applied sciences

MDPI

Article

A Novel Massive Deployment Solution Based on the Peer-to-Peer Protocol

Steven J. H. Shiau [1,2,*], Yu-Chiang Huang [3], Ching-Hsuan Yen [3], Yu-Chin Tsai [2], Chen-Kai Sun [2], Jer-Nan Juang [1], Chi-Yo Huang [4,*], Ching-Chun Huang [5] and Shih-Kun Huang [3]

1 Department of Engineering Science, National Cheng Kung University, No.1, University Road, Tainan City 701, Taiwan; jjuang@mail.ncku.edu.tw
2 National Center for High-performance Computing, No. 7, R&D Rd. VI, Hsinchu 30076, Taiwan; thomas@nchc.org.tw (Y.-C.T.); ceasar@nchc.org.tw (C.-K.S.)
3 Department of Computer Science, National Chiao Tung University, Hsinchu 300, Taiwan; tjjh89017@hotmail.com (Y.-C.H.); mangoking.cs01@g2.nctu.edu.tw (C.-H.Y.); skhuang@cs.nctu.edu.tw (S.-K.H.)
4 Department of Industrial Education, National Taiwan Normal University, Taipei 106, Taiwan
5 Department of Computer Science and Information Engineering, National Cheng Kung University, No.1, University Road, Tainan City 701, Taiwan; jserv@ccns.ncku.edu.tw
* Correspondence: steven@nchc.org.tw (S.J.H.S.); cyhuang66@ntnu.edu.tw (C.-Y.H.); Tel.: +886-3-5776085 (ext. 335) (S.J.H.S)

Received: 26 November 2018; Accepted: 10 January 2019; Published: 15 January 2019

Abstract: The BitTorrent (BT) is a peer-to-peer (P2P) file sharing protocol that was developed approximately 20 years ago, is becoming increasingly popular, and has been widely accepted. The BT-based mass deployment system can be used to improve performance and scalability that cannot be achieved by the unicasting, broadcasting, and multicasting protocols. However, when the BT-based system is applied in massive deployments, a major issue related to insufficient temporary storage space to store the whole system image before deploying needs to be resolved. Such problems arose because the system is deployed to the disk space, meaning that it cannot be used for temporary storage. Therefore, a novel BT-based solution that can remove the limitations caused by the insufficient temporary storage issue is proposed. The BT-based mass deployment system was designed by using the file system blocks transferring (FSBT) mechanism. The receiver of the FSBT mechanism can obtain the blocks of the file system from other peers. Then, those blocks will be written directly to raw disks or partitions. The sender of the FSBT mechanism can read the blocks of file systems directly from raw disks or partitions. Then, the blocks can be sent to other peers. This approach solves the insufficient temporary storage issue. The novel BT-based mass deployment system was tested and verified for the configuration consisting of at most 32 personal computers (PCs). To demonstrate the achievable performance of the novel BT-based system, comparisons were made between the novel program and the traditional multicast solutions, as well as other solutions for mass deployment. The proposed BT solution can be much faster than the multicast solution when deploying 11 machines or more. The experimental results demonstrated the feasibility and superior performance of the proposed system. Furthermore, performance comparisons of the proposed BT-based mass deployment system versus other solutions demonstrated the feasibility and efficiency of the proposed solution. In the future, the BT parameters can be further optimized, and the simultaneous read and write features can be implemented to improve the deployment performance. In addition, the BT-based mass deployment system can serve as the basis for the development of other mass deployment systems.

Keywords: massive deployment; system deployment; bare-metal provisioning; open source; file system imaging; peer-to-peer (P2P); BitTorrent (BT)

1. Introduction

System deployment or system provisioning [1–3]—the activities that enable the use of the operating system (OS) and applications in the computers that are being deployed—has played a daily critical role in computer system administration in the recent years. Meanwhile, "bare-metal deployment" [4], which is the deployment of an OS over a raw hardware system to be provisioned along with optional software installations and configurations [5], is emerging quickly. As scientific computing and information education are serving a more important role in the modern world, an efficient scheme for massive deployment will largely reduce the maintenance efforts of system administrators. Further, in the era of cloud computing, the hypervisor and container technologies are two major approaches to segment a physical machine into multiple virtual ones. The deployment of host machines that execute the hypervisor or container is becoming more important, because the OSs and applications of the host machine are the basis for the hypervisor and container. Albeit the mass deployment techniques are becoming more important and the number of machines to be deployed simultaneously is increasing daily, the performance and functionality of the current deployment solutions are limited. Consequently, the demands for efficient schemes of massive deployment on cloud servers, computer classrooms, and the high performance computing (HPC) cluster have kept increasing for a long while.

To fulfill the gap between customer needs and functionality, as well as the performance that is available for current mass deployment systems, research institutes and firms have been seeking a variety of solutions for massive deployment in bare metal, hypervisor, and container environments. The current available solutions include the Kickstart Installation [6], Fully Automatic Installation (FAI) [7], Cobbler [8], FSArchiver [9], Ghost [10], Metal as a Service (MaaS) [11], Mondo Rescue [12], OpenStack [13], Storix System Backup Administrator (SBAdmin) [14], Partimag [15], and OpenGnSys (Open Genesis) [16]. These massive deployment systems rely on the network connection, booting, and mechanisms such as unicast, broadcast, and multicast for massive deployment.

However, due to the limitations regarding the performance and scalability of these currently available network-based mass deployment systems, the possibilities of improving these systems are very low. Limitations for the widely available network-based deployment systems, as well as the previously mentioned unicast, broadcast, and multicast systems, are summarized as follows. For the unicast mechanism, all of the receivers are connected to the central sender to request data. Hence, both the network bandwidth and system loadings of the sender will become a bottleneck sooner or later as the number of receivers increases. For both broadcast and multicast systems, the User Datagram Protocol (UDP) network protocol is used for sending data [17]. However, as no handshaking mechanism is available, these two systems are not reliable; packet losses may happen at any time. Once the packet loss problem occurs, the receiver(s) will request that the sender sends the packet(s) again. As more receivers exist, the possibilities of, and thus the frequencies for the losses and requests of packets, increase. Although broadcast and multicast systems can provide one-to-all or one-to-many solutions, the issue of lost packet requests makes both systems inefficient and unable to be scaled up to support a large number of receivers.

The BitTorrent (BT) protocol [18] is a peer-to-peer (P2P) [19] file-sharing technology that was developed approximately 20 years ago, is becoming increasingly popular, and has recently been widely accepted for file sharing on the Internet. Typically, the Transmission Control Protocol (TCP) is the transport protocol for the BT solution. The BT solution also has some extension protocols such as the Micro Transport Protocol (uTP) [20], a UDP-based protocol that implements a congestion-control algorithm in order to provide reliable data transportations while reducing congestion control issues and poor latencies that can be found in the typical BT over the TCP. Nowadays, P2P systems have widely been adopted in different areas, including file distribution [21–24], audio and video streaming [25], voice over Internet protocol (VoIP) [26,27], content distribution [28], and even digital cryptocurrencies [29–31]. The major advantages of the P2P protocol include their efficiency, reliability, and scalability. Thus, the P2P protocol can be used to fill the technology supply and demand gap

in massive deployment via broadcast or multicast protocols. Many protocols are available for P2P file sharing, and the BT is one of the most popular file-sharing protocols with the highest adoption rate. However, when the BT protocol is used in bare-metal deployment, the problem stemming from insufficient temporary storage space to store the whole system image should be resolved. This temporary storage shortage issue happens because in bare-metal deployment, the whole local hard disk is the storage place where the OS and applications are to be deployed. Therefore, the hard disk cannot be used as a temporary storage space, as the local hard disk will be in the busy status when it is mounted to keep the image received from the sender. A single disk cannot be used for two purposes unless there are two or more disks in the computer. However, the availability of a second or more disks or partitions cannot be guaranteed. The random access memory (RAM) on the computer to be deployed is a possible candidate for serving as the temporary storage space. However, one cannot guarantee the availability of sufficiently large enough memory space to store the whole system image for both OS and applications.

Open source or free software allows for the public availability of the software source code, thereby permitting modification. The copyright owner of open source or free software permits others to use, copy, modify, and distribute the source code [32,33]. The open source or free software model empowers programmers to "stand on the shoulders of giants", so this model accelerates the use and improvement of existing software. Therefore, many high-quality programs have been developed and made available for the public. A lot of programs that use BT protocol to provide the P2P file-sharing ability are released under an open source, free software license. For example, BitTornado [34] was released under an Massachusetts Institute of Technology (MIT) license [35], while qBittorrent [36] was released under a General Public License (GPL) [33].

Although the BT protocol has been widely adopted in file sharing and many researchers have implemented the protocol for system deployment, to the best of the authors' knowledge, no BT-based deployment program in general can deploy OSs and applications absolutely without the availability of the temporary shortage issue. Hence, a novel efficient and reliable massive deployment system using the BT protocol that guarantees the system requiring no extra RAM or disk space is proposed. Furthermore, the system is backward compatible with older versions of mass deployment systems with the image format and procedure that the authors developed in the previous research [3]. The backward compatibility ensures interoperability with an older version of system deployment software. The backward compatibility support of the proposed BT-based mass deployment system is essential for existing users, because the novel system will not strand the users. Moreover, the diffusion and adoption of the novel system can be enhanced. When new features are added to an existing system, some functionalities might be removed or modified to prevent possible conflicts between the novel system and existing one(s). To support backward compatibility, some efforts are required. The development of the novel system can be slowed down. For the open source community, the backward compatibility is normally required for open source software. In addition, a novel open source system that provides backward compatibility allows minimal recoding and reintegration with existing systems [37]. In this work, we follow the general rules of the open source community. Hence, the newly added features in this study were developed to be compatible with our previous research [3] by providing the image conversion ability and the compatible deployment procedure.

The proposed mass deployment system adopts the BT protocol based on the file system blocks other than the traditional image files of the OSs as well as applications to be deployed. The Ezio [38] program was developed for this purpose. In addition, the implementation and integration of the P2P protocol to our prior work [3] were also addressed. To verify its feasibility and demonstrate efficiency, a novel system was implemented, tested, and verified in the experiments to deploy up to 32 physical personal computers (PCs). The performance of the proposed system versus that of other mass deployment programs was compared. Furthermore, the differences between the proposed method and the ones adopted by other existing systems are presented. The experimental results, comparisons, and discussions in this study demonstrate the feasibility and efficiency of the proposed system.

The rest of this paper is organized as follows. Section 2 reviews the massive deployment network protocols and discusses related works. Section 3 defines the system architecture, the massive deployment system with P2P protocol, and software implementation. The experimental results are presented in Section 4. The results and comparisons are discussed in Section 5. Section 6 concludes the whole work, and provides suggestions for future research.

2. Related Works

This section reviews and discusses the network protocols and related works for massive deployment. At least four types of communication protocols, including the unicast, broadcast, multicast [39], and P2P networking [40] can be used for massive deployment. Normally, the unicast protocol is not suitable for massive deployment due to the limitations of scaling up. The availability network bandwidth for each client machine decreases linearly as the number of client machines increases when a unicast protocol is used. The broadcast, multicast, and P2P protocols have specific advantages and disadvantages when applied to massive deployment. In the following sub-sections, the works related to unicast, broadcast, multicast, and P2P mechanisms are addressed.

2.1. Unicast, Broadcast, and Multicast Protocols for System Deployment

In computer networks, the most commonly seen routing schemes for data transmission include unicast, broadcast, and multicast [41–43]. The unicast protocol was designed for data transmissions between one sender and one receiver. The broadcast protocol was designed for data transmissions between one sender and more than one receiver. The multicast protocol was designed for data transmissions between one or many senders to many receivers. Figure 1 illustrates the topology of the three networking protocols.

The broadcast and multicast protocols differ in the selection of receivers. For the broadcast protocol, the receivers are those connected to the network (e.g., the entire subnet). The broadcast protocol automatically forwards the data to all of the receivers. There is no need and no allowance to select the target receiver(s) to receive the data. On the other hand, for the multicast protocol, the receiver(s) can decide whether they are going to join the subnet and receive data or not, although these receivers are accessible by the sender(s). Hence, the multicast protocol has better transmission efficiency than the broadcast protocol, because the receivers that are connected to the network will not be influenced by the sender if those receivers do not join the multicast group.

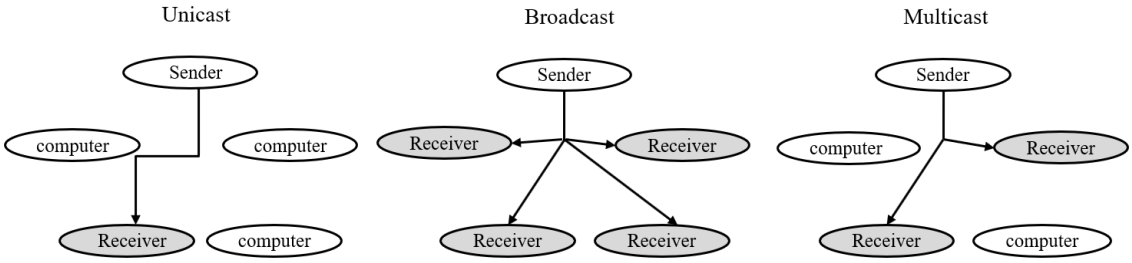


Figure 1. The unicast, broadcast, and multicast solutions (modified from reference [17]).

Nowadays, the multicast protocol is the most efficient mass deployment protocol among the previously mentioned widely adopted protocols. Therefore, several researchers have designed and developed multicast-based solutions for massive data deployment. Grönvall et al. [42] developed the JetFile file system, which uses multicast protocol as the mass deployment mechanism. Lee et al. [39] proposed a multicast approach to customize the deployment of OSs, and compared the proposed approach with other existing solutions, such as the Ghost and file transfer protocol (FTP), PreOS, and bootable cluster compact disc (BCCD). They asserted that the proposed approach was better for serving as a reliable multicast protocol, a heterogeneous infrastructure, and cloud hypervisor environment. The authors [3] also developed a mass deployment solution by incorporating the

UDPcast [39] in the previous study to support broadcast and multicast protocols. Although the multicast-based mass deployment solution is comparatively efficient and has been widely adopted, the packet loss problems caused by the UDP protocol influence the solution's scalability. When more clients join the multicast group, the frequencies of packet losses increase. The packet lost in the multicasting process should be resent to those receivers that have not received the packet without any fault. Thus, other receivers that have already successfully received the packets will wait until all of the receivers have received the correct packets. Therefore, the system based on multicast protocol cannot be scaled up for large-scale deployments. As a result, P2P-based solutions were proposed to resolve the scalability problem in the large-scale deployment. Section 2.2 reviews previous works related to the P2P protocol.

2.2. P2P Protocol for System Deployment

The P2P sharing leverages a distributed network architecture to enable participants to share part of their own computing resources to provide the services and contents offered by the network. These resources are directly accessible by other peers in the network. In addition, the participants of the distributed network are both resource providers and resource requestors [19]. Many protocols have been designed for P2P file sharing, such as the BT [44], eDonkey [45], FastTrack [46], and Gnutella [47]. Many applications based on the P2P protocol have been developed, include the BT software, Freenet [48], and Gnutella [47]. The BT protocol was proposed by Cohen [44] in 2001 and implemented as software. In the BT-based solution, the file to be shared is cut into segments called pieces. The general size of the pieces varies from hundreds of KBs to a few MBs. Normally, a BT-based distributing environment contains three parts: (1) a torrent server, where the torrent metadata file is kept; (2) a torrent tracker, which keeps track of the senders and receivers; and (3) peers, which are the instances on the network that can transfer data. A typical topology of such an environment is demonstrated in Figure 2.

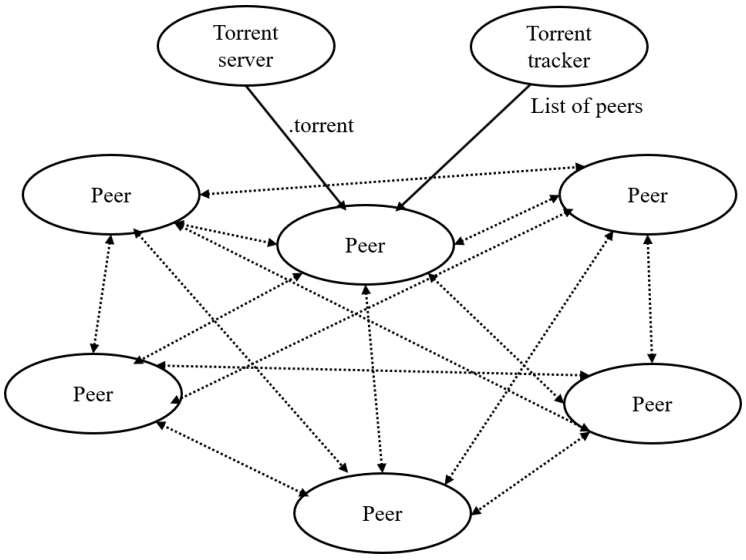


Figure 2. The P2P mechanism (adapted from reference [49]).

A peer having the entire file is called a seeder whereas a peer having only part of the file is called a downloader or a leecher [44]. To distribute a file in the BT-based solution, the initiator (i.e., the computer with the complete file) must generate a metainfo or metadata file with the ".torrent" file extension (e.g., os.torrent). The metainfo file contains the information of the torrent tracker and the metadata of the file to be distributed. The metadata of the file includes the file name, piece size, length, and the hashing information for all of the pieces derived by the Secure Hash Algorithm 1 (SHA-1). The torrent tracker is the service managing the file distribution. When a peer intends to

download the file, the metainfo file should be received first. Then, the BT client program will be used to connect the tracker assigned in the metainfo file. The tracker informs the BT client, and other peers can offer the pieces of the file. Once the peer that intends to download the file has the pieces of the file, the tracker will be informed. Thus, the peer that has downloaded the piece can also play the role of a piece provider as well as the downloader. In order to ensure the high availability of resources in the BT protocol, the peers download pieces in a random or in a "rarest-first" [50] approach, which enables the peers to search for the rarest piece available among the peers. Based on the solution provided by the BT protocol, the server(s) and the network can operate in an efficient manner when sharing large files. Thus, the scalability of the BT-based solution can be assured.

However, the torrent tracker can be the bottleneck in the BT-based solution, because the characteristics of the torrent tracker are not distributed. Using a distributed hash table (DHT) [51], which is the mechanism that can decentralize a distributed system by providing a lookup service, the previously mentioned torrent tracker is no longer required by the BT-based solution. Hence, the DHT removes the bottleneck and can scale up the peers in the distributing system without losing efficiency.

Since the BT-based solution is suitable for distributing large files and can be scaled up, many researchers have investigated related topics. Dosanjh et al. [52] proposed a BT-based solution to support dynamic-shared libraries in the HPC systems. They claimed that the proposed approach had substantial potential for improving the support for applications that can rely on dynamic linking and loading libraries in HPC systems. Ren et al. [34] designed, implemented, and evaluated a topology-aware BT system (TopBT) that is infrastructure-independent. Based on the proof provided by Ren et al. [34], such a BT client can significantly improve the overall Internet resource utilization without degrading users' downloading performance. Compared with several prevalent BT clients, the TopBT can reduce download traffic by about 25% while achieving a 15% faster download speed.

The BT-based solution was also used to deploy the image of virtual machines in cloud computing. García and Castillo [53] proposed a BT-based solution that can cache the virtual machine images in the local computers to be deployed. Such a system can reduce the transfer time when a large number of requests are raised. Chen et al. [54] also used the BT protocol to accelerate the image delivery to the cloud-based environment with a large number of virtual machines. In addition, Chen et al.'s solution [54] was integrated into the Filesystem in the Userspace (FUSE) software interface through a message-driven piece selection scheme. Thus, virtual machines and requested services can be enabled within the required time. O'Donnell [55] also implemented a BT-based solution to distribute the images of virtual machines to computers. By gluing some freely available applications and their own scripts with the BT protocol, the completion time was reduced to less than one-sixth.

As for the bare-metal provisioning based on the BT protocol, Xue et al. [56] introduced a BT-based solution that aims to achieve the efficiency, scalability, independence, and reliability (ESIR) of image transfers in mass deployment. Comparing the performance of both the multicast and ESIR-based solution, the ESIR-based system is more reliable, whereas the performance of image distribution is higher. Jeanvoine et al. [57] designed and implemented a BT-based solution in the Kadeploy3 software to perform bare-metal provisioning. The software has been used intensively on the Grid'5000 test-bed since the end of 2009. Approximately 620 different users have performed 117,000 deployments. On average, each deployment involved 10.3 nodes. The largest deployment involved 496 nodes. Thus, Kadeploy3 is particularly suitable for large-scale clusters. Anton and Norbert [58] also proposed system architecture utilizing the P2P protocol. The throughput can be increased while multiple OSs are deployed. According to Anton and Norbert [58], a system using the BT-based solution is much more efficient when deploying a larger number of machines than the performance that can be achieved by the traditional unicast mechanism. Shestakov and Arefiev [59,60] proposed adding a BT solution to the original OpenStack Ironic project in order to shorten the bare-metal provisioning time. They embedded the BT functionality directly on the storage nodes. The BT functionality is right on top of the OpenStack Swift Application Programming Interface (API). The image is downloaded directly from the object

store. Basically, the process consists of three steps: (1) update the OpenStack with appropriate drivers and patches; (2) create the torrent file and associate it with an image; and (3) boot the new node using the torrent-based image. The node will be able to use both peers and the object storage to download the image as well as share the downloaded data with other peers. The BT solution requires 15 minutes to provide 90 nodes with a three-GiB image. Compared to the solution based on the standard OpenStack Ironic, which took one hour to provide 15 nodes with a three-GiB image, the BT-based solution is more efficient. Based on the analytic results by Shestakov and Arefiev [59,60], the BT-based solution requires only 25% of the provision time as the standard OpenStack solution, even when the number of nodes to be deployed is six times more.

However, existing research related to bare-metal provisioning using the BT protocol has focused primarily on the efficiency, scalability, and reliability of the systems. To the best of the authors' knowledge, no existing study has tried to address the issue related to the shortage of temporary storage. Existing studies associated with bare-metal provisioning using the BT protocol mentioned in this section have all implicitly assumed the image size to be smaller than the available RAM size of the designated machine to be deployed. This assumption is applicable only when the image is designed for initial system installation. Such images contain the basic packages only. Thus, it is small enough to be stored in most RAMs. For an image consisting of the OSs and all of the required applications, the possibility of an image size to be larger than the available RAM size of the designated deploying machine is very high. Consequently, a solution dealing with the shortage of temporary storage issue must be designed and implemented.

2.3. Other Related Works

As previously mentioned, existing solutions for massive deployment depend mainly on the network connection and the file distribution protocols. In addition to such research, more efforts related to system deployment have focused on the live system [3,61,62], network booting (netboot) [63], Dynamic Host Configuration Protocol (DHCP) service relay [64], and post-deployment tuning, etc. These works are further reviewed in this section.

2.3.1. Live System

A live system usually means an OS booted on a computer from a removable medium, such as a compact disc read-only memory (CD-ROM) or universal serial bus (USB) stick, or from a network, and the OS is ready to use without any installation on the usual drive(s), with auto-configuration done at run time [65]. Since a live system is not loaded from a computer's hard drive, basically, nothing must be saved to the hard drive. Thus, the contents of the computer executing a live system will not be influenced. Live systems are extremely suitable for system deployment because when a computer is in the so-called bare-metal status, there is no OS or application that can be used for system deployment. By using a live system, the machine to be deployed can be booted. Then, the system deployment programs can be executed. The OS and applications can be written into the hard drive of the machine. Live systems are available for major OSs by the system providers. Common live systems include: (1) Linux live system (e.g., Debian Live [66], CentOS Live [67], Ubuntu Live [68]) and (2) Microsoft Windows live system (e.g., Windows Preinstallation Environment (PE) or WinPE in short [69]).

2.3.2. Netboot

The netboot-based mechanism enables the bare-metal machine to boot from the network interface card (NIC) instead of the local device. To achieve this purpose, the netboot-based mechanism requires specific services and files. From the server aspect, the provision of DHCP and Trivial File Transfer Protocol (TFTP) services [63] is essential. In addition, the server must provide the OS and applications so that clients can boot and execute. Since a live system contains the OS and applications, the server can just provide a live system for clients to boot from a network. From the client's aspect, a netboot mechanism based on the Preboot Execution Environment (PXE) [63] or the Unified Extensible Firmware

Interface (UEFI) network boot function [70,71] must be enabled on the Basic Input Output System (BIOS) of the machine's motherboard. By booting from the PXE or UEFI netboot function, the machine to be deployed can be booted from a network, retrieve a live system, and then initiate a deployment.

2.3.3. DHCP Relay Mechanism

The DHCP service relay is also an important function for the netboot-based mechanism. In most HPC clusters or computer classrooms, usually the DHCP service has been provided in the Local Area Network (LAN) by system administrators. However, the netboot mechanism normally provides another DHCP service, as described in Section 2.3.2. The co-existence of two DHCP services in the same LAN will cause conflicts. Thus, a DHCP relay mechanism will be very helpful in resolving these conflicts. Nowadays, the techniques of a traditional DHCP relay mechanism has become very mature. However, the traditional DHCP service cannot fulfill the needs of the netboot-based mechanism. The DHCP service required to support the netboot-based mechanism must also provide the information of the TFTP service to client machines. Due to the importance of DHCP functions, various researchers have proposed solutions. One of the well-known solutions is an open source program called Dnsmasq [72]. The software has provided the DHCP relay function for the PXE mechanism. However, when the secure boot is enabled by the UEFI mechanism, Dnsmasq is unable to relay a complete netboot mechanism for Linux client machines. Thus, the issue should be solved. Although some developers have provided solutions by either disabling both the existing DHCP service in the netboot server and the secure boot mechanism in the client machines, or modifying the boot loader program [73], these works have not been widely accepted by mainstream Linux systems.

2.3.4. Post-Deployment Tuning Process

The post-deployment tuning process is the mechanism for adjusting the machines to be deployed. After massive systems are deployed, some settings of each individual machine in the HPC clusters or computer classrooms should be unique. For example, the computer names and security identifiers (SIDs), etc., should be different for each PC with Microsoft Windows. For the computers with the Linux system, the hardware records for the media access control (MAC) address of the NIC and the disk identification (ID) must be modified to fit the machine to be deployed. To fulfill the needs of the post-deployment tuning process, Microsoft provides Sysprep [74]. The open source solution DRBL-Winroll [75] was also developed for the purposes. All of the post-deployment tuning programs are used to help the massive deployment programs finish the last parts, which cannot be done during a deployment. In the future, the turning functions can be added in the massive deployment program so that the massive deployment can be more efficient.

Table 1 summarizes the existing solutions for system deployment mentioned in this section. Based on the review results, the BT-based solution has the best scalability and can be used for data transmission between many-to-many nodes. Although the BT-based solution consumes the most bandwidth, from the aspect of performance enhancement, such a solution outperformed others. Therefore, in this work BT was chosen as the solution for the massive bare-metal provisioning.

Table 1. Comparisons of existing solutions for system deployment. BT: BitTorrent, TCP: Transmission Control Protocol, UDP: User Datagram Protocol, uTP: uTP Micro Transport Protocol.

Solution	Network Protocol	Transmission	Bandwidth Comsumption	Scalability
Unicast	TCP	One to one	Average	Worst
Broadcast	UDP	One to many, many to many	Least	Average
Multicast	UDP	One to all	Least	Average
BT	TCP, uTP, etc.	Many to many	Most	Best

The review of past works magnifies the scalability problem existing in current massive deployment solutions based on the broadcast or multicast protocols. Although the BT protocol-based solution can scale up the deployment, the problem of insufficient temporary storage still exists. Traditionally, adding more hardware resources to the deploying machine (e.g., putting enough RAM or adding secondary storage) are the sole solutions. Therefore, a novel scheme that can apply the BT protocol on massive deployment without adding hardware resources is very important to improve the massive deployment performance.

3. Design and Implementation

This research aims to propose a novel P2P scheme for massive system deployment. To accomplish this, a novel BT-based solution based on the authors' previous work [3] was developed by using the open source technique. In this section, the design of the software architecture, the design of the massive deployment solution using the P2P protocol, and the software implementation will be introduced.

3.1. Software Architecture for P2P Massive System Deployment

The open source P2P massive deployment system that is proposed in this work can fulfill the above-mentioned features and differs from existing solutions. The major features are summarized as follows. (1) Resolving the insufficient temporary storage issue: Traditionally, the file size of the P2P image data to be deployed should be smaller than the RAM size of the machine to be deployed. Otherwise, extra space for temporary storage will be required. The proposed solution can solve the problem. (2) Scalability: As the peers increase, the performance of the proposed system also increases. Thus, scalability can be achieved. (3) Compatibility: The proposed system can be fully compatible with our previous version of mass deployment software. Thus, existing image files can be deployed by the proposed BT-based solution. The features of our previous work [3], which include an open design, full automation, customization support, and the provision of a live system, can also be inherited.

In this study, the massive deployment system was implemented in a P2P-based solution, where the BT protocol was introduced to distribute the data of the file system. The system architecture (see Figure 3), including the computer hardware, live OS, deployment system, and P2P mechanism for massive deployment, was proposed. Four modules are included in the deployment subsystem [3]: (1) partition table processing; (2) partition image processing; (3) pre-processing, and (4) post-processing modules. Two sub-modules, including the file system processing sub-module and the compressing and decompressing sub-module for the partition image, are included in the partition image processing module. When massive machines are connected from the network to the system, the bare-metal system provisioning can be achieved by using the BT data-sharing module, which includes the torrent server, torrent tracker, and peers (seeders and leechers). More explanations and information regarding these parts will be introduced in Section 3.2.

Only one module from the deployment subsystem (refer to Figure 3)—namely, the partition image processing module—is directly related to the massive deployment module, because the major parts of the data being deployed are the partition images, which usually take a few to hundreds of GB. Therefore, a specific module is designed to enhance the performance of deployment. On the other hand, the information from the partition table is less than one KB. Furthermore, the pre-processing and post-processing modules do not use any data; instead, these two modules only execute some tasks before and after the partition is deployed. The file system blocks transferring (FSBT) data are massively distributed to all of the machines by the BT data-sharing module. Once the machine to be deployed receives the data, the machine's partition image processing module will write the FSBT data to the partition, and start sharing the FSBT data to other machines based on the BT protocol. Section 3.2 defines the details of massive deployment with the P2P protocol.

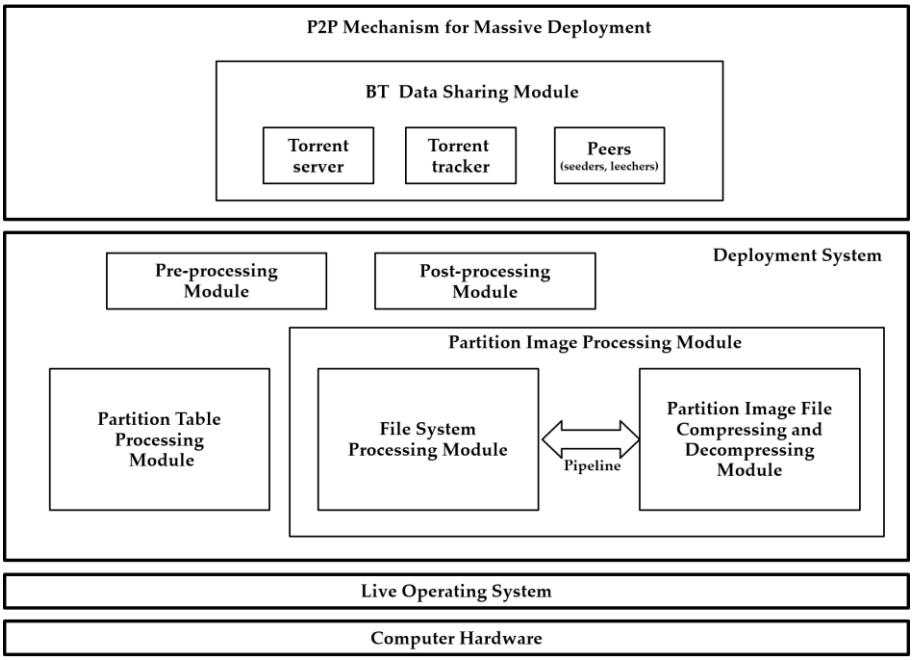


Figure 3. The proposed system architecture for P2P massive system deployment.

3.2. Massive Deployment with the P2P Protocol

In order to overcome the shortage issue of temporary storage, an FSBT-based mechanism was proposed to replace the image file from the file system transferring (IFFST)-based mechanism. The receiver of the FSBT mechanism can receive the blocks of the file system from other peers, and then directly write the received blocks to a raw disk or a partition. The sender can read the blocks of the file system from a raw disk or a partition directly, and then send these blocks to other peers. The P2P mechanism was implemented in a program called Ezio [38]. Further details on Ezio are provided in Section 3.3.

As reviewed in Section 2.2, many protocols are available for the P2P-based solution. The BT protocol was chosen due to the broad availability of the open source BT client programs, which include aria2c [76], BitTornado [34], Enhanced CTorrent [77], LFTP [78], qBittorrent [36], rTorrent [79], and Transmission [80]. In addition, the actively developed open source project, Libtorrent [81], provides a lot of libraries related to BT protocol. The performance of file distribution developed by the Libtorrent library has proven to be effective by another researcher [76].

In massive deployment, users prefer to store multiple system images in the repository, which can be achieved by the authors' previous work [3]. Therefore, various OSs and applications can be available for deployments in different scenarios. However, a mechanism that can be used to send the blocks of the file system from the previously saved image should be developed. By converting the previously saved image data to the blocks of the file system, the BT client program (e.g., Enhanced CTorrent, Transmission, or Ezio) can distribute the FSBT data to all of the clients in the BT-based solution. The procedures related to the P2P-based massive deployment, which is compatible to the authors' previous work [3], can be achieved by using the FSBT mechanism illustrated in Figure 4. In the first step, the image of the OS and applications is kept in the image repository in the local hard drive of the image server. The image is then converted to FSBT files. The metainfo file of the image is created. Once the metainfo and FSBT files are ready, the tracker will be initiated. Finally, the BT clients can be initiated to deploy the system.

As for the machines to be deployed, for example, if some specific peer—namely, peer x—deploys the OS and applications to the first partition in the hard drive (sda1), the metainfo file for sda1 (i.e., sda1.torrent) will first be received from the image server. Then, based on the information provided in the metainfo file, peer x can receive and send the FSBT data to other peers by using the BT-based

solution. Since the image server has the complete FSBT data for sda1, which means the image server is a seeder, only the FSBT data must be forwarded to other peers. There is no need for the peer on the image server to receive the FSBT data from other peers. After other peers receive the FSBT data from either the image server or the rest of peers, the FSBT data can be written to their local hard drives. Those peers with the FSBT data can start sending these data to the rest of the peers by reading those data from their local hard drives. This BT solution enables all of the peers to be servers and clients. Hence, the massive deployment efficiency can be greatly enhanced.

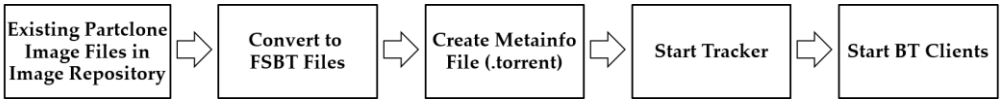


Figure 4. The flowchart about seeding and distributing FSBT files to all machines. FSBT: file system blocks transferring; BT: BitTorrent.

3.3. Software Implementation

This sub-section introduces the concepts and procedures for implementing the sub-module for transferring the blocks of the file system as well as the implementation and integration of the P2P protocol into the upgraded version of Clonezilla live, which is the open source live system developed in the authors' previous study [3].

3.3.1. Implementation of the Ezio Program for Transferring Blocks of the File System

The previously mentioned FSBT mechanism is based on transferring blocks of the file system. Therefore, a utility must be developed to read and write the blocks of the file system directly. The utility should be able to read and write the blocks from either raw devices or files. In computing storages, a block of a file system can be treated as the physical record, which is a sequence of bits or bytes. On the other hand, a file is the record stored on the device through blocks and inodes, the data structure that keeps the locations of disk blocks, as well as attributes of the file or directory data [82]. In this study, an FSBT utility called Ezio implements the transferring function of blocks based on the BT protocol. Ezio has three goals. (1) Firstly, it can serve as the receiver of the FSBT mechanism. Ezio can receive the blocks of file system(s) from other peers, and those blocks can then be written directly to raw disk(s) or partition(s). (2) Secondly, it can serve as the sender of the FSBT mechanism: Ezio can read the blocks of file system(s) directly from raw disk(s) or partition(s), and those blocks can then be sent to other peers. (3) Thirdly, it can serve as the mechanism to send the blocks of file system(s) from the previously saved image(s).

In order to meet the goals, we first define the image format for the FSBT mechanism. The proposed image format can be divided into three structures: the partition, Ezio torrent, and the Partclone [3] (see Figure 5). The partition structure at the bottom of the illustration includes the blocks of the file system on the top of a partition in the hard drive. The Partclone image structure includes the image file structure saved by the Partclone. The Ezio torrent structure enables the mapping of the blocks on the file system from either the partition or the image saved by the Partclone. The ability to map the blocks on the partition allows Ezio to read and write the blocks directly to and from the partition. Meanwhile, the ability to map the image saved by Partclone ensures the compatibility of Ezio to the existing images of Clonezilla, which was an earlier version of the massive deployment system proposed and implemented by the authors [3]. Thus, users can convert the existing images to the Ezio image format without any problem. In the design of Ezio, the file name of blocks data is defined as the offset address on the partition and is expressed in a hexadecimal format. Furthermore, the size of the used blocks is defined as the length of the file, whereas consecutively used blocks are combined in a file.

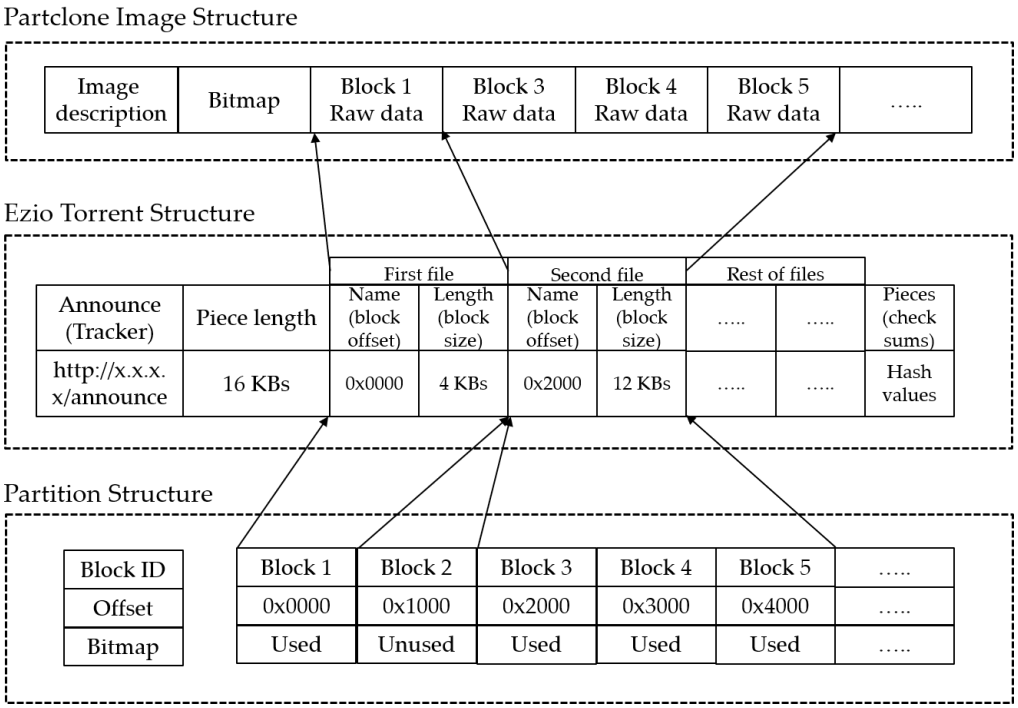


Figure 5. The image format for the proposed FSBT mechanism in this study.

For example, the used blocks for the first partition on the first disk (sda1) saved in the Ezio format are shown in Figure 6, which indicates that each file keeps a section of continuous blocks of the file system storing the data. The file name denotes the offset on the partition. Hence, the file '00000000000000000000000001f0fa000' in Figure 6 contains the data of used blocks starting from the offset address 0x1f0fa000 on partition sda1. The Ezio program was developed using the C++ language, and the BT library Libtorrent was heavily used. As the image of a partition may contain sensitive data, the DHT function was turned off so that the data would not be distributed to undesired peers. In addition, the mechanism for Ezio to read the blocks from FSBT files is compatible with other BT client programs, such as Enhanced CTorrent and Transmission. Therefore, Ezio installed on the image server can be optional, and Ezio can be replaced by a compatible BT client program (e.g., Enhanced CTorent or Transmission). Such design diversifies the whole mechanism and increases the robustness of the massive deployment system. Based on this method, not only can Ezio be adopted, but other BT client programs can also be adopted on the image server. Furthermore, Ezio can read and write the FSBT data directly from and to the local hard drive on the deployed machines. This feature of Ezio is unique; no other BT client programs have similar functionalities.

```
root@debian:~# ls /home/partimag/btzone/stretch-x64/sda1
00000000000000000000000000000000 00000000000000000000001f0fa000
00000000000000000000000440c0000 00000000000000000000000100000
00000000000000000000001f120000 00000000000000000000044100000
00000000000000000000000200000 00000000000000000000001f123000
00000000000000000000044120000 00000000000000000000000300000
00000000000000000000001f125000 00000000000000000000044140000
00000000000000000000000400000 00000000000000000000001f175000
00000000000000000000044180000 00000000000000000000000500000
00000000000000000000001f179000 00000000000000000000441af000
00000000000000000000000600000 00000000000000000000001f193000
```

Figure 6. The proposed image format for the FSBT mechanism.

3.3.2. Implementation and Integration of P2P Protocol with Previous Work

In addition to Ezio, more mechanisms should be developed and implemented based the authors' previous work [3]. Therefore, the P2P protocol can be used for massive deployments. These mechanisms include: (1) the conversion of images with the existing Partclone [3] format to the Ezio format, as described in Section 3.2, and (2) the assignment of the time and counts by a modified tracker for the BT service—namely, the ocs-bttrack [83], which can stop the tracker, thereby informing all of the peers regarding the termination of the job.

For the first part of the newly added mechanism (i.e., the conversion of images using the existing Partclone format to the Ezio format), the option "–btfiles" or "-T" in short enables the proposed solution to convert the image file to the FSBT format. Therefore, existing images can be read and then extracted as the FSBT format. Thus, a program based on the upgraded version of Partclone [3] is developed as another wrapper program called ocs-gen-bt-slices. An existing image of Clonezilla (i.e., IMAGENAME) can be converted to the FSBT format using the command "ocs-gen-bt-slices IMAGENAME". The converted files are then stored in the btzone sub-directory of the image repository. For example, in Figure 6, the converted FSBT files for the BT solution from the "stretch-x64' image, which has the image files for partition sda1, are put in the directory: /home/partimag/btzone/stretch-x64/sda1.

The second part of the newly added mechanism is the modified tracker program. The open source program (i.e., bttrack) developed by the BitTornado [34] was modified to ocs-bttrack. In addition, the options "–tracker_timeout" and "–tracker_max_completed" were added to the ocs-bttrack program. In other words, after the BT service for the massive deployment is started, the maximum time and maximum number of connecting clients that have finished seeding in the BT solution will stop the tracker. Ezio is designed to stop seeding when it cannot contact the tracker within a certain time; by default, the time is 30 s.

Based on these two mechanisms, our previous work [3] was enhanced to include the massive deployment feature based on the BT protocol. Figure 7 demonstrates the schematic diagram. The BT deployment solution includes the deploying server and client machines. The services provided by the deploying server include: (1) netboot-related services, including the DHCP and the TFTP; (2) the web service (http), which can provide the compressed root file system and deploying parameters; (3) BT-related services, which include the tracker (ocs-bttrack) and FSBT service (Ezio); and (4) the deployment manager (ocsmgrd), which aims to collect clients' information and generate the boot menu for the client's next netboot to enter local device booting; thus, the deployment will not be initiated again. To initiate the BT-based deployment, the deploying server first initiates these four types of services. Therefore, the bare-metal client machines to be deployed must boot from the network mechanism, which include either the PXE or the UEFI network booting. The remaining jobs will be unattended to—that is, the client will retrieve the root file system, torrent file, and the deploying parameters from the http service, starting the BT deployment by negotiating with the tracker and Ezio. Figure 8 demonstrates an example when the BT solution is used to deploy three machines. The FSBT files are located on the image server. These files can be distributed to all peers using BT solution. In the beginning, all peers download the metainfo file "sda1.torent" from the image server using the http protocol. By examining the information contained in the "sda1.torrent" file, a peer obtains the IP address of the torrent tracker. Then, the peer will query the torrent tracker via the TCP port 6969 to obtain the IP address of a peer or the image server that can provide the pieces of the FSBT files. After that, the peer uses Ezio [38] via the BT protocol to connect the image server or the peer with that IP address and the TCP port 6881, or the uTP based on UDP, to download the pieces of FSBT files. In the meantime, the peer also informs the torrent tracker which pieces have already been obtained by the peer itself. Once a peer receives the pieces of the FSBT files, these pieces will be written to the hard drive by the Ezio [38], and the pieces on the hard drive can then also serve as the source to be distributed by the Ezio [38] using BT protocol. When all of the deployments have been finished, the client will inform the deploying server, and the whole deployment is completed.

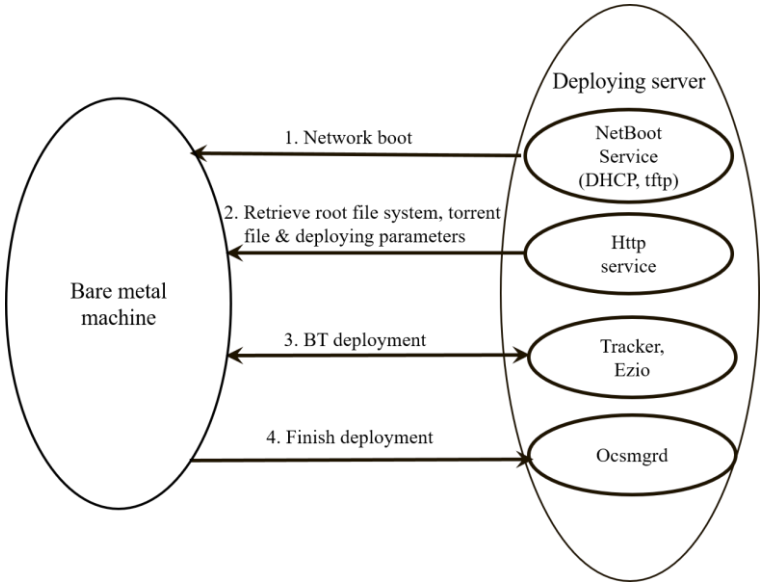


Figure 7. The schematic diagram about a massively deploying image to a bare-metal machine using the BT solution. DHCP: Dynamic Host Configuration Protocol.

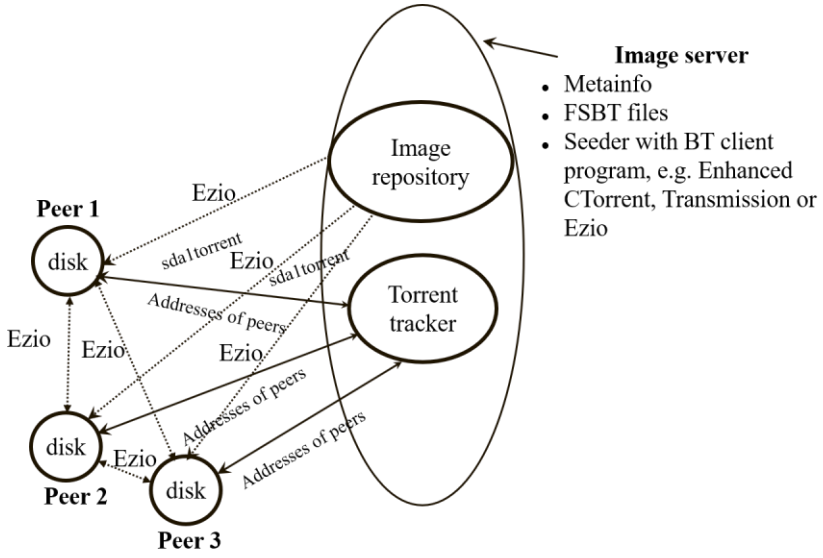


Figure 8. The BT solution for massive system deployment.

4. Experimental Process and Results

The section demonstrates the feasibility of Clonezilla live with the BT protocol for massive deployment by deploying Ubuntu Linux to massive machines in a computer classroom. The experiments deployed a Linux system to one to 32 clients using both the multicast solution previously developed by the authors [3] and the newly developed BT-based solution. The experimental environment consisted of the following machines and image:

- Network switch: the Cisco Catalyst 3560G switch with 48 gigabits ports was used as the network switch. The multicast function was enabled, and the spanning tree protocol was disabled to avoid the timeout of network booting in the client machines.
- Server: a Dell T1700 machine plays the role of a server. The central processing unit (CPU) is the 3.3 GHz Intel Xeon E3-1226 processor. The size of the Dynamic Random Access Memory (DRAM) is 16 gigabytes (GB). The size of the hard disk is one terabyte (TB).

- PC clients: Dell T1700 PCs with the same configuration as the one serving as the server were used as clients.
- The image of the Linux and applications: an Ubuntu Linux system with applications and data installed on a template PC occupying 50 GB of the hard disk. The files were saved by Clonezilla live, and the image was compressed using the parallel Zstandard (pzstd), which is a fast lossless compression algorithm [58]. The image size is deliberately designed to be larger than the RAM size of the PC client in the experiment to ensure the feasibility of the proposed system to solve the temporary storage shortage issue.

In the experiments, 32 PC clients were connected to the server using a Cisco Catalyst 3560G switch and the Category 5e network cables. The configuration of the experiments of massive deployment in large-scale computers is shown in Figure 9.

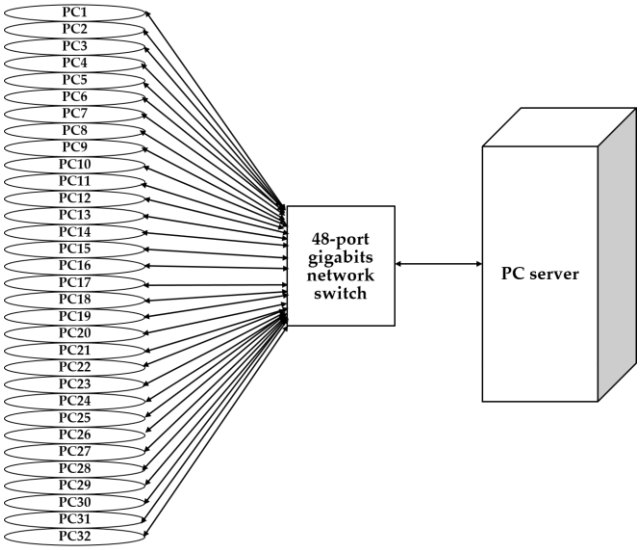


Figure 9. Configuration of experiments.

The image was put in the local hard drive of the server, and the server entered the lite-server mode of Clonezilla live, which can relay the DHCP requests from the PC clients to the existing DHCP service in the LAN. After choosing the image, the BT and multicast solutions were chosen for the massive deployment. The clients were booted from the PXE. Thirty-two experiments to deploy the image to one to 32 clients were conducted. When some of the clients were deployed, the remaining clients of the 32 machines that were not deployed were powered off. Table 2 summarizes the total time required to deploy the machines using the unicast, BT, and multicast solutions. The ratio of the total time required for massive deployment using the BT solution to the time required for the massive deployment by the multicast solution is also summarized in Table 2. For a perfect situation, we assume here that there is no overhead during the unicast deployment in sequence, so the time to deploy multiple machines is obtained by multiplying the time required for one machine by the number of client machine(s).

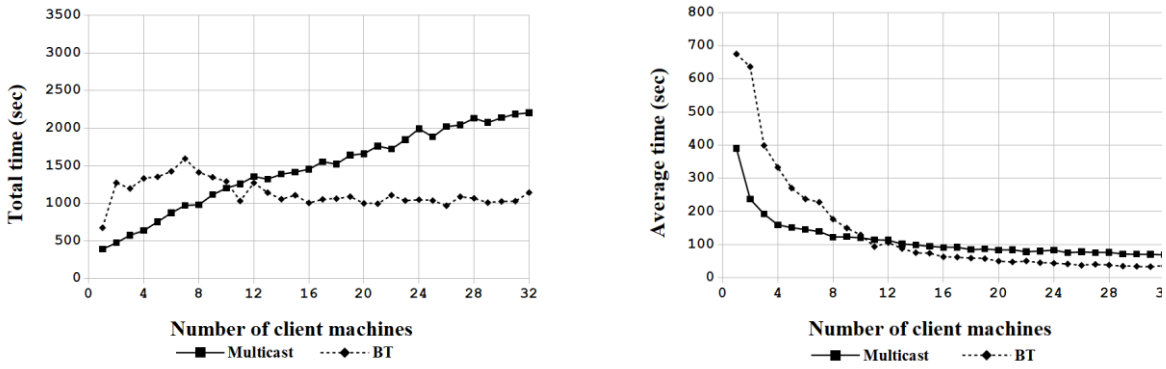
Here, "total time" means the time required to deploy the file system(s). Other time required for system booting, partition table creation, image file conversion, and informing the server regarding deployment results is not included. Figure 10 demonstrates the results of the massive deployments. Figure 10a,b demonstrate the total time and average time required by both the BT and multicast solutions to deploy the image, respectively. Based on the illustrations, when the number of clients to be deployed is smaller than 11, the total time required for system deployment was greater under the BT solution than the multicast solution. However, when the number of clients is equal to or larger than 11, the total deployment time required for the BT solution is less than the time required by the multicast solution. In the six experiments, all of the client machines that were deployed had been

verified as bootable and could enter the Ubuntu Linux. Hence, these six experiments verified the feasibility, efficiency, and scalability of the proposed BT solution.

Table 2. Total time required to deploy a 50-GB Ubuntu Linux system and applications using the unicast, multicast, and BT solutions. GB: gigabyte.

Number of Clients	t_{seq} (1)	t_{BT} (2)	$t_{multicast}$ (3)	Ratio ($t_{BT}/t_{multicast}$)
1	474	675	390	1.731
2	948	1273	474	2.686
3	1422	1197	576	2.078
4	1896	1331	638	2.086
5	2370	1352	754	1.793
6	2844	1425	872	1.634
7	3318	1594	973	1.638
8	3792	1412	980	1.441
9	4266	1347	1114	1.209
10	4740	1291	1202	1.074
11	5214	1031	1258	0.820
12	5688	1272	1356	0.938
13	6162	1142	1322	0.864
14	6636	1055	1387	0.761
15	7110	1108	1416	0.782
16	7584	1005	1454	0.691
17	8058	1053	1553	0.678
18	8532	1062	1522	0.698
19	9006	1089	1640	0.664
20	9480	1000	1660	0.602
21	9954	995	1762	0.565
22	10428	1108	1722	0.643
23	10902	1036	1846	0.561
24	11376	1048	1992	0.526
25	11850	1036	1883	0.550
26	12324	968	2020	0.479
27	12798	1088	2041	0.533
28	13272	1067	2131	0.501
29	13746	1009	2074	0.486
30	14220	1025	2138	0.479
31	14694	1029	2186	0.471
32	15168	1143	2203	0.519

Remarks: (1) The total time required by the unicast solution in sequence (secs); (2) the total time required by the BT solution (secs); and (3) the total time required by the multicast solution (secs).



(**a**) (**b**)

Figure 10. (**a**) Total time and (**b**) average time required for the massive deployment to various clients using the multicast and BT solutions.

Student's t-test and Levene's test were performed in order to verify our hypothesis, following the approach used by González-Briones et al. [84,85]. These two tests assessed the difference in means (total deployment time in seconds) and variances between the data obtained by using either the BT or the multicast solution. In the tests, the size of the samples was $n_1 = 32$, $n_2 = 32$, which was equivalent to the experimental cases. The level of α was set to 0.050. In addition, t is the t-test for equality of means, and F is Levene's test for equality of variances [86,87]. The established hypothesis H_0 stated that the mean time consumed by the massive deployment before the implementation of the BT solution is equal to the mean time consumed by the BT-based solution; H_1 stated that the average time consumption in the massive deployment is lower when the proposed BT solution is implemented. Hypothesis H_0 is rejected, whereas H_1 is proved to be true when the p-value is under 0.050 (significance value); therefore, the system consumes less time from the deployment by using the BT-based solution. As shown in Table 3, the percentage of use after implementing the BT solution was remarkably lower, as the *p*-value was much less than 0.050 for both Student's test and Levene's test. In conclusion, the proposed BT system greatly contributes to time savings in massive deployments.

Table 3. Results of Student's *t* test and Levene's test carried out to evaluate the difference of means and variances between the data obtained from BT and multicast solutions.

Solution	Mean	Stdr. Deviation	*t* Stat	*p* value (1-tailed)	F	*p* value
BT	1.13×10^3	1.74×10^2	2.860	3.75×10^{-3}	1.02×10^1	2.17×10^{-3}
Multicast	1.45×10^3	5.31×10^2				

To estimate the accuracy of performance enhancement accurately, the current set of samples must be partitioned into a training set and a separate test set. Here, the leave-one-out cross-validation (LOOCV) [88,89] method was performed, and the results are summarized in Table 4. The time-saving ratio value was obtained by the difference between the time required by the BT and the multicast solutions for the same number of clients. The difference was then divided by the time obtained from the multicast solution. According to the analytic results (refer to Table 4), the total time-saving ratio for 32 experiments was 81.760%. The LOOCV values were derived by leaving the observed time-saving ratio as the validation data, whereas the remaining observations were left as the training data. The difference between the total time-saving ratio and the observed time-saving ratio was then averaged (divided by 31 in this study). As indicated in Table 5, the results derived from the first 10 experiments are not consistent with the remaining 22 experiments. Hence, the accuracy is 22/32 = 68.750%. However, if we neglect the results derived from the first 10 experiments, the accuracy can reach 100%. From the aspect that the proposed P2P-based solution can resolve the scalability problem in large-scale deployment, the result is very reasonable. If more experiments are conducted, it is reasonable to predict that such high accuracy can be sustained. Possible reasons for the phenomenon will also be discussed further in Section 5.1.

Table 4. Speedups, speedup ratios, time-saving, and leave-one-out cross-validation for BT to multicast solution. LOOCV: leave-one-out cross-validation.

Number of Client(s)	SF_{BT}	SF_{MC}	$SR_{BT\text{-}MC}$	Time-Saving Ratio (%)	LOOCV (%)
1	0.702	1.215	0.578	−73.077	4.995
2	0.745	2.000	0.372	−168.565	8.075
3	1.188	2.469	0.481	−107.813	6.115
4	1.424	2.972	0.479	−108.621	6.141
5	1.753	3.143	0.558	−79.310	5.196
6	1.996	3.261	0.612	−63.417	4.683
7	2.082	3.410	0.610	−63.823	4.696
8	2.686	3.869	0.694	−44.082	4.059
9	3.167	3.829	0.827	−20.916	3.312
10	3.672	3.943	0.931	−7.404	2.876

Table 4. *Cont.*

Number of Client(s)	SF_{BT}	SF_{MC}	SR_{BT-MC}	Time-Saving Ratio (%)	LOOCV (%)
11	5.057	4.145	1.220	18.045	2.055
12	4.472	4.195	1.066	6.195	2.438
13	5.396	4.661	1.158	13.616	2.198
14	6.290	4.784	1.315	23.937	1.865
15	6.417	5.021	1.278	21.751	1.936
16	7.546	5.216	1.447	30.880	1.641
17	7.652	5.189	1.475	32.196	1.599
18	8.034	5.606	1.433	30.223	1.662
19	8.270	5.491	1.506	33.598	1.554
20	9.480	5.711	1.660	39.759	1.355
21	10.004	5.649	1.771	43.530	1.233
22	9.412	6.056	1.554	35.656	1.487
23	10.523	5.906	1.782	43.879	1.222
24	10.855	5.711	1.901	47.390	1.109
25	11.438	6.293	1.818	44.981	1.186
26	12.731	6.101	2.087	52.079	0.957
27	11.763	6.270	1.876	46.693	1.131
28	12.439	6.228	1.997	49.930	1.027
29	13.623	6.628	2.056	51.350	0.981
30	13.873	6.651	2.086	52.058	0.958
31	14.280	6.722	2.124	52.928	0.930
32	13.270	6.885	1.927	48.116	1.085

Remarks: (1) SF_{BT} means the speedup factors for mass deployment using the BT solution; (2) SF_{MC} means the speedup factors for deployment using the multicast solution; (3) SR_{BT-MC} means the speedup ratio of the BT solution to that of the multicast one.

5. Discussion

This section discusses the performance achieved by different solutions for massive deployment, and then compares differences between the proposed method and other studies. Finally, the limitations of this work are discussed.

5.1. Performance of Massive Deployment by the Proposed BT Solution

According to the authors' earlier definition [34], the speedup of the massive deployment for a large number of computers is the ratio of time required to deploy multiple computers in sequence versus the time required to deploy multiple computers by the BT or multicast solution. Figure 11 demonstrates the speedup measured in the experimental environments consisting of one to 32 clients. Table 4 further summarizes the speedup for the BT and multicast solutions. As the authors assumed no overhead for the ideal case in the massive deployments, the speedup increases linearly as the number of client machines increases. However, in the real world, the speedup becomes smaller as more computers are deployed using the BT or multicast solution. Compared to the multicast solution, the speedup of the BT solution increases as more computers are deployed (see Figure 10a and Table 4). As the number of clients increases, the speedup ratio of the BT-based solution to the multicast-based one also increases. When deploying 32 machines, which was the largest number in the experiment, the ratio hit 1.927, which means that the BT-based mass deployment solution proposed in this research can enhance the performance by about two times that of the traditional multicast solution in the experimental environment.

In Figure 10a, when there is only one client machine, the time to complete the deployment is the shortest in the BT-based solution. In this case, there is only one seeder (image repository server), so it is dedicated to serving the only client. However, when there are two or more client machines to be deployed, the available system resource and network bandwidth from the image repository server will be shared; hence, the performance decreases temporarily before the number of client machines can increase to a large enough number. Nevertheless, as the number of client machines grows large

enough, the merit of the BT solution is shown in Figure 10a; that is, the more peers that join the swarm, the more the access speed and availability increases, thereby contributing to the overall provisioning performance [90]. In addition, Figure 10a shows that the total time required by the BT solution is almost constant when the number of client machines is equal to or more than 10. In the BT-based solution, the available service capacity grows exponentially with time [91]. Hence, the total time required to complete the download tasks will be shortened. In the ideal case, assume that no bottlenecks exist in the CPUs, RAMs, or network(s). Meanwhile, no disk errors happen while reading or writing data. If the BT connections among all of the peers are in the topology of a complete tree-typed network [92,93] with n nodes, where the degree of each node is defined as c. Here, the degree of some specific node means that the number of connections of the node to other nodes. Assume that the child node does not send data to the parent node, and the nodes in the same level of the tree do not send data to each other; then, the total number of peers (n) can be formulated as [91]:

$$n = \sum_{i=0}^{l} c^i \tag{1}$$

where c^i is the capacity of service in the ith level of the tree, and l is the total number of levels in the tree. As only the leaf nodes in the tree cannot provide data in the BT-based solution, the total number of nodes in the tree that can provide the data is $n - c^l$. $n - c^l$ can be expressed as $(n-1)/c$. When n is large enough (e.g., 10), the total number of nodes in the tree that can provide data could be approximated as n/c.

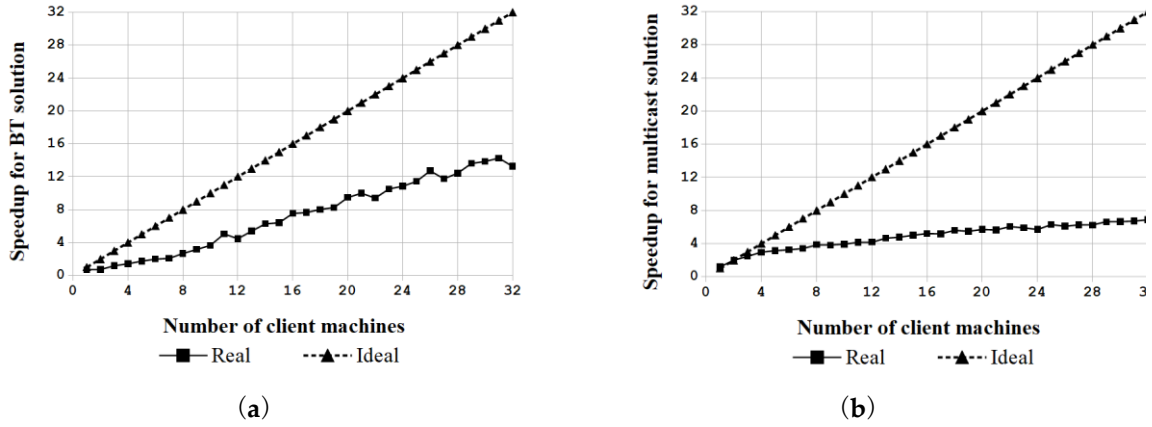
Let d be the data size to be transmitted to a peer. d_{total} is the total data to be transmitted in the BT-based solution. b is the network bandwidth that a peer can use, and b_{total} is the total bandwidth available in the BT-based solution. Thus, the total time for all of the peers in the BT solution to complete the download can be formulated as:

$$t_{total} = d_{total}/b_{total} = dn/(bn/c) = dc/b \tag{2}$$

Equation (2) formulates the total time required to complete the distribution of data size d to n peers in the BT environment when the available bandwidth for each peer is b. In the ideal case, which has no bottleneck or error in the hardware, and with some approximation, because the number of peers (n) will not influence the total time (dc/b) in Equation (2), the total download completion time is constant. This explains why the total time is almost constant when the number of client machines is more than 10.

Furthermore, the total download completion time dc/b can be approximated from the experimental results in Section 4. The mean value of the time to deploy 10 to 32 clients (refer to Table 2) using the BT-based solution is 1072.174 s. Figure 12 demonstrates that (1) the average deployment time for each configuration obtained from the experiments as well as (2) the division of t_{total} value (1072.174 s) by the number of client machines (i.e., $(dc/b)/n$). As shown in Figure 12, two curves match very well when the number of clients is equal to or larger than 10. These two curves also roughly match each other as the number of clients is less than 10, except for the case of one client. This result validates that Equation (2) can be used to estimate the total download completion time for the proposed BT solution.

As for the mismatches between the experimental results as well as Equation (2) in Figure 12, when the client number is less than 10, the major reason is due to the insufficient clients in the BT solution that cannot form a complete or good enough tree-typed network topology to optimize the network [92,93]. Therefore, some biases between the ideal case and the experimental results exist.



(**a**) (**b**)

Figure 11. The speedup factors achieved by (**a**) the proposed BT solution and (**b**) the traditional multicast solution.

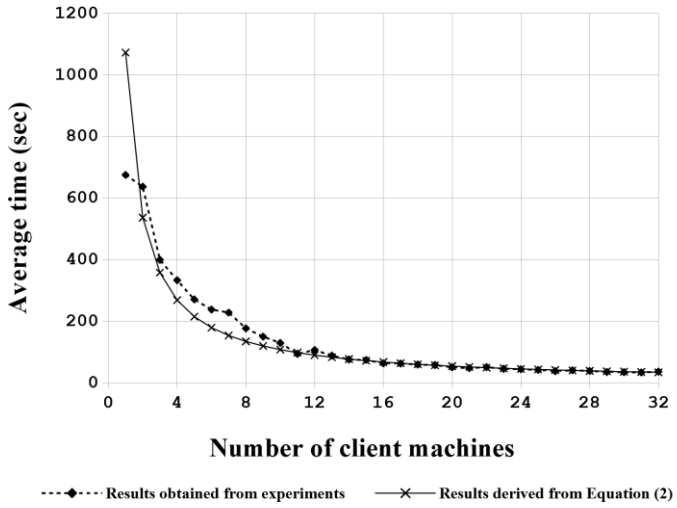


Figure 12. The average deployment time of each configuration obtained from the experimental results based on the proposed BT solution as well as the ideal deployment time derived from Equation (2).

Table 5 summarizes the pros and cons of the BT and multicast solutions in system deployment. For the pros, based on the experiment results, feedback from other users about using the proposed BT-based mass deployment software, and other studies [3,58], the BT-based solution is more reliable and flexible. When the number of clients to be deployed is large (for example, equal to or more than 11 clients in this work), the deployment performance is better than that of the multicast solution as well.

Although the BT-based solution has a lot of advantages, some cons exist. The quality barrier caused by the network switches and the requirement for more disk space for the image repository are significant disadvantages. For the quality barrier caused by the high loadings of the network switches, client(s) may fail to retrieve the required files for network booting, and the deployment manager, ocsmgrd, may fail to receive the signals from clients that have finished the deployment. Therefore, the quality of network switches, which have high data transmission rates and no tolerance for congestion, is required to ensure successful deployments. Another drawback of the proposed BT-based solution stems from the extra disk space that is required for the image repository. As the BT image must be prepared in the FSTB format, it will require more disk space in the repository.

In terms of the pros of the multicast solution, a lower bandwidth is required compared to the BT-based solution. Furthermore, no extra disk space is required for the image repository because the image does not have to be converted to the FSBT format. The multicast solution also has better performance with a smaller number of clients (e.g., fewer than 11 clients in this work).

The multicast solution has several disadvantages. First, the multicast protocol is unreliable for packet delivery due to the nature of the UDP protocol. Therefore, packets may be lost, because no handshaking mechanism is available in the UDP protocol, and the client must request the lost packets. When a client requests these packets, the remaining clients have to wait for the synchronization of all of the clients before receiving the next data. Thus, the deployment performance using the multicast solution is reduced. Therefore, the multicast-based solution is very difficult to scale up. In addition, the multicast function may be disabled in the network switch by the network system administrator for security reasons. Users may not have the authority to enable the multicast function. In this case, the multicast solution cannot be used.

Table 5. Comparisons of the BT and multicast solutions.

Solution	Pros	Cons
BT	More reliable. More flexible. Better performance in larger scale. Scalable.	Quality barrier of network switch is higher. More image repository disk space requirement.
Multicast	Low bandwidth requirement. Better performance in smaller scale. No extra disk space is required.	Packet loss issue. Function may be disabled in network switch. Not scalable.

Table 6 compares the work and the previous solutions for massive bare-metal provisioning. The multicast solution implemented in the previous work and the BT-based solution proposed and implemented in this work were shown to be feasible. From the aspect of efficiency, the BT solution has better efficiency than the multicast one. Based on the experimental results demonstrated in Section 4, the total time-saving ratio for 32 experiments was 81.760%. From this aspect, the total deployment time required by the BT-based solution is 18.240% of the time for the multicast solution. Thus, the efficiency of the BT-based solution can be 5.482 times (1/0.1824 = 5.482) that of the multicast ones. Considering the scalability issues, the efficiency enhancement can be much higher when the number of clients to be deployed is more than 32. In addition, from the experimental results in this study and previous works by other researchers [94,95], the BT-based solution was confirmed to have better reliability and scalability than the multicast solution.

Table 6. Summary of comparisons of characteristics of multicast and BT-based solutions in massive bare-metal provisioning.

Solution	Feasibility	Efficiency	Relability	Scalability
Multicast	Yes	x	Average	Average
BT	Yes	5.482x	Good	Good

5.2. Comparisons with Other BT Bare-Metal Provisioning Solutions

Table 7 compares the Clonezilla live and other bare-metal provisioning solutions that also provide the BT-based system. The Clonezilla live that was developed in this work is released under the open source, free software GNU General Public License (GPL) [33], while Kadeploy [57] is under another open source, free software license CEA CNRS INRIA Logiciel Libre (CeCILL) [96]. The software license for ESIR is unknown, because it is not mentioned in the research article [56]; nor could an online resource be found to download the software. The same situation regarding a software license was seen in the research work of Anton and Norbert [58]. As shown in Table 7, Clonezilla live is superior to other solutions in supporting more solutions, including booting from a live system and the ability to deploy a system using unicast, multicast, and BT solutions. Based on the comparison results, adding the BT solution to Clonezilla live in this research has extended its features. These aspects show that the newly

added BT solution in this research has made Clonezilla live a better choice for massive deployment in the computer classroom and HPC clusters.

Table 7. Comparisons of bare-metal provisioning solutions that include the BT solution. CeCILL: CEA CNRS INRIA Logiciel Libre, ESIR: efficiency, scalability, independence, and reliability; GPL: General Public License, OS: operation system.

Program	Software License	Case(*)				Notes
		1	2	3	4	
Clonezilla live	GPL	Yes	Yes	Yes	Yes	Open architecture, supports most of the mainstream OSs deployment.
ESIR [56]	N/A	U(**)	U(**)	U(**)	Yes	Dynamic module loading technique makes hardware independent deployment.
Kadeploy [57]	CeCILL	Yes	Yes	No	Yes	Provides a set of tools for cloning, configuring (post installation), and managing cluster nodes.
Work by Anton and Norbert [58]	N/A	U(**)	U(**)	No	Yes	Proposed system architecture utilizes P2P communication between nodes that leads to increasing throughput.

* 1. Boot from a live system (run without installation); 2. Support unicast solution; 3. Support multicast solution; 4. Support BT solution. ** Not mentioned in the paper; nor was a source code or binary program available for verification.

5.3. Limitations

Despite the successful implementation of the BT solution in our research, some limitations need to be addressed in this section. The limitations include the format and size of the image file for BT and piece transmissions in the network. The issues will be discussed in the following sub-sections.

5.3.1. File Format and Size of the Image for BT

As described in Section 3, the file format of the image for the BT solution is the binary data file. The file is not compressed, because the Ezio and P2P downloaders cannot split the compressed file into pieces, and still can map the raw data on the partition or disk for the BT solution, and then share the file directly with other peers. Hence, the file size of the FSBT image is much larger than the file size of the original Clonezilla image, which is compressed. More space in the image repository will be required if the BT-based solution is used to deploy the system. Therefore, a universal format of images for BT, unicast, broadcast, and multicast solutions will save the time required to convert the image. The required disk space for the image repository will also be significantly reduced, as will the complexity to maintain the image repository.

5.3.2. Piece Transmissions in the Network

When the pieces of the BT data file are shared with all peers, they are not compressed before sending. Hence, the receiver does not decompress the copied pieces. In traditional BT-based file-sharing activities, the files to be shared have been compressed. However, as described in Section 5.3.1, uncompressed image files are required by the FSBT mechanism. As the burden of the network can be relieved if the data transmission in the network can be compressed before sending, the compression function for the pieces of the BT data file is very helpful in the BT-based solution. For the time being, Ezio and other BT client programs (e.g., Enhanced CTorrent or Transmission) only communicate with each other in an uncompressed way. The Libtorrent library does not provide any compression function either. By incorporating the compression function, the network traffic can be greatly reduced.

5.4. Future Research Possibilities

The procedures of the BT-based mass deployment introduced in this study are based on the partition and logical volume (LV). For example, if there are two or more partitions in a hard drive, the first partition must be deployed first; then, the deployment of the second partition will be initiated. In the implementation, the modified tracker for the BT service, the ocs-bttrack, uses the parameter

"–tracker_timeout" or "–tracker_max_completed" to control whether the deployment of the first partition for all of the client machines has been completed or not. However, such a procedure has limited the flexibility of BT-based deployment, because the simultaneous read/write features of the BT-based solution for all of the partitions and LVs have not been fully utilized. For the multicast-based solution, all of the client machines should be synchronized when the used blocks are written to partitions. However, such limitations do not exist in the BT solution. Furthermore, the BT solution should allow a client to join the deployment at any time before all of the BT-related services have been terminated. In general, the simultaneous read/write feature and the inclusion of a client at any time in the deployment process can be included in the future. The flexibility of the BT-based deployment solution can greatly be improved.

6. Conclusions

This paper proposed a novel P2P-based solution for massive deployment. An open source program was implemented accordingly so that the traditional limitations of BT-based bare-metal provisioning in massive deployment could be lifted. The traditional limitations caused by the minimum RAM size that is required to accommodate the whole P2P data file or the requirement for an extra storage on the designated deploying machine to store the whole P2P data file could be lifted. To the best of the authors' knowledge, this is the first open source system deployment program that can use the BT protocol to deploy large system images without requiring RAM size or an extra temporary storage to be equal to or larger than the images. In addition, this newly added BT-based deployment feature is fully compatible with the authors' earlier version of the mass deployment program. Therefore, all of the merits described in our previous works, including open design, full automation, supports for customization options, and the provision of a live system that is ready to be used, can be inherited in this work. A comparison between our research results with other programs and research results demonstrated the advantages and superior performance of our works. The feasibility of the proposed work was verified using experiments to deploy a Linux system as well as applications to one to 32 machines. The superior performance was further demonstrated based on comparisons between the proposed BT-based solution and the traditional multicast solution. The proposed BT solution can be about two times faster than the multicast solution when deploying 32 machines. In general, the comparisons of features and performance with other utilities have proved and demonstrated the superior efficiency of the proposed massive deployment solution.

In the future, the BT parameters for massive deployments, which include the piece size and maximum number of peers, can be further optimized to improve performance. Furthermore, adding post-deployment tuning functions to the massive deployment system may improve the massive deployment performance. In addition, a universal image format for BT, unicast, broadcast, and multicast solutions can further enhance the compatibility and reduce the complexity of image repository maintenance. Meanwhile, the addition of compression features to the piece transmission of the BT solution can reduce the network traffic, and hence enhance the network performance. In addition, to utilize the BT solution effectively, a better deployment flow could be developed in the future so that the machine to be deployed can join and quit the massive system deployment pool in a more flexible way. Moreover, extending the mechanism developed in this work to the OpenStack Ironic [97], an OpenStack program that provisions bare-metal machines, would provide another bare-metal provisioning solution for the cloud-based infrastructure. Finally, to make the massive deployment on a larger scale, developing a tracker-less mechanism to use DHT while sending data to the desired machines only is a very good topic for future exploration.

Author Contributions: Y.-C.H. proposed the idea of Ezio and developed the Ezio program with C.-H.Y. S.-K.H. and C.-C.H. advised the development of Ezio. Y.-C.T. developed the mechanism in Partclone to convert an existing image to the format that the BT solution requires, and S.J.H.S. designed and added the BT solution to the program Clonezilla. Y.-C.H., C.-H.Y., C.-K.S., Y.-C.T., and S.J.H.S. conceived and designed the experiments and analyzed the data; S.J.H.S. wrote and edited the paper; and J.-N.J. instructed the goal of this study. C.-Y.H. rewrote the whole article and revised the work.

Funding: This research was funded by Ministry of Science and Technology, Taiwan with grant number MOST 107-2634-F-492-001.

Acknowledgments: The authors would like to thank Ping-Chun Huang for inspiring the idea to develop Ezio and Yu-Sung Wu for research advice. This project was supported by the National Center for High-Performance Computing in Taiwan. The authors thank those Clonezilla users who provided feedback about the program.

Conflicts of Interest: The authors declare no conflict of interest.

References

1. Papadopoulos, P.M.; Katz, M.J.; Bruno, G. NPACI Rocks: Tools and techniques for easily deploying manageable linux clusters. *Concurr. Comput. Pract. Exp.* **2003**, *15*, 707–725. [CrossRef]
2. Mirielli, E.; Webster, L.; Lynn, M. Developing a multi-boot computer environment and preparing for deployment to multiple workstations using Symantec Ghost: A cookbook approach. *J. Comput. Sci. Coll.* **2005**, *20*, 29–36.
3. Shiau, S.J.; Sun, C.-K.; Tsai, Y.-C.; Juang, J.-N.; Huang, C.-Y. The Design and Implementation of a Novel Open Source Massive Deployment System. *Appl. Sci.* **2018**, *8*, 965. [CrossRef]
4. Chandrasekar, A.; Gibson, G. *A Comparative Study of Baremetal Provisioning Frameworks*; Technical Report CMU-PDL-14-109; Parallel Data Laboratory, Carnegie Mellon University: Pittsburgh, PA, USA, 2014.
5. Hirway, M. *Hybrid Cloud for Developers: Develop and Deploy Cost-Effective Applications on the AWS and OpenStack Platforms with Ease*; Packt Publishing: Birmingham, UK, 2018.
6. Kickstart Document. Available online: https://docs.fedoraproject.org/en-US/Fedora/html/Installation_Guide/chap-kickstart-installations.html (accessed on 15 October 2018).
7. FAI Project. Available online: https://fai-project.org (accessed on 15 October 2018).
8. Aswani, K.; Anala, M.; Rathinam, S. Bare metal Cloud Builder. *Imp. J. Interdiscip. Res.* **2016**, *2*, 1844–1851.
9. FSArchiver—File System Archiver for Linux. Available online: http://www.fsarchiver.org (accessed on 3 September 2017).
10. Cougias, D.J.; Heiberger, E.L.; Koop, K. *The Backup Book: Disaster Recovery from Desktop to Data Center*; Network Frontiers: Lecanto, FL, USA, 2003.
11. Petersen, R. *Ubuntu 18.04 LTS Server: Administration and Reference*; CreateSpace Independent Publishing Platform: Scotts Valley, CA, USA, 2018.
12. Cornec, B. Mondo Rescue: A GPL disaster recovery solution. *Proc. Linux Symp.* **2008**, *1*, 77–84.
13. Kumar, R.; Gupta, N.; Charu, S.; Jain, K.; Jangir, S.K. Open source solution for cloud computing platform using OpenStack. *Int. J. Comput. Sci. Mob. Comput.* **2014**, *3*, 89–98.
14. Storix System Backup Administrator. Available online: https://www.storix.com (accessed on 3 September 2017).
15. Partimage Software. Available online: http://www.partimage.org (accessed on 3 September 2017).
16. Sanguino, T.M.; de Viana, I.F.; García, D.L.; Ancos, E.C. OpenGnSys: A novel system toward centralized deployment and management of computer laboratories. *Comput. Educ.* **2014**, *75*, 30–43. [CrossRef]
17. Williamson, B. *Developing IP Multicast Networks*; Cisco Press: Indianapolis, IN, USA, 2000; Volume 1.
18. Manini, D.; Gaeta, R.; Sereno, M. Performance modeling of P2P file sharing applications. In Proceedings of the Workshop on Techniques, Methodologies and Tools for Performance Evaluation of Complex Systems (FIRB-PERF'05), Torino, Italy, 19 September 2005; pp. 34–43.
19. Schollmeier, R. A definition of peer-to-peer networking for the classification of peer-to-peer architectures and applications. In Proceedings of the First International Conference on Peer-to-Peer Computing, Linkoping, Sweden, 27–29 August 2001; pp. 101–102.
20. Rossi, D.; Testa, C.; Valenti, S.; Muscariello, L. LEDBAT: The new BitTorrent congestion control protocol. In Proceedings of the 19th International Conference on Computer Communications and Networks (ICCCN), ETH Zurich, Switzerland, 2–5 August 2010; pp. 1–6.
21. Saroiu, S.; Gummadi, P.K.; Gribble, S.D. Measurement study of peer-to-peer file sharing systems. In Proceedings of the Multimedia Computing and Networking, San Jose, CA, USA, 19–25 January 2002; pp. 156–171.
22. Le Fessant, F.; Handurukande, S.; Kermarrec, A.-M.; Massoulié, L. Clustering in peer-to-peer file sharing workloads. In Proceedings of the International Workshop on Peer-to-Peer Systems, La Jolla, CA, USA, 26–27 February 2004; pp. 217–226.

23. Shah, R.; Narmawala, Z. Mobile torrent: Peer-to-peer file sharing in Android devices. *Int. J. Comput. Sci. Commun.* **2016**, *7*, 20–34.
24. Mastorakis, S.; Afanasyev, A.; Yu, Y.; Zhang, L. nTorrent: Peer-to-Peer File Sharing in Named Data Networking. In Proceedings of the 26th International Conference on Computer Communications and Networks (ICCCN), Vancouver, BC, Canada, 31 July–3 August 2017.
25. Liu, Y.; Guo, Y.; Liang, C. A survey on peer-to-peer video streaming systems. *Peer-to-Peer Netw. Appl.* **2008**, *1*, 18–28. [CrossRef]
26. Guha, S.; Daswani, N. *An Experimental Study of the Skype Peer-to-Peer Voip System*; Cornell University: Ithaca, NY, USA, 2005.
27. Baset, S.A.; Gupta, G.; Schulzrinne, H. Openvoip: An open peer-to-peer voip and im system. In Proceedings of the ACM SIGCOMM, Seattle, WA, USA, 17–22 August 2008.
28. Androutsellis-Theotokis, S.; Spinellis, D. A survey of peer-to-peer content distribution technologies. *ACM Comput. Surv.* **2004**, *36*, 335–371. [CrossRef]
29. Antonopoulos, A.M. *Mastering Bitcoin: Unlocking Digital Cryptocurrencies*; O'Reilly Media, Inc.: Sebastopol, CA, USA, 2014.
30. Wood, G. Ethereum: A secure decentralised generalised transaction ledger. *Ethereum Proj. Yellow Pap.* **2014**, *151*, 1–32.
31. Nakamoto, S. Bitcoin: A Peer-to-Peer Electronic Cash System. Available online: https://bitcoin.org/bitcoin.pdf (accessed on 15 October 2018).
32. Feller, J.; Fitzgerald, B. *Understanding Open Source Software Development*; Addison-Wesley: London, UK, 2002.
33. Stallman, R. *Free Software, Free Society: Selected Essays of Richard M. Stallman*; Lulu Press: Morrisville, NC, USA, 2002.
34. Ren, S.; Tan, E.; Luo, T.; Chen, S.; Guo, L.; Zhang, X. TopBT: A topology-aware and infrastructure-independent bittorrent client. In Proceedings of the IEEE INFOCOM, San Diego, CA, USA, 14–19 March 2010; pp. 1–9.
35. Rosen, L. *Open Source Licensing: Software Freedom and Intellectual Property Law*; Prentice Hall PTR: Upper Saddle River, NJ, USA, 2004.
36. The qBittorrent Project. Available online: https://www.qbittorrent.org/ (accessed on 12 October 2018).
37. Wang, H.; Wang, C. Open source software adoption: A status report. *IEEE Softw.* **2001**, *18*, 90–95. [CrossRef]
38. Ezio Project. Available online: https://github.com/tjjh89017/ezio (accessed on 18 August 2018).
39. Lee, K.-M.; Teng, W.-G.; Wu, J.-N.; Huang, K.-M.; Ko, Y.-H.; Hou, T.-W. Multicast and customized deployment of large-scale operating systems. *Autom. Softw. Eng.* **2014**, *21*, 443–460. [CrossRef]
40. Shojafar, M.; Abawajy, J.H.; Delkhah, Z.; Ahmadi, A.; Pooranian, Z.; Abraham, A. An efficient and distributed file search in unstructured peer-to-peer networks. *Peer-to-Peer Netw. Appl.* **2015**, *8*, 120–136. [CrossRef]
41. Neumann, C.; Roca, V.; Walsh, R. Large scale content distribution protocols. *ACM SIGCOMM Comput. Commun. Rev.* **2005**, *35*, 85–92. [CrossRef]
42. Grönvall, B.; Marsh, I.; Pink, S. A multicast-based distributed file system for the internet. In Proceedings of the 7th Workshop on ACM SIGOPS European Workshop: Systems Support for Worldwide Applications, Connemara, Ireland, 9–11 September 1996; pp. 95–102.
43. Zhang, L.; Qin, B.; Wu, Q.; Zhang, F. Efficient many-to-one authentication with certificateless aggregate signatures. *Comput. Netw.* **2010**, *54*, 2482–2491. [CrossRef]
44. Cohen, B. Incentives build robustness in BitTorrent. In Proceedings of the Workshop on Economics of Peer-to-Peer Systems, Berkeley, California, USA, 5–6 June 2003; pp. 68–72.
45. Heckmann, O.; Bock, A.; Mauthe, A.; Steinmetz, R.; KOM, M.K. The eDonkey File-Sharing Network. *GI Jahrestag.* **2004**, *51*, 224–228.
46. Wierzbicki, A.; Leibowitz, N.; Ripeanu, M.; Wozniak, R. Cache replacement policies revisited: The case of P2P traffic. In Proceedings of the IEEE International Symposium on Cluster Computing and the Grid, Chicago, IL, USA, 19–22 April 2004; pp. 182–189.
47. Ripeanu, M. Peer-to-peer architecture case study: Gnutella network. In Proceedings of the First International Conference on the Peer-to-Peer Computing, Linkoping, Sweden, 27–29 August 2001; pp. 99–100.
48. Clarke, I.; Sandberg, O.; Wiley, B.; Hong, T.W. Freenet: A distributed anonymous information storage and retrieval system. In Proceedings of the Designing Privacy Enhancing Technologies, Berkeley, CA, USA, 25–26 July 2000; pp. 46–66.
49. Steinmetz, R.; Wehrle, K. *Peer-to-Peer Systems and Applications*; Springer: Berlin, Germany, 2005; Volume 3485.

50. Legout, A.; Urvoy-Keller, G.; Michiardi, P. Rarest first and choke algorithms are enough. In Proceedings of the 6th ACM SIGCOMM Conference on Internet Measurement, Rio de Janeriro, Brazil, 25–27 October 2006; pp. 203–216.
51. Dinger, J.; Waldhorst, O.P. Decentralized bootstrapping of P2P systems: A practical view. In Proceedings of the International Conference on Research in Networking, Aachen, Germany, 11–15 May 2009; pp. 703–715.
52. Dosanjh, M.G.; Bridges, P.G.; Kelly, S.M.; Laros, J.H. A peer-to-peer architecture for supporting dynamic shared libraries in large-scale systems. In Proceedings of the 41st International Conference on Parallel Processing Workshops (ICPPW), Pittsburgh, PA, USA, 10–13 September 2012; pp. 55–61.
53. García, Á.L.; del Castillo, E.F. Efficient image deployment in cloud environments. *J. Netw. Comput. Appl.* **2016**, *63*, 140–149. [CrossRef]
54. Chen, Z.; Zhao, Y.; Miao, X.; Chen, Y.; Wang, Q. Rapid provisioning of cloud infrastructure leveraging peer-to-peer networks. In Proceedings of the 29th IEEE International Conference on Distributed Computing Systems Workshops, Montreal, QC, Canada, 22–26 June 2009; pp. 324–329.
55. O'Donnell, C.M. Using BitTorrent to distribute virtual machine images for classes. In Proceedings of the 36th annual ACM SIGUCCS Fall Conference: Moving Mountains, Blazing Trails, Portland, OR, USA, 19–22 October 2008; pp. 287–290.
56. Xue, Z.; Dong, X.; Li, J.; Tian, H. ESIR: A Deployment System for Large-Scale Server Cluster. In Proceedings of the Seventh International Conference on Grid and Cooperative Computing, Shenzhen, China, 24–26 October 2008; pp. 563–569.
57. Jeanvoine, E.; Sarzyniec, L.; Nussbaum, L. Kadeploy3: Efficient and scalable operating system provisioning for clusters. *USENIX Login* **2013**, *38*, 38–44.
58. Anton, B.; Norbert, A. Peer to Peer System Deployment. *Acta Electrotech. Inform.* **2016**, *16*, 11–14.
59. Shestakov, A.; Arefiev, A. Patch File to Allow User to Provision Image Using Bittorrent Protocol in OpenStack. Available online: https://review.openstack.org/#/c/311091/ (accessed on 21 December 2018).
60. Cut Ironic Provisioning Time Using Torrents. Available online: https://www.mirantis.com/blog/cut-ironic-provisioning-time-using-torrents/ (accessed on 21 December 2018).
61. Abreu, R.M.; Froufe, H.J.; Queiroz, M.J.R.; Ferreira, I.C. MOLA: A bootable, self-configuring system for virtual screening using AutoDock4/Vina on computer clusters. *J. Cheminform.* **2010**, *2*, 10. [CrossRef] [PubMed]
62. Zhang, J.; Wang, L. An integrated open forensic environment for digital evidence investigation. *Wuhan Univ. J. Nat. Sci.* **2012**, *17*, 511–515. [CrossRef]
63. Books, L. *Network Booting: Preboot Execution Environment, Bootstrap Protocol, Netboot, Gpxe, Remote Initial Program Load*; ACM: New York, NY, USA, 2010.
64. Droms, R. Automated configuration of TCP/IP with DHCP. *IEEE Internet Comput.* **1999**, *3*, 45–53. [CrossRef]
65. Debian Live Systems Manual. Available online: https://live-team.pages.debian.net/live-manual/html/live-manual/the-basics.en.html (accessed on 3 November 2018).
66. Debian Live Image. Available online: https://www.debian.org/blends/hamradio/get/live (accessed on 22 October 2018).
67. CentOS Linux ISO Images. Available online: https://wiki.centos.org/Download (accessed on 22 October 2018).
68. Ubuntu Live. Available online: https://tutorials.ubuntu.com/tutorial/try-ubuntu-before-you-install (accessed on 22 October 2018).
69. Windows PE (WinPE). Available online: https://docs.microsoft.com/zh-tw/windows-hardware/manufacture/desktop/winpe-intro (accessed on 22 October 2018).
70. Nyström, M.; Nicholes, M.; Zimmer, V.J. UEFI Networking and Pre-Os Security. *Intel Technol. J.* **2011**, *15*, 80–100.
71. Bricker, G. Unified extensible firmware interface (UEFI) and secure boot: Promise and pitfalls. *J. Comput. Sci. Coll.* **2013**, *29*, 60–63.
72. Takano, Y.; Ando, R.; Takahashi, T.; Uda, S.; Inoue, T. A measurement study of open resolvers and DNS server version. In Proceedings of the Internet Conference, Tokyo, Japan, 24–25 October 2013.
73. Patch File about Get Bootstrap Info from Proxy Offer Packet. Available online: https://lists.gnu.org/archive/html/grub-devel/2016-04/msg00051.html (accessed on 22 October 2018).
74. Rhodes, C.; Bettany, A. An Introduction to Windows Installation Methodologies and Tools. In *Windows Installation and Update Troubleshooting*; Springer: Berlin, Germany, 2016; pp. 1–27.

75. DRBL-Winroll Project. Available online: https://drbl-winroll.org/ (accessed on 22 October 2018).
76. Deaconescu, R.; Rughinis, R.; Tapus, N. A bittorrent performance evaluation framework. In Proceedings of the Fifth International Conference on Networking and Services, Valencia, Spain, 20–25 April 2009; pp. 354–358.
77. Sirivianos, M.; Park, J.H.; Chen, R.; Yang, X. Free-riding in BitTorrent Networks with the Large View Exploit. In Proceedings of the 6th International workshop on Peer-To-Peer Systems (IPTPS), Bellevue, WA, USA, 26–27 February 2007.
78. LFTP Project. Available online: http://lftp.yar.ru/ (accessed on 12 October 2018).
79. The rTorrent BitTorrent Client. Available online: https://github.com/rakshasa/rtorrent (accessed on 12 October 2018).
80. Zeilemaker, N.; Pouwelse, J. 100 Million DHT replies. In Proceedings of the 14-th IEEE International Conference on Peer-to-Peer Computing (P2P), London, UK, 9–11 September 2014; pp. 1–4.
81. Libtorrent Project. Available online: https://www.libtorrent.org/ (accessed on 6 October 2018).
82. Carrier, B. Defining digital forensic examination and analysis tools using abstraction layers. *Int. J. Digit. Evid.* **2003**, *1*, 1–12.
83. Ocs-Bttrack Source Code. Available online: https://gitlab.com/stevenshiau/ocs-bttrack (accessed on 18 August 2018).
84. González-Briones, A.; Chamoso, P.; Yoe, H.; Corchado, J.M. GreenVMAS: Virtual organization based platform for heating greenhouses using waste energy from power plants. *Sensors* **2018**, *18*, 861. [CrossRef]
85. González-Briones, A.; Prieto, J.; De La Prieta, F.; Herrera-Viedma, E.; Corchado, J.M. Energy optimization using a case-based reasoning strategy. *Sensors* **2018**, *18*, 865. [CrossRef]
86. Schultz, B.B. Levene's test for relative variation. *Syst. Zool.* **1985**, *34*, 449–456. [CrossRef]
87. Sergio, A.; Carvalho, S.; Marco, R. On the Use of Compact Approaches in Evolution Strategies. *Adv. Distrib. Comput. Artif. Intell. J.* **2014**, *3*, 13–23.
88. Kohavi, R. A study of cross-validation and bootstrap for accuracy estimation and model selection. In Proceedings of the International Joint Conference on Artificial Intelligence (IJCAI), Montreal, QC, Canada, 20–25 August 1995; pp. 1137–1145.
89. Mikshowsky, A.A.; Gianola, D.; Weigel, K.A. Assessing genomic prediction accuracy for Holstein sires using bootstrap aggregation sampling and leave-one-out cross validation. *J. Dairy Sci.* **2017**, *100*, 453–464. [CrossRef] [PubMed]
90. Liu, X.; Guo, Z.; Wang, X.; Chen, F.; Lian, X.; Tang, J.; Wu, M.; Kaashoek, M.F.; Zhang, Z. D3S: Debugging deployed distributed systems. In Proceedings of the 5th USENIX Symposium on Networked Systems Design and Implementation, San Francisco, CA, USA, 16–18 April 2008.
91. Legout, A. *Peer-to-Peer Applications: From BitTorrent to Privacy*; INRIA: Paris, France, 2010.
92. Magharei, N.; Rejaie, R.; Guo, Y. Mesh or multiple-tree: A comparative study of live p2p streaming approaches. In Proceedings of the 26th IEEE International Conference on Computer Communications, Anchorage, Alaska, USA, 6–12 May 2007; pp. 1424–1432.
93. Small, T.; Li, B.; Liang, B. Outreach: Peer-to-peer topology construction towards minimized server bandwidth costs. *IEEE J. Sel. Areas Commun.* **2007**, *25*, 35–45. [CrossRef]
94. Silva, T.N.D.M.D. Bitocast: A Hybrid BitTorrent and IP Multicast Content Distribution Solution. Ph.D. Thesis, Universidade NOVA de Lisboa, Lisboa, Portugal, 2009.
95. Agrawal, P.; Khandelwal, H.; Ghosh, R.K. MTorrent: A multicast enabled BitTorrent protocol. In Proceedings of the Second International Conference on Communication Systems and Networks (COMSNETS), Bangalore, India, 5–9 January 2010; pp. 1–10.
96. Cecill and Free Software. Available online: http://www.cecill.info (accessed on 15 October 2018).
97. Lima, S.; Rocha, Á.; Roque, L. An overview of OpenStack architecture: A message queuing services node. *Cluster Comput.* **2017**, 1–12. [CrossRef]