BitNBD: BitTorrent-Based Network Block Device for Provisioning Virtual Machines in IaaS Clouds

Yong-Ju LEE†, Member, Hag-Young KIM†, Nonmember, and Cheol-Hoon LEE††, Member

SUMMARY Infrastructure-as-a-Service (IaaS) cloud computing is emerging as a viable alternative to the acquisition and management of physical resources. The new main feature of IaaS cloud computing is the virtual machine (VM) technology which improves the flexibility of resource management. VMs use virtual machine images that are preconfigured and ready to run. Typically, VM image management uses local file copy and distribution via a network file system (NFS). This potentially means that a more efficient method can be used for VM image distribution. For efficient VM image management, we have designed and implemented a BitTorrent-based network block device (namely, BitNBD) for provisioning VM images in IaaS clouds. The BitNBD mainly provides a ‘split read/write mechanism’ to deal with concurrent VM instances where the same pieces of a VM are shared. With respect to the legacy BitTorrent protocol, the BitNBD enhances the piece picker policy and energy-saving mode. It is very effective in minimizing VM startup delays and providing a hibernating capability.

key words: BitTorrent, network block device, IaaS clouds, cloud computing

1. Introduction

Cloud computing is the emergent technology that promises on-demand, dynamic and easily accessible computing power. Dynamic provisioning of computing resources through the use of Operating System (OS) and Virtual Machine (VM) technologies can be considered. Rapid OS deployment has always been a challenging task in enterprise data center environments. In particular, deploying hundreds of thousands of systems is necessary to automate the installation process. In large-scale data centers, manual OS installation is not practical due to its huge time/labor cost. With the advent of virtualization technologies like Xen, VMware, and Kernel-based Virtual Machine KVM, data centers have been going through a revolution in design. Virtualization enables multiple operating systems and applications to run concurrently on a single server without interfering with each other. Today, virtualization turns general clusters into an Infrastructure-as-a-Service (IaaS) cloud. In terms of VM image management, VM images are first cloned from a base image. In the cloning process, a copy command is executed on a raw image file. The cloned images are then copied using a secure shell (SSH) or shared via a network file system (NFS) to the physical nodes. The time taken to provision an instance is not negligible. Another method for cloning VMs is a copy-on-write (CoW) mode that only writes modified blocks to a disk. CoW is designed to save disk space and allow fast clones, but will slightly slow down normal disk performance.

In this paper, we propose a BitTorrent-based network block device (namely, BitNBD) for VM image management. It contains an efficient VM image sharing mechanism inherited from a legacy BitTorrent protocol and certain protocol modifications in terms of reduction of VM startup delay and energy-saving capability. To the best of our knowledge, this is the first work to specifically look at, and offer, a BitTorrent-based network block device focusing on VM file staging.

The rest of the paper is organized as follows. Section 2 investigates related work. Section 3 presents our BitTorrent-based network block device for provisioning VMs and providing storage-savings and energy-efficient file sharing through BitTorrent enhancement. In Sect. 4, we apply these formulas to the case of an experimental scenario and discuss the obtained results. Section 5 concludes the paper.

2. Related Work

With the recent growth and spread of cloud computing, there are many VM-based cloud management platforms. Table 1 compares the main features of open source IaaS Cloud solutions such as Eucalyptus [1], Open Nebula [2], Nimbus [3], ASPEN [4], and Tashi [5].

As shown in Table 1, image management of an open source IaaS cloud solution uses a naive method (e.g., local file copy and distribution via NFS). This is a common way to deploy VM image management. This potentially means that a more efficient method can be used for VM image distribution. Therefore, we apply the BitTorrent protocol to distribute virtual machines (VMs) in IaaS clouds.

In terms of VM transfer protocols, Smith et al. [6] investigates unicast, multicast, binary tree, and BitTorrent-based methods for distributing VM images to physical machines in a cloud computing environment. However, it contains only the effect of distributing images in terms of various transport protocols. VM images cannot be used until they are copied entirely.

In terms of storage-saving issues, deduplication is an effective way to reduce the total storage. Policromiades et al. [7] revealed a more important difference in duplicated data detected using the content-defined chunking method. Storage systems may exploit data deduplication patterns...
Table 1: Infrastructure-as-a-service clouds.

| VM          | Image management                                      |
|-------------|-------------------------------------------------------|
| Eucalyptus  | Xen, KVM: Local file copy, distribute images via SOAP |
| Open Nebula | Xen, KVM: Local file copy, distribute images via ssh or NFS |
| Nimbus      | Xen: File copy, unknown distribution method           |
| ASPEN       | Xen: Copy-on-write, snapshots, distribute over NFS    |
| Tashi       | KVM, Xen: Distribute over NFS                         |

to optimize the use of storage space and bandwidth. Jin et al. [8] investigated VM deduplication chunking methods such as overlays to build multiple VMs from a single ‘base’ image. While these researches can provide some storage savings, they can be used only when timing constraints (e.g., file transfers with multiple nodes) is not considered.

Another key issue of cloud computing environments is energy efficiency. Some studies [9] claim that data centers account for between 1.2 and 2.0 percent of electricity consumed in the United States. It is also a known fact that many of the servers in these data centers run at a low utilization level of 10% to 15%. This causes significant waste through redundant hardware, memory, network devices, and power supplies. This energy waste could be easily avoided by just switching the power off. For this reason, energy-efficient BitTorrent proposals have been studied in [10] and [11]. Blackburn et al. [10] designed and implemented a green BitTorrent client that can drop its connections and go to sleep, yet still be active as a swarm member. This work is the first version to specifically look at the BitTorrent protocol in terms of energy savings. Anastasi et al. [11] proposed a green BitTorrent proxy model. The BitTorrent proxy gathers client requests at certain intervals. User PCs can then be switched off. As soon as user PCs join a swarm, the BitTorrent proxy provides the corresponding file to the user PCs. However, the energy-efficient BitTorrent proxy has minor flaws. While a proxy-based approach can save up to 95% of the energy consumption, it requires a cache replacement policy for how or when a very popular file resides in a proxy.

3. BitTorrent-Based Network Block Device

3.1 Image Provisioning Overview

To realize the large-scale distribution of VMs in cloud data centers, a data center administrator determines when or how VMs may be installed. Figure 1 illustrates the deployment sequence of physical and virtual machine images.

After a node is powered on, the DHCP server supplies a PXE boot plug-in with IP address, TFTP server address, and the boot images to download and execute. This boot image loads the installation kernel by performing the actual installation. The BitTorrent mechanism is used by the system image installation. The image server takes an important role in distributing Physical Machine (PM)/Virtual Machine (VM) images and conducting a BitTorrent tracker.

We classify each installation of physical and virtual images into one of three categories for VM deployment: static, homogeneous, and dynamic VM deployment. In the case of static and homogeneous VM deployment, VM images can be installed at the same installation time as a physical machine image. This approach is suitable to support the predefined PM/VM installation of large-scale data centers. In the case of dynamic VM deployment, only the torrent files of a VM image are distributed. Dynamic VM deployment is effective for dealing with various VM images. In this paper, we focus on dynamic VM file staging using the BitTorrent-based protocol.

3.2 The BitNBD Architecture

Figure 2 shows a BitTorrent-based network block device (BitNBD) architecture for provisioning virtual machines in large-scale data centers.

A client has both an NBD client module and our BitNBD module. The NBD client module provides a network block device (e.g., /dev/nbd0) to the virtual machine emulator (e.g., QEMU). The QEMU emulator launches a full system environment for guest systems. The NBD client module also communicates with the BitNBD module for reading and writing blocks via block I/O operations. The BitNBD module gets blocks of a VM image whenever they are requested by the NBD client module. The BitNBD also uses a ‘split read/write’ mechanism where the data is read from other peers while it is being written locally. If a block does not exist in the local cache (called a ‘fresh read’), the BitNBD module communicates with neighbor peers and a seeder (i.e., an image server), which contain the block. If a block is already cached, the block is obtained by the local cache. Write/rewrite blocks are stored in this local cache. Note that our BitNBD has the same restrictions for exclusive control as normal NBD.

Assume that a cloud service provider has used a group of nodes. There are two possibilities regarding the way an IaaS cloud is used; in the first, there are a small number of standardized VM images that are used repeatedly on many servers. In this first case, the server must duplicate the same
VM image on a physical node so that each VM instance uses it independently. Moreover, the standardized VM image is difficult to manipulate for many management issues: periodic OS updates, software installation, etc. In the second case, there are a large number of heterogeneous VM images. These images are impossible to be preinstalled on all the servers. Thus, the file sharing method is an alternative way to share VM images on many servers. However, an image server cannot process burst I/O requests from VM instances.

The BitNBD architecture has the benefit of management flexibility beyond the limitation of storage utilization and fast deployment. Frequently used blocks of VM images are shared in a BitTorrent network. Unused blocks of VM images are not cached until they are first used. Moreover, the BitNBD has no necessity to duplicate the same VM image on a physical node. Two clients in a physical node can share the same VM image, each with its own dirty writes. In the case of VM migration, the BitNBD only transfers updated blocks and its bitmap. This is a convenient way to improve VM file staging.

Now, we consider storage utilization in comparison with file copy and BitNBD methods. We assume the VM image size \( F \) to be in unity. The file copy method only requires \( F \) storage space to deal with one VM image. However, if the number of VM images increases up to \( n \), the total size of the storage space is equal to \((n + 1) \times F\). The storage space contains a source image (i.e., 1 image) and replicated images (i.e., \( n \) images). The BitNBD method in the same condition is from \( F + d \) to \( F + n \times d \), respectively. Value \( n \) is the number of VM instances. Value \( d \) is the sum of cached block size and bitmap index size (i.e., \( d = bs \times bn + i \)). Value \( bs \), \( bn \), and \( i \) represents the block size, the number of cached blocks, and the index size, respectively. Since the index size \((i)\) is relatively small, we disregard the current measurement in forming the new estimation. The number of cached blocks \((bn)\) is calculated as follows:

\[
bn = r(read) + w(write) + rw(rewrite).
\]

We classify each occurrence of a block I/O into one of four categories: \( r(read) \), \( rr(reread) \), \( w(write) \), and \( rw(rewrite) \). When a VM instance requires a read operation exactly once, it is called \( r(read) \). At this time, the BitNBD module receives a requested block among its peers and then saves the local caches. If the VM instance requires exactly the same block that is already cached, it is called \( rr(reread) \). The \( rr(reread) \) case has no storage overhead because the \( reread \) block is already cached. When a VM instance requires write/rewrite operations, \( w(write) \) and \( rw(rewrite) \) may occur in the same manner. Given VMs with file size \((F)\), the maximum storage quantity of the BitNBD method \((W_b)\) is less than or equal to that of the file copy method \((W_f)\). The total storage size of the two methods is calculated as follows:

\[
W_b = F + n \times bs \times (r + w + rw), \quad W_f = (n + 1) \times F.
\]

This expression is rewritten as \( bs \times (r + w + rw) < F \). It is true that all block I/O operations are dominant in file size \((F)\).

Another method for using a VM image is a file sharing method. A network file system (NFS) is used to share files over a network. A VM image is located in an image server. A client can access the exported directories. The storage overhead in this case is from zero to \((n + 1) \times F\), depending on the number of concurrent accesses. However, client access can potentially be burdened with a heavy read/write load of the image server. Unfortunately, files in the NFS protocol are cached only if they are larger than a specified minimum size (currently 1 page, 4 KB), smaller than a specified maximum size (currently 4096 pages, 16 MB), and have not changed recently (currently, this timeout is 30 seconds) [12]. A typical virtual machine (VM) image can range in size from a few gigabytes to many tens of gigabytes. Therefore, the NFS protocol has no benefit in terms of caching effect in VM instances.

Table 2 summarizes various VM staging methods in terms of total storage size, file caching operations, communication protocol, network I/O operations, and VM startup delay. The BitNBD method is an intermediate method used to efficiently distribute VM images and save the storage requirements.

3.3 BitTorrent Protocol Enhancement

The two main piece exchange strategies of BitTorrent are local rarest first [13] and tit-for-tat strategies [14]. Local rarest first strategy means that peers attempt to retrieve the rarest pieces based on their local knowledge. The tit-for-tat strategy means that a peer wanting to download pieces of other pieces has to upload its pieces to its peers. Studies have revealed that these two strategies guarantee close-to-ideal entropy. Moreover, a peer in BitTorrent usually keeps connections with many other peers. For a connection between two
peers, there are two states: choked or not choked. A peer could choose to ‘choke’ a connection by refusing to upload any more pieces. However, a client must be fully powered-on to establish a TCP connection even when that peer is choked. If the connection is disconnected, other clients are assumed to be physically disconnected.

With these points in mind, there are some aspects to consider when applying the BitNBD method in the traditional BitTorrent protocol: block request ordering and an energy-saving issue.

Initially, the local rarest first strategy is not efficient to minimize the startup delay of VMs. Legacy BitTorrent splits the file to be downloaded into pieces that are downloaded in a non-sequential order. While booting, however, the BitNBD module only requires the boot image and its associated libraries. For this reason, it is possible in our current implementation for all VM images to be preprocessed in order to generate a hint file (i.e., a data stream is a bounded sequence of \(<\text{SEQ}, \text{Offset}, \text{Number of blocks}>\) pairs). The hint file in the image server is created only once at the first time. The BitNBD module stores the related offset and the number of blocks. The time to creation is the same as the initial booting time. The BitNBD module contacts each of the peers in order to download pieces in the same order as the hint file’s data stream. The amount of time spent while booting is determined solely by the popularity of the pieces. Fortunately, the boot image and its associated libraries are globally popular pieces.

Figure 3 shows an example of BitNBD’s block request patterns with six active peers in the same VM image.

The beginning point for downloading pieces is defined as the first piece that has not yet been downloaded but has not missed its SEQ number. The brighter-filled box in Fig. 3 represents popular pieces that have already been downloaded. A check mark indicates the current downloading position of each peer. The sixth peer (namely, ‘L6’) joins a swarm with one seeder and two neighboring peers. If neighbor peers (i.e., ‘L4’ and ‘L5’) joined before the arriving L6 peer, the two neighbor peers already own pieces available for sharing. After a period of downloading all blocks in the hint file, the BitNBD module does not require the pieces unlike in a legacy BitTorrent protocol. At the time of a new block request of the NBD client module, the BitNBD module requires the corresponding pieces.

The piece picker is a central component in a BitTorrent implementation. In general, piece picker algorithm keeps a list of all available pieces sorted by rarity. The piece picker algorithm can also be modified to download specific pieces by the selective download feature. So, we modified the piece picker algorithm for BitNBD implementation. As shown in Table 3, our piece picker function (i.e., \texttt{get\_next\_pieces()}) provides the portion of master boot records (MBRs) firstly, and then it returns the pieces in a hint file. Last line in Ta-

| Table 2  | Comparison of various VM staging methods. |
|----------|------------------------------------------|
|          | Total storage size | File caching operations |
| File copy method | \((n+1)F\) | Read/rew
d/\write/rewrite |
| BitNBD method   | \(b\ast(r+w+rw)+F\) | R
d/\write/rewrite |
| File sharing method | \(F\ast(n+1)F\) | None |

| Table 3  | Piece picker algorithm of BitNBD. |
|----------|----------------------------------|
| def get\_next\_pieces(self,havefunc,seed=false) |
| # To speed things up, mbr blocks are requested firstly. |
| mbr = [0,1,2,3] | # block list of master boot records |
| # ‘rarest\_first\_cutoff’ means the initial status of BitNBD. |
| if self.numgot < self.config['rarest\_first\_cutoff']:
  for i in mbr:
    # havefunc is the existing flag whether it has or not. |
    if havefunc(i):
      return i | # return block list of mbr. |
| # Requesting blocks from a hint file. Self.pnbd\_list stores them. |
| if i in self.pnbd\_list:
  if havefunc(i):
    self.pnbd\_list.remove(i) |
| # If all blocks in a hint file are exhausted, waits until new requests are received. Self.go\_ahead is the flag whether it starts or not. |
| if not self.go\_ahead:
  return None |
| # General rarest\_first policy is called. This function represents general piece picker algorithm in BitTorrent implementation. |
| general\_piece\_picker\_algorithm(havefunc,seed) |
Table 3 indicates that our `get_next_pieces()` function calls general rarest-first policy in BitTorrent.

Secondly, a legacy BitTorrent protocol is not energy efficient. All peers have to stay connected to the BitTorrent networks during the entire downloading activity of the requested files. In previous work, we designed and implemented an energy-saving mode to allow unused peers to hibernate [15]. Table 4 shows peer wire protocol messages utilizing this energy savings.

The hibernation message is of a fixed length, and is used for hibernation. The payload is identical to that of the `peer_id` and is used during `energy-saving mode`. A wakeup message is used to reactivate a peer with the full copy of the file. These energy-saving messages allow the peer to hibernate or reactivate easily.

Figure 4 shows an instance of communication among an enhanced seeder, tracker, and leechers.

To implement the communication, the leecher’s communication protocol has to be modified, whereas the structure of the peers is unchanged. We define a new peer status including energy-saving mode, timers, and events to achieve these objectives. The detailed workflow is as follows. If

Leecher-1 wants to download a VM image file, Leecher-1 requires a tracker via a `<GET>` message. After a successful handshake, the connection is established and both clients (i.e., Seeder and Leecher-1) will start communicating by sending messages to one another. Leecher-1 requests a certain portion of a piece of the image, including its index, offset, and length. After downloading is complete, Leecher-1 changes its state from a leecher to a seeder. The re-activation message could be a standard magic packet or other predefined packet type. The leecher needs to check the appropriate energy-saving mode as well since the tracker can obtain peer information. The connection between Leecher-1 and Leecher-2 must be re-established before file pieces can be uploaded or downloaded. The energy-saving mode of a peer cannot maintain a TCP connection with other peers, or respond to a request from others.

The energy-saving mode of the BitNBD is useful to minimize unnecessary BitTorrent traffic. When a VM is put into suspend (pause) mode, the corresponding BitNBD module can also be halted. The next time a user tries to resume (unpause) the VM (after it has been paused for many hours), the BitNBD module may be reactivated.

### 4. Experimental Results

#### 4.1 Testbed Environment

Figure 5 illustrates a high-level picture of the cluster being installed in late 2009 at ETRI. The platform currently consists of 512 nodes, each with quad-core X3320 processors and 2 GBs of main memory. Each compute node has 500 GB storage and 2 X 1 Gb/s link to each core switch. For the purpose of VM deployment, we use one of the 512 nodes as the image server. It contains entire VM images for distributing other nodes. The initial VM image size is 4 GB.

#### 4.2 Power Consumption Evaluation

Figure 6 illustrates power consumption (watts) in 1U server. Tickless idle state consumes 114.6 watts, but hibernation state only consumes 27.6 watts. Dynamic power management (DPM) provides 5–6 power management level and reduces energy costs by 80%. Through reducing the energy demands of cloud data center through dynamic management of power level, a pool of server or necessary resources (i.e., memory, disk, and network) can be saved. This leads to save power/cooling costs, and to provide simple ways to make your server room green.

To demonstrate effectiveness of our energy-saving BitTorrent protocol, we use a simple fluid model. The derivation here follows the fluid modeling approach of Qiu and Srikant [16]. In our model, seeder peers are divided into two classes in a BitTorrent system: seeders and hibernated seeders.

A glossary of the model notations and parameters is listed in Table 5.

### Table 4 Energy-saving peer wire protocol messages.

| Message type  | Message format, meaning |
|---------------|-------------------------|
| ES_HIBERNATION| `<0003><9><peer_id>` Tell his tracker that no data will be downloading until new peer requests. |
| ES_WAKEUP     | `<0003><10><peer_id>` Announce that a peer with peer_id will be awakened. |

**Fig. 4** BitTorrent node communications with energy-saving mode.
As shown in Fig. 7, we assume that all U upload connections are fully utilized for all peers. The downloaders enter the swarm system at a rate $\lambda$, and get converted to seeders and hibernated seeders at a rate $(1 - p)(x + y)UC$ and $p(x + y)UC$, respectively.

A deterministic fluid model for the evolution of the number of peers (downloaders, seeders, and hibernated seeders) is given by

$$
\begin{align*}
\frac{dx}{dt} &= \lambda - (x + y)UC, \\
\frac{dy}{dt} &= (x + y)UC + \omega z - \mu y \\
\frac{dz}{dt} &= p\mu y - \omega z \\
\end{align*}
$$

Table 5: Notations and model parameters.

| Notation | Description |
|----------|-------------|
| $x(t)$  | Number of downloaders in the system at time $t$ |
| $y(t)$  | Number of seeders in the system at time $t$ |
| $z(t)$  | Number of hibernated seeders in the system at time $t$ |
| $\lambda$ | The arrival rate of the new downloaders |
| $\mu$   | The departure rate of seeders. This means that $1/\mu$ is seed residence time in the system. |
| $U$     | Maximum number of simultaneous upload connections by a peer |
| $\omega$ | The wake-up rate of hibernated seeders |
| $\rho$  | The ratio of the number of hibernated seeders to the sum number of seeders (hibernating ratio: $0 \leq \rho < 1$) |

To study the system in steady state, we let

$$
\frac{dx}{dt} = \frac{dy}{dt} = \frac{dz}{dt} = 0
$$

Solving the equation from (1) and (2), we have

$$
\begin{align*}
\overline{x} &= \lambda \left( \frac{1}{UC} - \frac{1}{(1 - p)\mu} \right) \\
\overline{y} &= \frac{\lambda}{(1 - p)\mu} \\
\overline{z} &= \omega \frac{\rho}{1 - p}
\end{align*}
$$

The steady-state results show that the number ofdownloaders, seeders, and hibernated seeders in the system is linearly dependent on the peer arrival rate.

The average download latency $T$ from (3) can be directly computed using Little’s law. Specifically:

$$
T = \frac{\overline{x}}{\lambda} = \left( \frac{1}{UC} - \frac{1}{(1 - p)\mu} \right) \times \frac{1}{\mu}
$$

This expression shows that the expected download time...
in steady state is independent of the peer arrival rate. We also find that the download latency ($T$) decreases as the seeders (i.e., seeders and hibernated seeders) residence time (i.e., $1/\mu$) increases. Obviously, the download latency ($T$) also decreases as the hibernating ratio $p$ increases. Specifically, if the ratio $p$ is zero (i.e., it means that hibernated seeders are empty), the download latency ($T$) is equal to [17]. Theoretically, the energy-saving BitTorrent protocol reduces power consumption as the hibernating ratio $p$ increases.

4.3 Minimum Transfer Time of Various Transport Protocols

Table 6 shows the minimum transfer time of various transport protocols: unicast, multicast, pure BitTorrent, NFS file sharing, and BitNBD. We measure total elapsed time for transferring 10 VM images into 10 physical machines each.

To distribute VM images, the most frequently used protocol is unicast (e.g., FTP). The benefits of unicast are that it is simple to deploy and to manage straightforward, but its drawbacks are network congestion if the large number of VMs requires simultaneously. Four test cases with “Unicast” labels show the effect of how many VMs are copied sequentially or simultaneously. Obviously, “Unicast (3/3/1 VMs sequentially)” is better than other “Unicast” test cases. To ensure how many VMs are copied simultaneously, however, it depends on the network bandwidth. “Multicast with FEC (10 VMs simultaneously)” is to use UDP over IP-multicast. A server-driven multicast strategy theoretically scales very well compared to the classical client-server approach (i.e., unicast). However, UDP over IP-multicast requires a highly reliable network (e.g., Forward Error Correction (FEC)). With the supporting of networking hardware, Multicast can be the best choice of transferring VM images. “Pure BitTorrent (10 VMs simultaneously)” means that clients join the BitTorrent networks and then receive the entire VM images. The main difference between pure BitTorrent and BitNBD is that pure BitTorrent requires image transfer before using the VM image. However, BitNBD method has no requirement of image transfer. Last two methods (i.e., NFS file sharing and BitNBD) have no necessity for transferring VM images into particular locations in physical machines. In terms of VM deployment, file sharing approach is the most convenient way to use.

4.4 Booting Time of Various Transport Protocols

Table 7 shows booting time of instances ranging from 10 to 50 VMs. Its VM image size is 4 GB.

Local boot takes about 1 minute 25 seconds. Their images are located in 50 physical nodes independently. In NFS boot case, all the images are located in a NFS server. Thus, NFS boot leads to network congestion when a large number of VMs are accessed simultaneously. In BitNBD boot case, single image is located in a BitNBD server. BitNBD boot has little interference with a large number of VM instances. It shows that VMs can start up in 102 to 134 seconds. Our BitNBD method is better than NFS boot method in terms of VM startup. Most of all, Local boot is superior to other methods in terms of boot times. After first boot, BitNBD receives booting related blocks from other peers. “BitNBD first boot” has a slight startup delay compared to NFS boot at the case of 10 or 20 VMs. “BitNBD next boot” means that the BitNBD has locally cached blocks related booting stage. The boot time of “BitNBD next boot” is exactly same as the time of “Local boot”.

4.5 Block Request Pattern of the BitNBD

Figure 8 illustrates block request patterns of rarest-first and BitNBD. At the initial state, peers have no pieces entirely. Thus, rarest-first policy leads to request pieces evenly. Moreover, general rarest-first policy has no knowledge of request order. The BitNBD policy, however, has known block request order and necessary block numbers from preprocessed hint file. Hint file in this experiment contains request orders of eight hundreds blocks approximately. The BitNBD block request pattern in Fig. 8 shows that the block numbers over 17,000 are not used and lower blocks (i.e., 0 to 5,000) are used frequently. Lower two graphs in Fig. 8 show comparison of rarest-first policy and BitNBD policy in terms of VM disk layout. As a result, frequently used blocks of VM images are shared in a BitTorrent network. Unused blocks of VM images are not cached until they are first used.

| Methods                                      | Time(h:mm:ss) |
|----------------------------------------------|---------------|
| Unicast(1/1/1/1/1/1 VMs sequentially)       | 0:53:36       |
| Unicast(2/2/2 VMs sequentially)             | 0:28:36       |
| Unicast(3/3/1 VMs sequentially)             | 0:23:06       |
| Unicast(10 VMs simultaneously)              | 0:59:07       |
| Multicast with FEC (10 VMs simultaneously)  | 0:40:17       |
| Pure BitTorrent (10 VMs simultaneously)      | 0:16:30       |
| NFS file sharing                             | 0:00:00       |
| BitNBD                                       |               |

| Number of VMs | Local boot | NFS boot | BitNBD first boot | BitNBD next boot |
|---------------|------------|----------|-------------------|------------------|
| 10 VMs        | 0:01:25    | 0:01:31  | 0:01:42           |                  |
| 20 VMs        | 0:01:47    | 0:01:54  |                   |                  |
| 30 VMs        | 0:02:14    | 0:02:56  |                   |                  |
| 40 VMs        | 0:02:52    | 0:02:12  |                   |                  |
| 50 VMs        | 0:03:29    | 0:02:34  |                   |                  |
4.6 VM Transfer Bandwidth of the BitNBD

Figure 9 illustrates the VM transfer bandwidth curves of 10 to 50 VM instances. The x-axis denotes the total elapsed time and the y-axis denotes the bandwidth (KB/s). The characteristic of VM startup is that VMs draw lots of disk I/Os at initial time but drop considerably after thirty seconds. For these reasons, startup delay depends on how fast data can be transferred at initial time.

4.7 Storage Utilization of the BitNBD

The initial VM image size is 4 GB, and is located in an image server. Table 8 shows the number of request blocks of various operating systems. The BitNBD only requires a few hundred megabytes or less. The total number of bytes received from an image server ranges from 144 MB to 383 MB until the booting stage. The two activities, web surfing and viewing YouTube clips, hold from 10 MB to 2.2 MB, respectively.

As a result, the BitNBD can start up a VM instance within 30 seconds and keep potentially whole blocks of a VM image after 120 seconds. For the purpose of storage saving, the BitNBD, however, only requires necessary blocks that are required for current VM instances. The actual transfer size is between approximately 150 MB and 400 MB. Therefore, the BitNBD reduces the storage space for the fifty VMs by over 73% to 90%.

Table 8 Number of blocks in BitNBD operations (various oses, 4 GB VM size).

| OS            | Read blocks | Write blocks | Rewrite blocks | Total Mbytes |
|---------------|-------------|--------------|----------------|--------------|
| Windows 7     | 94,371      | 2,998        | 3              | 383MB        |
| Windows XP    | 27,510      | 3,657        | 0              | 144MB        |
| Fedora Core 11| 47,185      | 1,164        | 0              | 189MB        |
| Ubuntu 9      | 39,519      | 1,719        | 0              | 163MB        |
| RedHat Linux 5| 81,522      | 4,353        | 0              | 343MB        |
| CentOS 5.3    | 89,109      | 3,261        | 0              | 372MB        |

5. Conclusion

Today, a typical virtual machine (VM) image can range in size from a few to many tens of gigabytes. Naive approaches to transferring VM images can result in intolerable latencies and storage overhead. This paper provides a solid idea that cloud resources (i.e., virtual machines) can be efficiently distributed via the BitTorrent protocol. The ‘split read/write’ mechanism of BitNBD can reduce disk space. We also propose a new method for reducing VM startup delays and so improving energy efficiency. Experimental results showed that the BitNBD can save storage space for fifty VMs by over 73% to 90%.

We noted that a hint file works well for decreasing initial startup delays. In particular, homogeneous VM deployment results in minimum distribution time and better storage utilization. The energy-saving concept eliminates permanent connectivity requirements of a legacy BitTorrent protocol. Our goal for the near future is to consolidate the BitNBD into a scheduling method focusing on energy-saving resource management, prefetching, and caching of VM images.

Our study will be potentially useful in environments where large-scale virtual machines are driven by a cloud ser-
vice platform. A cloud service provider allows users and applications to access data efficiently from any location.

References

[1] D. Nurmi, R. Wołski, C. Grzegorczyk, G. Obertelli, S. Soman, L. Yousef, and D. Zagorodnov, “The eucalyptus open-source cloud-computing system,” CGCC, 2009.

[2] B. Sotomayor, R.S. Montero, L.M. Llorente, and I. Foster, “Virtual infrastructure management in private and hybrid clouds,” IEEE Internet Computing, vol.13, no.5, pp.14–22, 2009.

[3] Nimbus. http://workspace.globus.org

[4] ASPEN. http://storage.cloudwww.com

[5] M.A. Kozuch, M.P. Ryan, R. Gass, S.W. Schlosser, D. O’Hallaron, J. Cipar, E. Krevat, J. López, M. Stroucken, and G.R. Ganger, Location-Aware Cluster Management, ACDC, 2009.

[6] M. Schmidt, N. Fallenbeck, M. Smith, and B. Freisleben, “Efficient distribution of virtual machines for cloud computing,” PDP, 2010.

[7] C. Policroniades and I. Pratt, “Alternatives for detecting redundancy in storage systems data,” Proc. 2004 USENIX Annual Technical Conference, 2004.

[8] K. Jin and E.L. Miller, “The effectiveness of deduplication on virtual machine disk images,” SYSTOR, 2009.

[9] Data Center Energy Forecast Executive Summary: https://microsite.accenture.com/svlgreport/Documents/pdf/SVLG_ExecutiveSummary.pdf

[10] J. Blackburn and K. Christensen, “A simulation study of a new green BitTorrent,” GreenComm, 2009.

[11] G. Anastasi, I. Giannetti, and A. Passarella, “A BitTorrent proxy for green Internet file sharing: Design and experimental evaluation,” Comput. Commun., vol.33, no.7, pp.794–802, 2010.

[12] http://badros.com/greg/doc/enhanced-linux-nfs-client/node6.html

[13] A. Legout, G. Urvoy-Keller, and P. Michiardi, “Rarest first and choke algorithms are enough,” Proc. ACM SIGCOMM, 2006.

[14] http://en.wikipedia.org/wiki/Tit_for_tat

[15] Y.J. Lee, J.H. Jeong, H.Y. Kim, and C.-H. Lee, “Energy-saving set-top box enhancement in BitTorrent networks,” NOMS, 2010.

[16] D. Qiu and R. Srikant, “Modeling and performance analysis of BitTorrent-like peer-to-peer networks,” Proc. ACM SIGCOMM, 2004.

[17] N. Parvez, C. Williamson, A. Mahanti, and N. Carlsson, “Analysis of BitTorrent-like protocols for on-demand stored media streaming,” Proc. ACM SIGMETRICS, 2008.

Yong-Ju Lee received the B.S. and M.S. degrees in computer engineering from Chonbuk National University, Korea in 1999 and 2001, respectively. He joined ETRI (Electronics and Telecommunications Research Institute), Daejeon, Korea in 2001. From 2001 to 2002, he involved in development of SAN-based data repository system supporting general file system interfaces. From 2003 to 2006, he involved in development of Next Generation Internet Server providing HDTV quality multimedia services. Since 2007, he has currently developed resource management system for future internet service, and has been working toward the Ph.D. degree in computer engineering at ChungNam National University, Korea. His research interests include multimedia streaming, distributed file system, and large-scale cluster management system.

Hag-Young Kim received the B.S. and M.S. degrees in electronics engineering from Kyungpook National University, Korea in 1983, 1985, and Ph.D. degree in computer engineering from Chungnam National University, Korea in 2003. He joined ETRI in 1988, and he has currently served as the Project Leader of the Media Streaming Research Team of ETRI. His current interests include computer architecture, high-speed network architecture, multimedia, middleware, and digital cable broadcast system.

Cheol-Hoon Lee is the corresponding author and received the B.S. degree in electronics engineering from Seoul National University, Seoul, Korea in 1983, and the M.S. and Ph.D. degrees in electrical engineering from Korea Advanced Institute of Science and Technology, Seoul, Korea in 1988 and 1992, respectively. From 1983 to 1994, he worked at Samsung Electronics Company in Seoul, Korea as a researcher. From 1994 to 1995, he was with the University of Michigan, Ann Arbor, as a research scientist at the Real-Time Computing Laboratory. Since 1995, he has been a professor in the Department of Computer Engineering, Chungnam National University, Taejeon, Korea. His research interests include parallel processing, operating system, real-time and fault-tolerant computing, and microprocessor design.