taiwan AI startup company. His works have improved the infrastructure of this company in terms of both security and efficiency. Since 2016, he has contributed to this research, including the initial development and experimental data.

CHI-YO HUANG received the B.S. degree in electrical engineering from National Cheng-Kung University, Taiwan, in 1990, the M.S. degree in computer engineering from Syracuse University, Syracuse, NY, USA, in 1993, and the Ph.D. degree in the management of technology from National Chiao-Tung University, Taiwan, in 2006. From 1994 to 2006, he worked with the IC industry as an IC Design Engineer, the Marketing Manager, and the Director. He is currently a Distinguished Professor with the Department of Industrial Education, National Taiwan Normal University, Taiwan. His current research interests include the management and applications of technology, multiple attribute decision making, and business data analytics.

• • •