Automating OS/SW Provisioning for Building Enterprise Data Center

Yong-Ju Lee[a], Hag-Young Kim[a], Cheol-Hoon Lee[b]

[a]*Dept. of Internet Platform Research, SW & Contents Research Lab, Electronics and Telecommunications Research Institute*

[b]*Dept. of Computer Engineering, Chungnam National University*

{yongju, h0kim}@etri.re.kr, clee@cnu.ac.kr

***Abstract*— 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. This paper describes BitTorrent-based OS provisioning method to afford large-scale data centers. It can save the time necessary for installing and deploying operating systems and software.**

Index — OS provisioning, BitTorrent protocol, Enterprise data center

I. INTRODUCTION

In large-scale data centers, manually installing OS is not practical due to its huge time/labor cost. There must be a way to enable auto OS installation, PXE(Preboot eXecution Environment)[1] technology. PXE is maintained by the Intel Corporation, it is an industry standard client/server interface that allows networked computers that are not yet loaded with an operating system to be configured and booted remotely by an administrator. The PXE code is typically delivered with a new computer on a read-only memory chip or boot disk that allows the computer (a client) to communicate with the network server so that the client machine can be remotely configured and its operating system can be remotely booted. There are other methods available for doing automatic installs, such as RedHat's KickStart[2] which installs systems based on a list of pre-defined packages. But package based installs are very limiting in that they generally don't have an automated way for dealing with non-packaged files. If you re-compile your kernel, add a piece of non-packaged software, or modify certain configuration files, you are usually required to do some sort of scripting or programming to deal with these special cases.

This paper gives you a solid idea of the rapid OS deployment using BitTorrent protocol. We explore how the BitTorrent protocol can support rapid OS installation, and we compare performance in terms of various transport protocols

II. BITTORRENT PROTOCOL OVERVIEW

A. BitTorrent protocol overview

BitTorrent[2] uses a centralized server called tracker for peer management. Only one tracker works in each BitTorrent network. The origin peer of the content that is distributed in the network is called initial seed, and often becomes the same peer as the tracker. Each BitTorrent network deals with only one file and the file is divided into a large number of smaller data chunks called pieces. The two main piece exchange strategy of BitTorrent is local rarest first[3] and tit-for-tat strategy[4]. Local rarest first strategy means peers attempt to retrieve rarest pieces based on their local knowledge. Tit-for-tat strategy means that a peer want to download pieces in other pieces has to upload its pieces to the peers. Studies have revealed that these two strategies guarantees a close-to-ideal entropy.

A peer in BitTorrent usually keeps connections with many other peers. For a connection between two peers, there are tow states: choked or not. A peer could choose to 'choke' a connection to refusing to upload pieces any more. 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. Figure 1 illustrates the communication paths of a typical BitTorrent swarm. As seen here, there exists a control path between each peer and the centralized tracker, cooperates among peers while uploading and downloading a content.

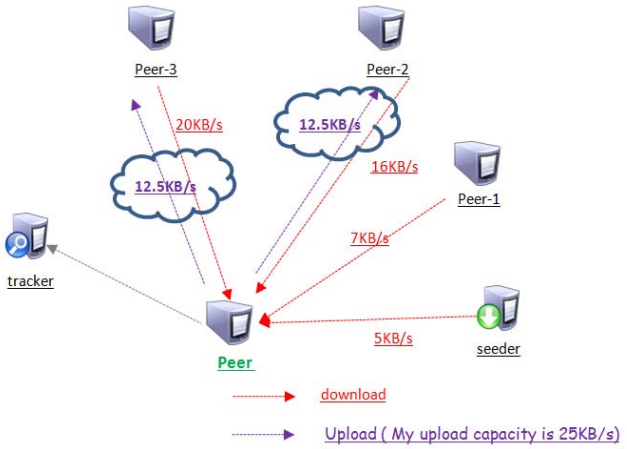

Figure 1 BitTorrent peer communications

BitTorrent consists of two protocols [5]. At first, Tracker HTTP Protocol(THP) is resposible for how an agent joins a swarm and how an agent learns of other agents in the swarm. Next, Peer Wire Protocol(PWP) is used for how agents connect with each other, how agents tell each other what data they have , and how agents request and send data. Standard PWP messages are as follows.

Table I shows standard peer wire protocol messages and its format. Depending on message type, each message has its own format. Thus, message length will have differing values as well. The following gives the relationship between the message type and message (as well as message length). Note that BT_CHOKE, BT_UNCHOKE, BT_INTERESTED, BT_UNINTERESTED, and BT_KEEP-ALIVE do not have this field.

TABLE I
STANDARD PEER WIRE PROTOCOL MESSAGES

Message type	Message format, meaning
BT_CHOKE	<0001><0> No data will be uploaded until unchoking occurs.
BT_UNCHOKE	<0001><1> Tell his peer that his requests will be answered.
BT_INTERESTED	<0001><2> Tell his peer that he has something to download from him.
BT_UNINTERESTED	<0001><3> Tell his peer that he does not want to download from him.
BT_HAVE	<0005><4><index> Announce that he has successfully downloaded a piece of that index.
BT_BITFIELD	<0001+length><5><bitfield> Sent at the beginning of each successful handshake.
BT_REQUEST	<000d><6><[index][begin][length]> Request for a portion of a piece of that particular index.
BT_PIECE	<0009+block length><7><[index][begin][block]> Send a portion of a piece of that index at offset “begin”.
BT_CANCEL	<000d><8><[index][begin][block]> Cancel an earlier request of a piece of that particular index.
BT_KEEP-ALIVE	<0000> Send a keep alive.

III. HIERARCHICAL OS/SW PROVISIONING

A. *Physical/virtual resources*

Resource set distinguish two types of resources, Physical Resource Set (PRS) and Virtual Resource Set (VRS). Both of these service types provide a management front-end API for a set or pool of resources in order to allow higher level services to automate setup and demand-based scalability, fail-over and operating system hosting. Examples of PRS include Emulab [6] and iLO [7]. VRS service includes Amazon EC2 [8], Eucalyptus [9], Tycoon [10], Nimbus [11]. The reason to split these resource set services into two types allows automated management of physical as well as virtual resources.

B. *OS provisioning sequences*

Figure 2 illustrates OS provisioning sequence which indicates at the time between system power-on and monitoring daemon startup. Each transport protocol consumes different times in terms of number of nodes, image file size, and request patterns.

After a node is powered on, the DHCP server supplies the PXE boot plug-in with and IP address and TFTP server address, and the state 1 image boot-loader name to download and execute. This image loads the installation kernel with performs the actual installation. Various transport mechanism (FTP, NFS, HTTP, Multicast, and BitTorrent) is used by the installation of system image. Before rebooting a node, post action procedures execute to accommodate the system-specific configurations.

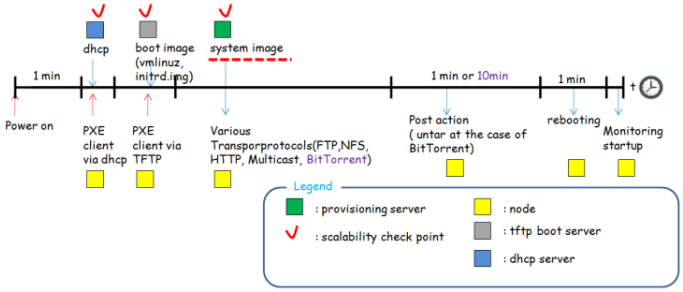

Figure 2 Various transport protocol evaluation

C. *Enhanced BitTorrent-based OS provisioning*

Figure 3 shows how we can install a large amount of nodes using BitTorrent-based provisioning. At first, we get a system image from a client and then install several nodes per rack. Next, we install entire nodes in a rack simultaneously. This approach can guarantee the scalability of nodes.

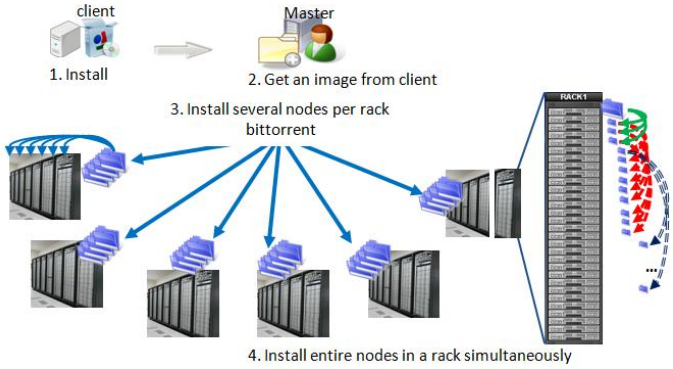

Figure 3 Enhanced BitTorrent-based OS provisioning

D. *Dynamic SW provisioning*

Initial deployment and subsequent dynamic reconfiguration of a software is difficult because of many interdependdent factors including software dependencies and installing orders. We have developed a dynamic SW provisioning technique for carrying out the deployment of softwares. A service consists of several software packages that are used rpm-based installation procedures. For rapid deployment of services, dynamic deployment policy is required. At the time of a service request, service manager sends a software request message in Figure 4.

```
||$(TYPE)||$(CLASS)||$(COMMAND)||$(CODE)||$(MESSAGE0)||$(MESSAGE1)||$(MESSAGE2)||...||eom

TYPE: req, res
CLASS: provisioning
CODE: null
MESSAGE0: node name
MESSAGE1: service name
MESSAGE2: messages

||req||provisioning||install||null||node17||web||eom   : request node17 to install web service.
```

Figure 4 Service software request format

Figure 5 illustrates service software DB schema. A

service(e.g., VOD service, Web service) consists of softwares. Each RPM has its installation order, startup scripts, and process name. We also use the Yellowdog Update, Modified(YUM[14]) that is an open-source command-line package-management utility for RPM-compatible Linux operationg systems and has been released under the GNU.

Service name	Software name	RPM name	Install order	Script	Process name
VOD	cs#1	std_cs-1.0-1.i386.rpm	2	/etc/init.d/cs	mediaServer
	cd#1	std_cd-1.0-1.i386.rpm	1	Null	null
WEB	apache#1	httpd-2.2.9-4.GxD1.i386.rpm	1	/etc/init.d/httpd	httpd

Figure 5 Software DB schema

Figure 6 shows an example of service GUI. Each service uses several service nodes. For elastic load balancing, we have used LBS servers. The LBS server generates performance metrics.

Figure 6 Example of service GUI

IV. Performance Evaluation

A. Performance metrics

Table II shows performance measurent environments. We have used ten nodes which consists of Intel dual core CPU, 2GB memory, and 500GB SATA disks.

TABLE II
PERFORMANCE MEASUREMENT ENVIRONMENT

Number of nodes	10 nodes
CPU	Intel(R) CPU X3320 @ 2.50GHz
MEMORY	2GB
DISK	SATA 500GB 7200RPM
FILE	1.5GB HD MPEG-2 TS
Transport protocol	rsync, ftp, multicast, BitTorrent

B. Comparison of the Transport protocols

To deliver OS/VM images, the most frequently used protocol is ftp. The design of this protocol respects the classical client-server approach, so it has a scalability protocol: each client's bandwidth can be evaluated as Us/N, where Us is the upload bandwidth and N is the total number of the clients. Another approach is to use UDP over IP-multicast. A server-driven multicast strategy theoretically scales very well compared to the classical client-server approach. In this case the download bandwidth of each client is equal to Us. However, UDP over IP-multicast requires a highly reliable network (e.g., Forward Error Correction (FEC). A BitTorrent allows each client to join the distribution of the content at any time, requesting the pieces it needs in any order. Using the simple fluid model of Qiu and Srikant [15,16], the total upload, obviously, is limited to Us. Figure 7 shows elapsed time of various distribution methods

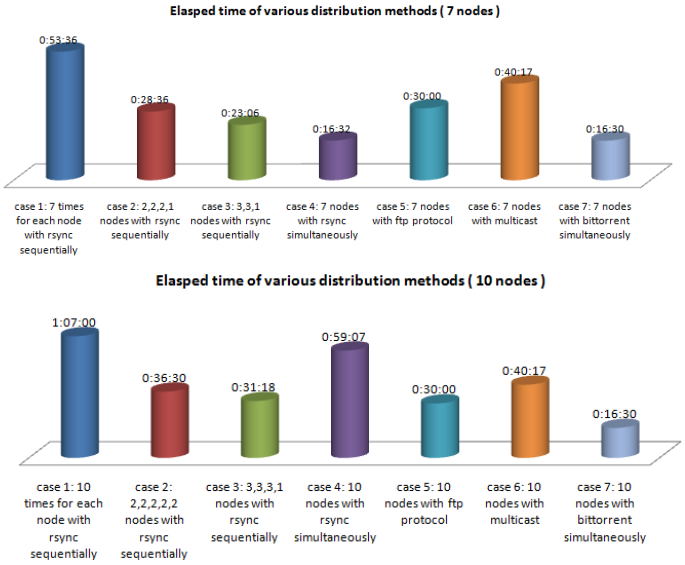

Figure 7 Elapsed time of various distribution methods

Figure 8 illustrates installation time. OS provisioning time for 230 nodes (920 cores) is from 14 to 23 minutes. It takes 17 minutes 15 seconds on average.

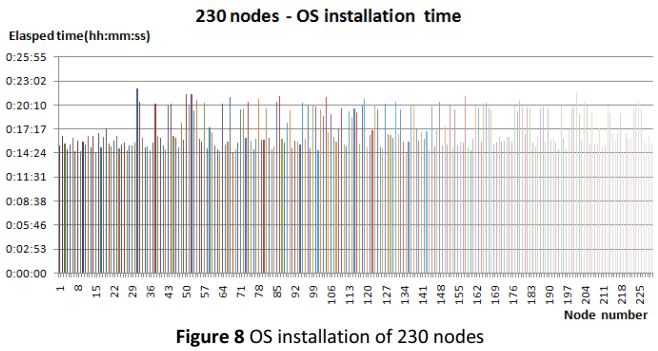

Figure 8 OS installation of 230 nodes

Figure 9 shows download bandwidth curve of 230 nodes. X-axis denotes elapsed time. Y-axis denotes download bandwidth (KB/s), and Z-axis denotes node number ranging from 1 to 230.

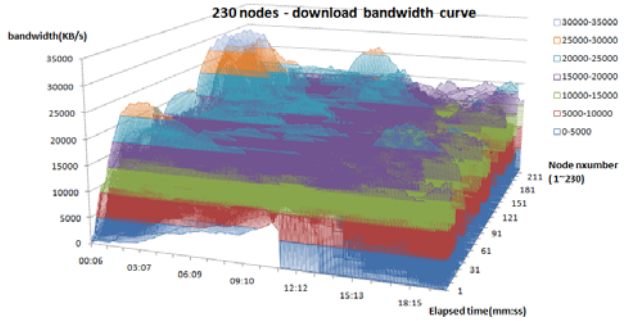

Figure 9 Download bandwidth curve of 230 nodes

Figure 10 depicts a node's download/upload bandwidth as download percent increases. The node is one of 230 nodes. At the time of 4% downloading, uploading other peers is activated.

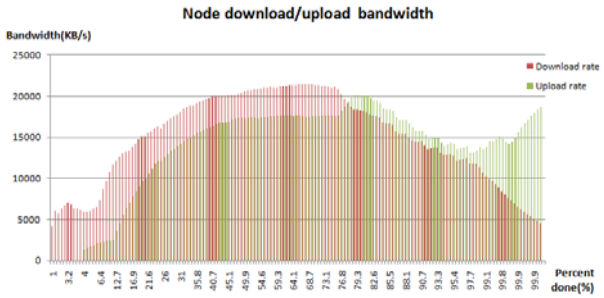

Figure 10 1-node download/upload distributions

V. CONCLUSION

In this paper, we explore how the BitTorrent protocol can support rapid OS installation, and we have compared performance in terms of various transport protocols. Automating OS/SW provisioning for large-scale data centers is also essential feature of innovative cloud computing. In future work, we will extend our research in virtual machines(VMs) provisioning.

REFERENCES

[1] http://en.wikipedia.org/wiki/PXE
[2] http://www.stanford.edu/~alfw/PXE-Kickstart/PXE-Kickstart.html
[3] A. Legout, G. Urvoy-Keller, and P. Michiardi. Rarest first and choke algorithms are enough. In Proc. ACM SIGCOMM Internet Measurement Conference, Rio de Janiero, Brazil, Oct. 2006.
[4] http://en.wikipedia.org/wiki/Tit_for_tat
[5] http//www.bittorrent.com
[6] M. Hibler, R. Ricci, L. Stoller, J. Duerig, S. Guruprasad, T. Stack, K. Webb, and J. Lepreau. Large-scale Virtualization in the Emulab Network Testbed. In Proceedings of the 2008 USENIX Annual Technical Conference, 2008.
[7] Hewlett-Packard. HP Integrated Lights-Out 2 User Guide. Technical report, HP, 2009.
[8] Amazon Web Services. Amazon web services homepage. http://aws.amazon.com, seen: 2008-12-05, 2008.
[9] D. Nurmi, R. Wolski, C. Grzegorczyk, G. Obertelli, S. Soman, L. Youseff, and D. Zagorodnov. The Eucalyptus open-source cloud-computing system. Proceedings of Cloud Computing and Its Application, 2008.
[10] K. Lai, L. Rasmusson, E. Adar, S. Sorkin, L. Zhang, and
[11] B. A. Huberman. Tycoon: an Implementation of a Distributed Market-Based Resource Allocation System. Multiagent and
[12] Grid Systems, 1(3):169–182, Aug. 2005.
[13] Nimbus. http://workspace.globus.org.
[14] http://www.phy.duke.edu/~rgb/General/yum_HOWTO/yum_HOWTO/yum_HOWTO.html
[15] D. Qiu and R. Srikant. Modeling and Performance Analysis of BitTorrent-like Peer-to-Peer Networks. In Proc. ACM SIGCOMM 2004.
[16] Y. Tian, D. Wu, K. Ng, Modeling, analysis and improvement for bittorrent-like file sharing networks, in: Proc. of IEEE INFOCOM, 2006