Efficient Streaming in Virtualized Thin Client Environment

Abhishek Mathur[1], Aparajita Rai[2], Ikya Jupudy[3], Lavanya K[4]
School of Computing Science and Engineering
VIT University, Vellore 632014, Tamil Nadu, India
{abhishekmathur0525@gmail.com} {aparajitarai01@gmail.com} {ikya07@gmail.com} {lavanya.sendhilvel@gmail.com}

ABSTRACT: *In the current scenario of a thin client environment the client mostly depends on the cloud or a remote server for most of its computing needs. As a result huge amount of data has to be transferred continuously over the network all the time. In such a case network performance is a severe issue. It is also important to acknowledge that network bandwidth and performance is more critical in any type of cloud-based computing model. If data transfer is taking too much time then irrespective of the cloud technologies that you use, it will not result in performance and user satisfaction. IT organizations spend huge money on dedicated computers and network infrastructure. This project is an attempt to address this problem and come up with some improved solution to manage the network traffic that will result in seamless operation of thin client devices in a virtualized environment. Firstly, we aim to study the factors affecting the network performance in a virtualized thin client environment and then based on this research propose and implement a solution to efficiently address the problems. Also when an organization uses multiple thin client machines for their business the scalability of the system will also be improved by our product.*

Keywords: Thin Clients, Cloud Client Computing, Data Compression, Deduplication, Multithread Download, OS Image Streaming, Bandwidth utilization

Received: Received 7 September 2019, Revised 30 November 2019, Accepted 4 December 2019

© 2020 DLINE. All Rights Reserved

1. Introduction

This project is an effort to reduce the time taken to stream the Operating system image file across a network in a virtual desktop environment. This requirement also comes up in cloud dependent systems where most of the computations are carried on the cloud and the large amount of data has to be transferred between the client and cloud. Modern applications are mostly based on cloud servers which dynamically take care of load balancing, user experience, scalability and many more benefits that cloud computing has to offer. Organizations are nowadays migrating towards cloud based applications for these reasons. They don't have to worry about managing users on their servers, they can simply deploy their applications on cloud and focus on core business logic. Our proposed system to transfer OS .iso files improves existing technologies by using a set of operations on the file. We first compress it, then remove redundancies and transfer it on multithreaded client. The whole process is carried over a TCP connection because file transfer required a connection oriented service.

Thin-client systems are speedily gaining quality in both wired and wireless communication environments and in massive and medium-sized enterprises. Additionally, they are being used by enterprises, they're accustomed to deliver delay sensitive services together with academic tools, gambling platforms, and extra process power for computing complicated functions. During this form of computing model, the server is to blame for process user input and application knowledge whereas the thin-client terminals render the server response and forward the user input to the server. There are various benefits for using thin-client systems like reduced info technology (IT) prices, easier IT management of devices, and reduced overall possession prices of devices. A thin-client systems service supplier are to blame for maintaining the computer code and hardware, thereby reducing the user's expenses. As an example, the user doesn't have to upgrade any hardware to accommodate new computer code needs. Thin-clients were at the start meant to be used over native space networks (LAN) solely, but recent usage trends show totally different usage. There are several thin-client platforms on the market to satisfy user wants, whether or not they are business specific, gambling connected, or strictly academic. These platforms embrace Virtual Network Computing (VNC), X-Window System, and Gaming. Thin-client systems are centralized computing models wherever the process, running, and storing of information is shifted to a foreign location. The user accesses the info remotely employing a device referred to as a thin-client. The term thin-client indicates that the user terminal solely contains the vacant minimum of technology to access the remote server. A thin-client conjointly takes the user input via keyboard or alternative input devices and transmit them to the server. The server reciprocally can send the user screen updates in varied formats.

1.1 Comparative Study of Compression Algorithms

As reducing the data is an important part of this project we first study some of the well-known and widely used compression algorithms.

	Huffman	**LZ77**	**LZW**	**LZMA**
Compression Methodology	Binary codes assigned to the data file characters such that the most common characters have shortest binary codes and vice-versa.	Words repeated are encoded as a pointer to previous occurrences of the same words, along with certain number of characters to be matched.	Modification of LZ77, uses pointers to words in a dictionary of words or parts of words in a text file.	The output from the dictionary compression algorithm used by LZMA is encoded with a range encoder which uses a complex model to make a probability prediction of each bit.
Compression Ratio	High	High	Average	High
Type (lossy / lossless)	Lossless	Lossless	Lossless	Lossless
Suitable Scenario	Optimal for a symbol by symbol encoding with a known input probability distribution for the symbols.	Better used for encoding text as they capture the higher order relationships between words and phrases.	Modification of LZ77, can handle text data files with better speed. Fails when file size is large.	Used when compression ratio is critical where we can trade the compression speed to achieve it
Complexity	O (nlogn)	O (n)	O (n)	O (n)
Compression speed	Relatively faster because of usage of static code word.	Faster as pointers are used, but large text makes the compression complicated and makes it slower.	Slower when large text files are processed.	Its slower as it gives higher compression ratios.

Table 1. Comparison of Compression Algorithms

1.2 Data Deduplication Technique

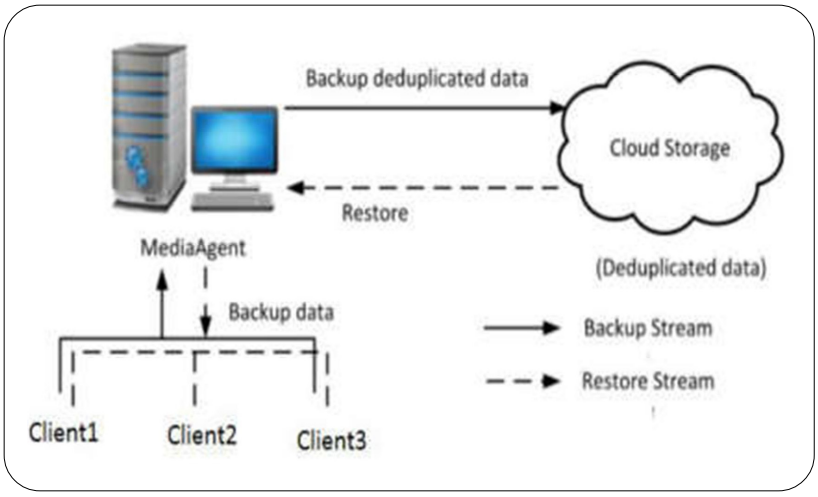

Figure 1. Direct Deduplication to Cloud Storage

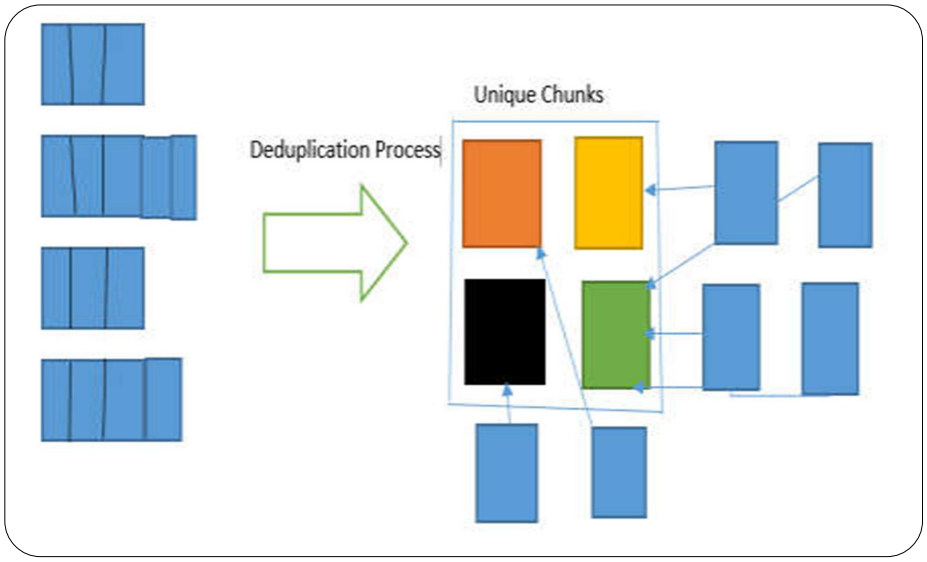

Figure 2. Deduplication Process

Data deduplication is used to optimize the file storage in case of large volumes of data. Deduplication of data can be used to eliminate the redundant data across several files and thus store only data that is unique. This helps in saving of storage volumes. It is to be noted that there is absolutely no loss of data in this process. The process starts of by dividing the data into several chunks, which can be in one file or across multiple files. The chunks are then compared with the help of their hash values. Only one copy of each unique chunk needs to be maintained and the duplicate chunks can just point to the reference of the first chunk. The files can then be replaced only with reference points of each chunk, thus reducing the files size. The chunk size is an important factor in obtaining effective results. If the chunks are too large, then there would be negligible duplicate chunks. Similarly if the chunks are too small, there would be too many chunks to manage and hence the overall volume would not be affected significantly. When new files are added to the volume, they are not optimized right away. Only files that have not been changed for a minimum amount of time are optimized. Whenever a backup needs to be performed, then only those chunks can be updated in which there have been changes, instead of transferring the entire data. This is particularly useful in Cloud Storage where data is transferred to the storage target over WAN. It will not only reduce the storage requirements, but if performed before the data is transferred over the network it will also significantly save the network bandwidth resulting in faster transfers.

1.3 Multithreaded Downloading

Often download rates are much lower than the actual overall bandwidth available, because the download rates per connction are capped at times. So if many connections can be used to download a file instead of a single connection, then it would significantly increase the download speed. This can be done by creating multiple threads and each thread would access a specific part of the file at a time. Thus multiple parts of the file would be downloaded at the same time and stored in a buffer. Once transfer of all the parts has been completed, the segments are arranged according to sequence numbers to get back the file. The number of threads however should not be too large as it would cause an overhead on the process. An optimal number of threads would be equal to the number of processors or twice the number of processors.

2. Related Works

Several models have been implemented to improvise streaming in a virtualized thin client environment to provide a satisfactory user experience at the client side when accessing remote desktop applications.

The identification of replicase in database is fundamental to improve the quality of the information processed. Deduplication is the technique of data reduction by breaking streams of data down into very granular components, and storing only the first instance of data items on the destination media and all the other similar occurrences to an index. Algorithms like Fixed size chunking (FSC) which breaks the data into fixed size chunks or blocks from the beginning of the file, Dual Side Fixed size chunking (a modified version of FSC) , Content defined chunking (CNC) which is a variable block sized technique limited to an expected chunk size parameter, parallel deduplication algorithm called FERAPARDA have been implemented to achieve better performance in a virtualized environment, all targeted on data deduplication [1] [2].

Multinode downloading protocol specifies the synchronous downloading of Distributed Virtual Environment (DVE) world from the multiple downloading nodes and renders them into whole world on the user node. This protocol is different than all the other DVE downloading methods presented before as it makes use of all grid resources and improves speed of interacting and downloading the DVE world [8].

Techniques like Run Length Encoding, LZMA (Lempel-Ziv-Markov chain) algorithm, Huffmann Coding and ORC (Optimized Row Columnar) in HDFS for higher compression ratio and better IO output have been implemented for High performance Big Data Load, hence improving efficiency in a virtualized environment [3] [4].There are several other protocols where a combination of Virtual Network Computing (VNC) and streaming protocols have been used to allow efficient remote web access to virtualized applications within a cloud architecture.

3. System Architecture

Our proposed system has client-server architecture where each client and server has some layers as well. The reason for choosing this type of architecture are many fold. Firstly, this represents a distributed system. All the node (subsystems) run the same piece of software and anyone can be a server or client dynamically. In a typical cloud-client scenario there is one server connected to multiple thin clients. It is to be noted that there is no peer to peer connection here. Data compression, deduplication and redundancy removal all takes place at server side. Thin clients are usually installed with device manager softwares that handle boot agents, agents that request data and agents that handle control instructions. The product will be a single piece of code installable on all network nodes that contains client service which will run continuously and GUI for using server modules of sending compressed data.

4. Implementation

Before we describe the algorithms for detailed implementation of the product, we present a flowchart of the product lifecycle. Data which is in unstructured format is stored on the repository located along with the servers on the cloud. This data is compressed and deduplicated using LZMA algorithm. It is to be noted that the deduplication here is on each file. Now they are stored on the cloud repository. When the thin client sends the request to transfer the file it will be sent through the network and on the client side it is downloaded using our multithreaded downloading concept. We have written a console application for the server side and Windows Forms application having a proper GUI on the client side. Figure 4 represents the overall process and features in a flow chart.

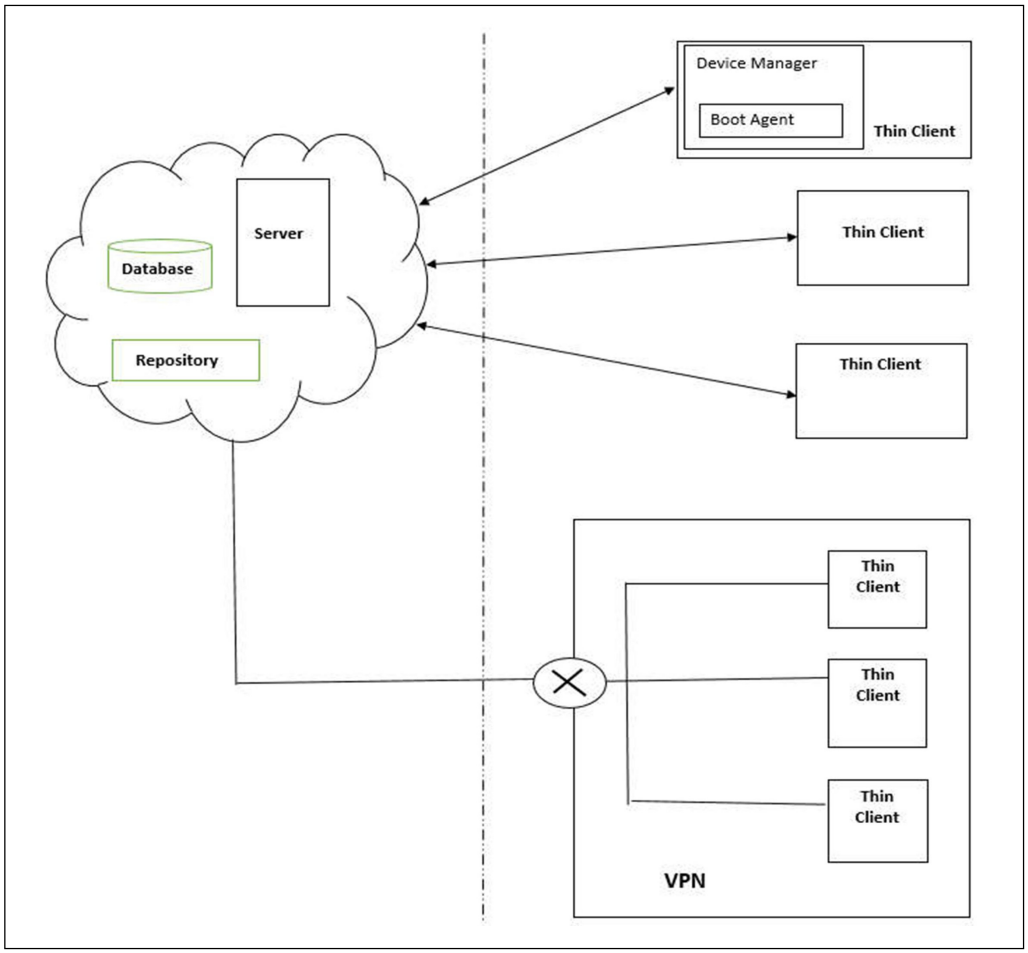

Figure 3. Architecture of Proposed System

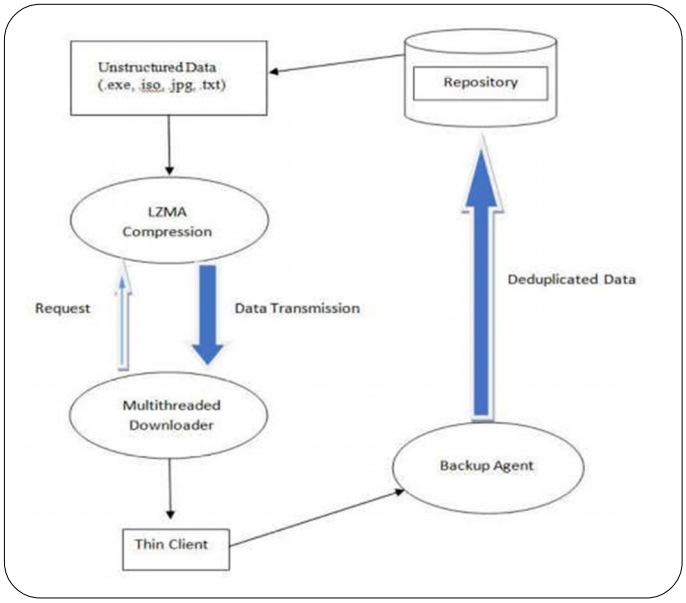

Figure 4. Application Flowchart

The compression of data and other CPU intensive operation take place at the server side of the application while client is only responsible for data synchronization with the server and following control instructions. The flow chart above explains our approach in a high level. The LZMA algorithm has steps described below after which we get a compressed version of the OS image or application data or other files. But the file still has redundancies which are removed by further deduplication of data.

LZMA works on the dictionary form of data. It iteratively tries to find if the current character in the input stream has been encountered earlier in the dictionary. If yes take it from the dictionary and output otherwise add to the dictionary. But whydeduplication is needed is that only single characters are used to find repeated patterns. If there is a whole block that is repeated at certain distance apart in the input then it still takes two spaces in the memory. That's where deduplication comes into picture. This algorithm will then take a pass in the data and find redundant pieces of data and add pointer to data at second finding of the data.

4.1 Algorithm for Multithread Download

Input: URL to download the file, local path to save the file.

Start

1. Perform a check on the input URL to see if the following conditions are satisfied
   1.1 It is a valid accessible URL.
   1.2 The file exists.
   1.3 The size of the file is more than 0.
   1.4 The remote server supports "Accept-Ranges" header.

2. If the test is passed, get the path to store the downloaded file and begin the download.

3. Else if the test is not passed because "Accept-Ranges" header is not supported by remote server, then the file cannot be downloaded through multiple threads. Download it normally through a single thread.

4. On starting download, a HttpDownloadClient object is created and the following properties StartPoint, EndPoint, BufferSize, MaxCacheSize, BufferCountPerNotification, Url DownloadPath and Status are initialized.

5. The HttpDownloadClient creates threads equal to twice the number of processors. Each download thread will read a buffer of bytes from its StartPoint till the EndPoint, and store the buffer to a MemoryStream cache first.

6. Raise the event DownloadProgressChanged when read a specified number of buffers.

7. If the download is paused, each HttpDownloadClient will store the downloaded size. When it is resumed, start to download the file from a start point.

8. Update the used time and status when the current download stops.

9. Raise the event DownloadCompleted when the download is completed or canceled.

End

4.2 Algorithm for Backup Deduplication

Input: Path to the directory whose files and sub-directories are to be deduplicated.

Output: Hash Table of the files where keys are the SHA1 indexes and values are data blocks.

Start

1. Search the given directory for file names and sub directories also.

2. Declare block size depending on requirement. Small block size saves more data but results in less performance and large block size give less compression but faster performance.

3. Declare a file pointer and hash table with specific block parameters and loop through all the files. For each
block do:

a) Compute SHA1 hash using standard library function.
b) if hash key already exists then increment dupe count
c) else add new entry in hash table with value as SHA1 hash. and key as index.

End

---

5. Performance Analysis

***assuming 1Mbps = 128 KBps = 0.125 MBps connection speed in average.**

Data	Normal Transfer Time	Compressed Size	Multithreaded Transfer Time
1 MB pdf	8s	0.75 MB	0.21s
10 MB pdf	93s	8.92 MB	12s
100 MB pdf	1081s	84.9 MB	142s
212 MB Multitype data (doc, pdf, ppt)	2234s	168 MB	332s
1096 MB Ubuntu .iso file	8768s	1075 MB	1753s

Table 2. Results of Applied Algorithms

This analysis depends on network congestion, packet loss, delay, jitter and other network parameters that cannot be remv ed in real networks. For experiment purpose we assumed that normally we get a constant internet speed which is never true. But the factor by which performance is improved is significant.

6. Conclusion

This product can be used in all those applications that use or depend on the cloud. The scope of extension of this project is huge. It can be integrated as a service in native operating systems to efficiently utilize the network. Mobile devices have limite applications on cloud when the mobile processing capability is limited. As a future project we can develop a mobile device that only runs the hypervisor and uses a virtual OS instead of android or ios and runs all the applications on the cloud. This is not

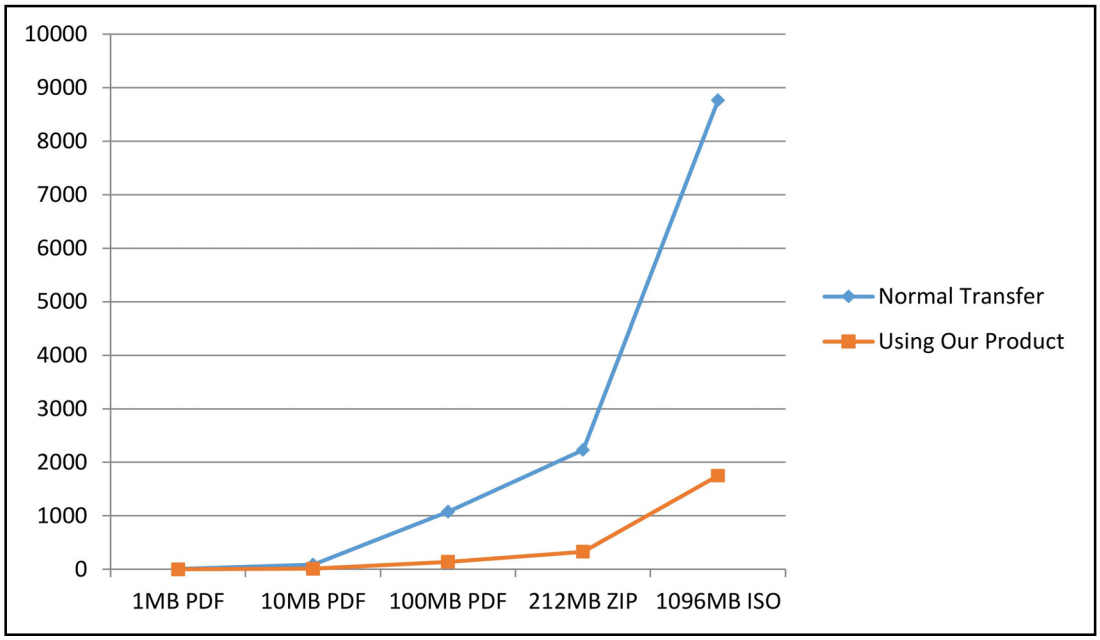

Figure 5. Performance Graph

possible now because there is no system that provides network optimization and bandwidth utilization as our product does. The results of our project have shown that this cascaded processes are providing sufficient compression ratio that are significant enough that can be used in mobile tcp/ip environment. Apart from this, individual clients and organizations can use thin clients efficiently and get all the benefits that cloud computing has to offer.

References

[1] Santos, Walter., Teixeira, Thiago., Machado, Carla., Meira Jr., Wagner., Altigran, S., Silva, Da., Ferreira, Renato., Guedes, Dorgival. (2007). A scalable Parallel Deduplication Algorithm, 19th International Symposium on Computer Architecture and High Performance Computing.

[2] Krishnaprasad, P. K., Narayamparambil, Biju Araham. (2013). A proposal for Improving Data DeDuplication with Dual Side Fixed Size Chunking Algorithm, 2013 Third International Conference on Advances in Computing and Communications.

[3] Zhang, Liping., Chen, Qi., Miao, Kai. (2014). A Compatible LZMA ORC -based Optimization for High Performance Big Data Load, 2014 IEEE International Congress on Big Data.

[4] Li, Bing., Zhang, Lin., Zhuangzhuang., Dong, Qian. (2014). Implementation of LZMA compression algorithm on FPGA, *Electronics Letters*, 50 (21) 1522-24.

[5] Sharma, Anupam., Singh, Bhupender., Kumar, Rishi. (2016). Versatile Web Based Thin Client OS using Cloud Services by using concept Cloud Driving, 6th International Conference-Cloud System and Big Data Engineering.

[6] Zhang, Yucheng., Feng, Dan., Jiang, Hong., Xia, Wen., Fu, Min., Huang, Fangting., Zhou, Yukun. (2017). A Fast Asymmetric Extermum Content defined Chunking Algorithm for Data Deduplication in Backup Storage System, IEEE Transactions on Computers, 66 (2), (February).

[7] Shamieh, Fuad., Wang, Xianbin. (2016). Novel Lightweight Multicasting Protocol for Thin-Client System, 978-1-5090-0304-4/16/$31.00 ©2016 IEEE.

[8] Wang, Yuying., Lu, Guanghua, Jia, Jinyuan., Yang, Hongji. (2007). An Architecture to Support Multi-node Downloading of Distributed Virtual Environment on Grid, *In*: Proceedings of the 2007 IEEE International Conference on Networking, Sensing and Control, London, UK, 15-17 (April).