Applied Mathematics & Information Sciences
An International Journal

http://dx.doi.org/10.12785/amis/092L20

Streaming Task Distribution Method for Reliable Distributed Streaming Service in Cloud Environment

Myoungjin Kim[1], *Seungho Han*[1], *Yun Cui*[1] *and Hanku Lee*[2,*]

[1] Department of Internet and Multimedia Engineering, Konkuk University, Seoul, Korea
[2] Center for Social Media Cloud Computing, Konkuk University, Seoul, Korea

Received: 20 May 2014, Revised: 21 Jul. 2014, Accepted: 22 Jul. 2014
Published online: 1 Apr. 2015

Abstract: We previously proposed a cloud-based distributed multimedia streaming service (CloudDMSS) that delivers rich multimedia streaming services to heterogeneous devices in cloud environments with unstable and unpredictable traffic. The proposed system deals primarily with task management, load balancing, and fault tolerance. In this paper, we focus on improving the streaming task distribution capacity of our proposed system by providing a streaming resource-based connection (SRC) algorithm for use by the cloud management module (CMM) of CloudDMSS. The main function of the SRC algorithm is to select a streaming server with optimal conditions for streaming services among streaming servers deployed in our system by considering factors such as streaming network traffic and CPU and RAM usage rates. This enables our system to reduce content delays and avoid traffic bottlenecks while providing streaming services. The results of simulations conducted on both our local testbed and an actual cloud computing environment, Cloudit 2.0, with 600 virtual user requests for streaming service, indicate that the SRC algorithm enables our system to more effectively distribute streaming tasks than conventional Round Robin and Least Connection-based systems.

Keywords: Media service, distributed streaming, task distribution, cloud computing, Hadoop

1 Introduction

The recent advent of high-performance heterogeneous smart devices, and the spread of social networking services (SNS), such as Facebook and Twitter, has resulted in large volumes of social media content being produced in various forms and being shared in wired and wireless environments [2]. Recently, the most noteworthy and influential emerging technologies include cloud-based media streaming, transcoding, and content distribution to deliver rich multimedia services with guaranteed quality of service (QoS).

Previously, traditional distributed and cluster-based computing approaches were utilized for media transcoding and streaming processing. However, it is difficult to provide rich media services incorporating multimedia traffic and mobile services using traditional approaches [14]. In a previous paper [4], we presented a distributed multimedia streaming service (CloudDMSS) system designed to run on conventional cloud computing infrastructure. CloudDMSS comprises three main modules: a Hadoop-based distributed multimedia transcoding module (HadoopDMT), a Hadoop-based distributed multimedia streaming module (HadoopDMS), and a cloud management module (CMM).

Because of the varied resolutions of video extensions, codecs, and streaming protocols, content generated by users has to be transcoded into a format suitable for streaming to smart devices. Traditional multimedia transcoding approaches focused on distributed and cluster-based video media approaches, such as those found in [3,5,15,18,20], which reduce processing time and maintenance costs when building a computing resource infrastructure. However, these approaches procure computing resources for the video transcoding process simply by increasing the number of cluster machines in the distributed computing environment. In addition, they do not consider load balancing, fault tolerance, and data replication methods to ensure data protection and to expedite recovery.

We overcome these limitations by executing HadoopDMT in a cloud computing environment and improve quality and speed by using the Hadoop Distributed File System (HDFS) [12] to store the large

* Corresponding author e-mail: hlee@konkuk.ac.kr

volumes of video data created by users, MapReduce cite12,13 for distributed processing of the video data, and Xuggler [16] and Java Advanced Imaging (JAI) [17] for media transcoding. In addition, our system achieves improved distributed processing capabilities and simplified system design and implementation by incorporating the data replication, fault tolerance, load balancing, and file splitting and merging policies provided by Hadoop. After completing the transcoding process in HadoopDMT and storing transcoded content in HadoopDMS, the CMM in CloudDMSS manages and distributes streaming tasks via distributed streaming servers to provide a seamless streaming service to users without content delay.

In a previous paper [8], we proposed two streaming task distribution methods based on two algorithms: Round Robin (RR) and Least Connection (LC). Recently, researchers have proposed various approaches [6,7,8,9] for effectively distributing streaming tasks in a distributed system environment when numerous requests for streaming services are generated by users. However, systems that apply RR and LC do not consider physical resource usage (CPU and RAM) and streaming network traffic; consequently, they are limited and impose a heavy burden on current Internet infrastructure and streaming servers. In this paper, we propose a streaming resource-based connection (SRC) scheduling algorithm that considers the CPU, RAM, and streaming transmission rate usage of each streaming server; thereby addressing the limitations of the RR and LC distribution methods. In addition, our system introduces a redirection mechanism [10] for HTTP requests.

Distributed servers that include content in their system lack countermeasures against data losses and system failures. Consequently, the number of distributed servers has to be increased in order to create replicas of the data to compensate for failures. However, the reliability and scalability of these systems are not guaranteed because of the absence of automatic recovery policies. Our system obviates data losses and system failures by adapting and incorporating the structure and policies of Hadoop.

In this paper, we compare the streaming task distribution capacity of RR-, LC-, and SRC-based systems in terms of the network transmission distribution throughput based on the streaming servers in a local testbed. In addition, to verify the performance in commercial cloud computing environments, including unpredictable and unstable network traffic over WAN, we test the streaming task distribution capacity in Cloudit 2.0 [19], which is operated by Innogrid.

The remainder of this paper is organized as follows. In section 2, we discuss load distribution methods and Hadoop. Section 3 provides a detailed explanation of CloudDMSS. The proposed algorithm and its core concepts are described in section 4. In section 5, we present our proposed prototype of a distribution streaming system on CloudDMSS and its system architecture. In section 6, we compare our distribution streaming system with the RR and LC algorithms in terms of transmission rate. Finally, section 7 presents our conclusions and outlines the scope for further research.

2 Related Work

2.1 Load and Task Distribution Methods

Load and task distribution methods that reduce the idle time and that minimize redundant or wasted effort in distributed environment are used in many domains such as web servers, file servers, and video streaming servers. The RR and LC algorithms are used to distribute loads and tasks among distributed servers. Indeed, most current distributed server platforms typically use some form of RR or weighted RR, followed by LC. First, the RR-based method distributes tasks in a fixed order according to the user requests. Second, the LC-based method distributes tasks to the server with the lowest number of users according to the task request. In addition, many researchers have studied other distribution algorithms, such as weighted RR (WRR) and dynamic weighted RR (DWRR) [6]. However, because most of the methods for task distribution target web servers and they are generic, they are very inefficient and inappropriate when applied to a distributed server platform for media streaming, which requires intensive server resources. Therefore, we propose a task distribution method that uses the SRC algorithm. The SRC algorithm considers the usage rate of servers and the streaming transmission throughput when distributing a streaming task, thereby allowing it to overcome the limitations of existing task distribution methods.

2.2 HDFS and MapReduce

Hadoop, inspired by Googles MapReduce and Google File System [6], is a software framework that supports data-intensive distributed applications, which are capable of handling thousands of nodes and petabytes of data. Hadoop facilitates the scalable and timely analytical processing of large datasets to extract useful information. Hadoop comprises two important frameworks: 1) HDFS [11], which is a distributed, scalable, and portable file system written in Java, like the Google file system (GFS); and 2) MapReduce cite12, which was the first framework developed by Google for processing large datasets.

The MapReduce framework provides a specific programming model and a runtime system for processing and creating large datasets that are amenable to various real-world tasks. This framework also handles automatic scheduling, communication, and synchronization when processing huge datasets and it is fault tolerant. The MapReduce programming model is executed in two main steps: mapping and reducing. Mapping and reducing are

defined by mapper and reducer functions. Each phase uses a list of key and value pairs as inputs and outputs. In the mapping step, MapReduce receives the input datasets and feeds each data element to the mapper in the form of key and value pairs. In the reducing step, all of the outputs from the mapper are processed and the final result is generated by the reducer using the merging process.

2.3 Redirection Mechanism for HTTP Requests

Mourad et al. [10] proposed a redirection-based hierarchical web server architecture where a redirection mechanism facilitates the load distribution for HTTP requests. Figure 1 shows the redirection-based hierarchical web server architecture.

The categorized content is stored in distributed servers. The architecture comprises two component levels: redirection servers and normal HTTP servers. Each HTTP server is categorized and it stores a portion of the data that is available on the site. The RR DNS selects a redirection server in order to provide a requested service using the RR algorithm when users request specific content for their system. Subsequently, the selected redirection server distributes a task that connects the user request with the HTTP server that includes the required content.

Each file has a unique base URL (e.g., www.lucent.com/news/today.html), such as a domain name. A user employs the base URL to connect to the redirection server that maps the base URL to the target URL, which is returned to the user in a redirection message. Next, the user employs the new URL to connect to the HTTP server to obtain the specific content. Figure 2 shows the redirection mechanism when an HTTP request is sent to the redirection server.

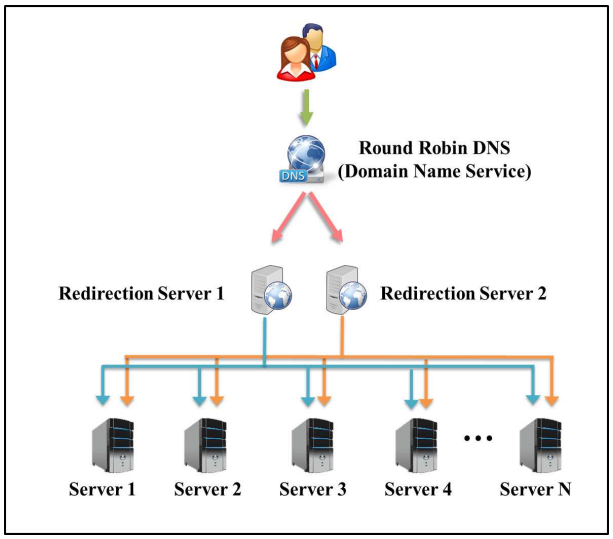

Fig. 1: Hierarchical redirection-based web server architecture.

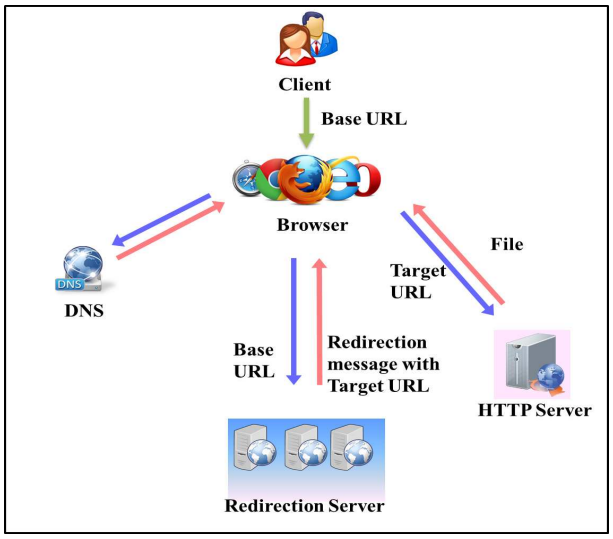

Fig. 2: Redirection mechanism for HTTP requests.

3 Brief Overview of CloudDMSS

In this section, we provide a brief review of CloudDMSS [4], where we describe the fundamental concept of our service model. Figure 3 shows the concept of our service model.

Personal media data such as movies, music videos, and animations are distributed and stored by a transcoding Hadoop cluster. Users and administrators upload media data for transcoding to share the data with other users. After uploading, the media data are transcoded into a standard format (MPEG4), which is suitable for streaming to heterogeneous devices. To reduce the transcoding time, our model applies a transcoding function that utilizes the MapReduce framework. The transcoded contents are then migrated automatically and stored on the content servers of a streaming Hadoop cluster. The migrated contents are streamed to the end users with a guaranteed QoS by controlling the streaming servers that run on the streaming Hadoop cluster. To reduce content delays and avoid traffic bottlenecks, our service model utilizes a streaming job distribution algorithm, which balances and distributes the load of the streaming servers.

Our proposed CloudDMSS is designed to run on a Hadoop cluster that streams media content to heterogeneous devices in a distributed manner. The overall system architecture is shown in Fig. 4. CloudDMSS has three main components: HadoopDMT, HadoopDMS, and CMM. The main characteristics of our system are as follows. (1) Our system transcodes large volumes of media content into the MPEG-4 video format for delivery to a variety of devices, including PCs, smart

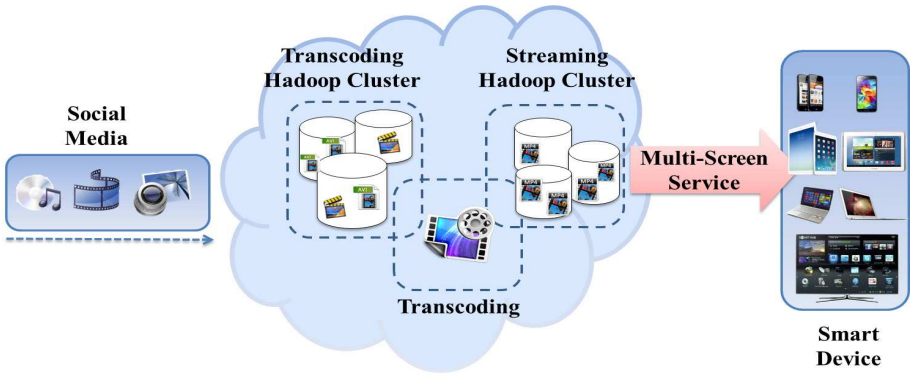

Fig. 3: Fundamental concept of the cloud-based distributed multimedia streaming service model.

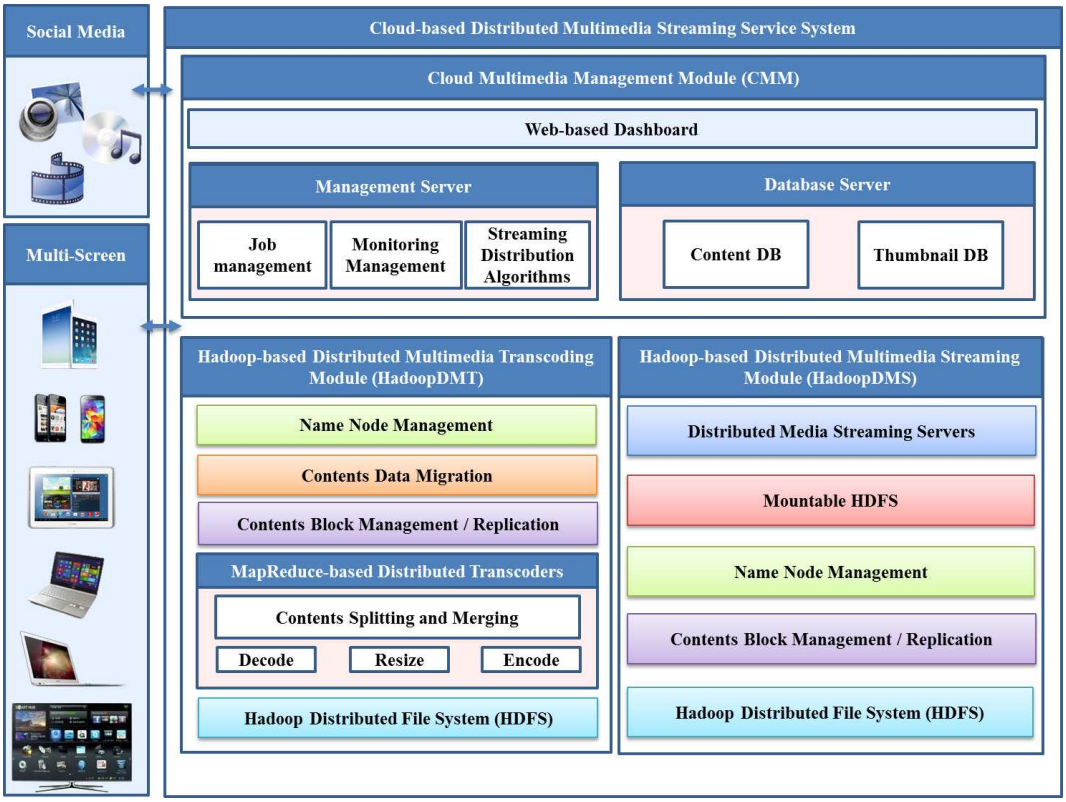

Fig. 4: Overall system architecture of CloudDMSS.

pads, and smartphones. (2) It reduces the transcoding time by using HDFS for multimedia data storage and MapReduce for distributed parallel processing. (3) CloudDMSS controls streaming servers to reduce content delays and avoid traffic bottlenecks by using a streaming job distribution algorithm. (4) Dual-Hadoop clustering on each physical cluster is used to improve the overall performance and to distribute job tasks between transcoding and streaming. (5) CloudDMSS provides efficient content distribution and improved scalability by adhering to Hadoop policies. (6) It automatically conducts the workflow of sequential tasks for a streaming service deployment process.

4 Streaming Task Distribution Method with SRC

In this section, we present a streaming task distribution method where the SRC algorithm is used by the CMM

component. The most important benefits of CloudDMSS are reduced content delays and the avoidance of traffic bottlenecks in cloud computing environments with unstable and unpredictable traffic by using RR-based and LC-based methods. However, these two algorithms are not suitable for processing streaming services in distributed computing environment because they do not consider the elements related to intensive and massive physical resource usage, i.e., the CPU, memory, network bandwidth, and storage required for streaming processes. In addition, the quality of a streaming service is determined by fluctuating network conditions, thus task distribution methods that only consider invariable elements are unsuitable. Therefore, these algorithms are only suitable for static distributed computing environments, where they can process tasks based on simple text data and web pages that do not require intensive resources. Therefore, we propose a streaming task distribution algorithm that uses the SRC algorithm to consider variable elements.

HadoopDMT transcodes collected social media content into a target format that is suitable for heterogeneous devices. Next, the transcoded contents in HadoopDMT are migrated and stored by the content storage servers of HadoopDMS. As a core component of CloudDMSS, CMM comprises a web-based dashboard module, management server module, and database server module. The web-based dashboard module provides an interface that allows users to select the options required for transcoding tasks, such as the resolution, bit rate, and frame rate, as well as monitoring the usage rate of the dual Hadoop cluster and streaming servers in HadoopDMS (CPU, RAM, and streaming transmission rate). The management server controls and conducts the scheduling of the overall process such as transcoding, migration, extracting thumbnail images, registering the images and information related to the transcoded content to the database server module of CMM, and job distribution. The content on content storage servers is streamed by distributed streaming servers in HadoopDMS. The core function of the management server is to effectively balance and distribute any rapidly increasing streaming tasks, while maintaining the simultaneous connections of multiple users.

We describe the SRC algorithm in detail because RR and LC have been described in many previous studies. The RR streams media content by selecting streaming servers in a prescribed order after video streaming requests are received from users. In LC, the media content is streamed by selecting the streaming server with the lowest number of streaming tasks. However, systems that apply RR and LC do not consider the CPU and RAM utilization rate and network transmission throughput; thus, they are limited because they impose a heavy burden on the current Internet infrastructure and streaming servers. Thus, we introduce an SRC scheduling algorithm that considers the CPU, RAM, and streaming transmission rate usage of servers, which resolves the limitations of the RR and LC distribution methods. The algorithm considers the CPU, RAM, and streaming transmission throughput for each streaming server. The Linux command mpstat is used to generate statistics for the CPU usage servers. The free command is used to determine the RAM usage and /proc/net/dev is used to obtain the streaming transmission rate. The streaming server completes a streaming task request from a user according to the SRC algorithm shown in Table 1.

Table 1: Streaming Resource-based Connection Algorithm

Algorithm1

Data: ss_n_ID: id for streaming server 1, server 2, ... , server$_n$
Data: ssu_n: system usage rate of ss_n
Data: scu_n: CPU usage rate of ss_n
Data: sru_n: RAM usage rate of ss_n
Data: stt_n: streaming transmission throughput of ss_n
Data: $request\_ID$: id for requesting a streaming service

```
1: while system available
2:    if request for streaming service then
3:      for(each ss_m_ID)
4:        calculate_system usage (ss_n_ID){
5:        calculate ssu_n ;
6:        calculate stt_n ;
7:        }
8:      end for
9:
10:     select streaming server (request_ID){
11:       if the number of the smallest ssu_n == 1 then
12:         set server(request_ID, the smallest ssu_n);
13:         start_service (request_ID, content);
14:       else
15:         set server(request_ID, the smallest stt_n);
16:         start_service (request_ID, content);
17:       end if
18:     }
19:   end if
20: end while
```

The parameters of the SRC algorithm are as follows. ss_n_ID is the id for streaming server in HadoopDMS. Let $ss_n\_ID = (ss_1, ss_2, ..., ss_n)$ be the collection of all streaming servers. ssu_n is the system usage rate for ss_n, which is calculated by adding the CPU and RAM usage rates. The CPU usage rate of ss_n calculated by the Linux command *mpstat* is defined as scu_n. The RAM usage rate of ss_n calculated by the command *free* is represented as sru_n. stt_n is the current transmission throughput for 1 second generated in ss_n. $request\_ID$ represents the id of a user requesting for streaming service.

To connect optimal ss_n and the users streaming request, the algorithm calculates two values for each streaming server: ssu_n and stt_n, respectively. ssu_n is calculated by Eq. (1).

$$ssu_n = \frac{scu_n + sru_n}{2} \quad (1)$$

In addition, stt_n is calculated by Eq. (2). *current* stt_n is the total throughput of *ssn* since the start of the streaming service. *past* stt_n represents the total throughput in *ssn* before 1 second of the current measurement point.

$$stt_n = current\ stt_n - past\ stt_n \quad (2)$$

The streaming task distribution sequence used to select the idlest streaming server is as follows. First, the management server in CMM distributes the current streaming task to the streaming server with the lowest value for ssu_n. If there are two or more servers with the lowest value for ssu_n, the server distributes the task to the streaming server with the lowest value for stt_n. Thus, it is possible to provide a seamless and rich media streaming service via a streaming server with the optimal conditions in a cloud computing environment with unpredictable network traffic.

5 Implementation of a Distributed Streaming System Using SRC

In this section, we describe the implementation details for a distributed streaming system, including a streaming task distribution method based on the SRC algorithm. The distributed streaming system was implemented on our local testbed, while considering the design issues of CloudDMSS.

5.1 Implementation Environment

The distributed streaming system comprises three modules: a streaming server module (SSM) that acts as a distributed streaming server in HadoopDMS, a HDFS-based content storage server module (HSCSM) that acts as a content storage server in HadoopDMS, and a streaming task management module (STMM) that acts as a management server in CMM. In this implementation, we exclude the implementation of HadoopDMT to transcode the original content into target format because we focus on demonstrating the feasibility of the streaming system with the SRC algorithm under the assumption that HadoopDMT completes the transcoding process.

For the first prototype of the distributed streaming system, we deployed 14 nodes that ran on our own private cloud computing cluster. Each node had the hardware specifications shown in Table 2. To implement SSM, three nodes were designated as the streaming server running on NginX [21] to stream content that was transcoded into the MPEG4 video format on HSCSM. To implement the HSCSM, 10 nodes were designated on HDFS. To implement the STMM, one node was designated as our management server for streaming task distribution using SRC. STMM uses JSP to implement a web-based dashboard, swfupload 2.2.0.1 libraries [23] for the content upload function, a Java library and bash shell script to run the SRC algorithm, and Google chart APIs to generate the graph showing the monitoring system status. In addition, the software specifications used to implement our system included an H.264 streaming module 2.2.7 for the streaming function, NginX 1.2.7 for the streaming servers, and fuse_dfs_0.1.0 to allow HDFS to be mounted on the UNIX system as a standard file system.

Table 2: Hardware specifications

Classification	content
CPU	Intel Xeon 4 Core 2.13 GHz
Memory	16 GB
HDD	1TB SATA-2
Network	100 Mbps Ethernet adapter
OS	Ubuntu 10.04 LTS

5.2 Configuration of the Distributed Streaming System for SRC-based Streaming Task Distribution

In this section, we present details of the system configuration and we briefly describe the three modules. Figure 5 shows the configuration of the distributed streaming system for SRC-based streaming task distribution.

5.2.1 Streaming Task Management Module (STMM)

STMM is the core component of the distributed streaming system, which plays an important role in efficient load balancing. Robust load balancing is achieved by scheduling the streaming task distribution using the SRC algorithm. First, users can select the list of contents by accessing a web dashboard in STMM using their devices. Second, when a user chooses their specific content, STMM determines the optimal streaming server in SSM by calculating the usage rate of each streaming server and the current streaming transmission throughput. Finally, users connect to the selected server via the redirection

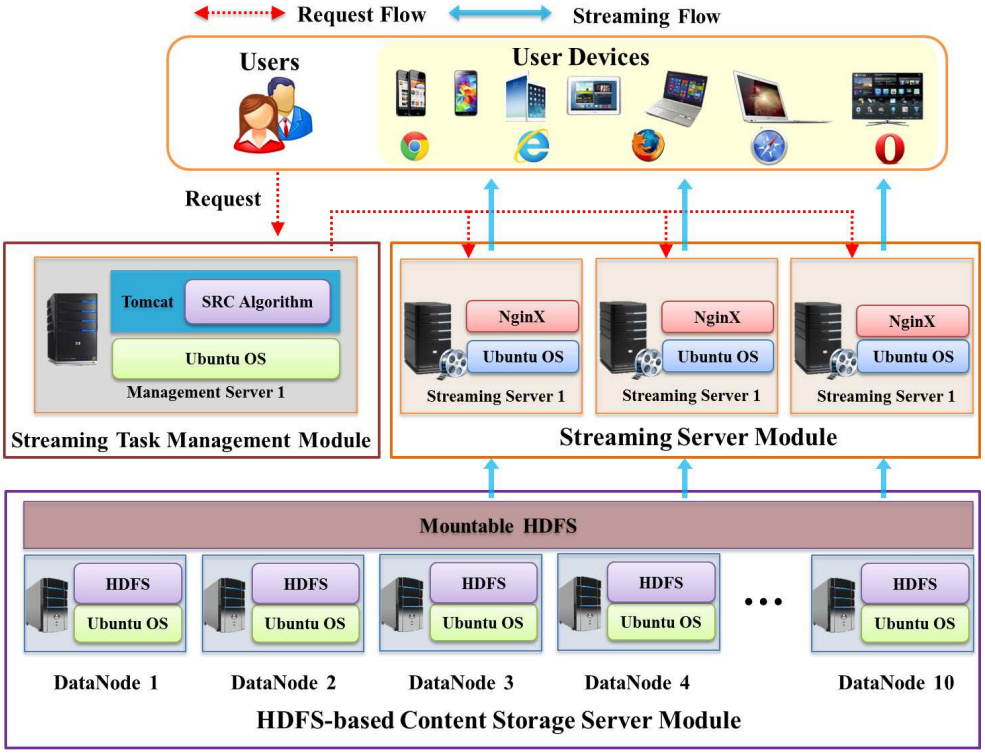

Fig. 5: Configuration of the distributed streaming system for SRC-based streaming task distribution.

mechanism for HTTP requests and they then receive their requested streaming service.

5.2.2 Streaming Server Module (SSM)

SSM includes streaming servers that are scheduled by the SRC algorithm in STMM. Whenever the number of the servers in the SMM increases, it is possible to provide rich multimedia streaming service to users because the number of streaming tasks processed per server is reduced.

However, the number of servers should be determined by considering the total network throughput in the system and the physical configuration of the content storage servers in HSCSM, rather than simply increasing the physical resources without a specific reason. Each streaming server utilizes NginX with an H.264 streaming module installed. NginX with an H.264 module, a typical web server, is used so an HTTP-based streaming service can be provided, instead of a streaming server with an exclusive streaming protocol, i.e., RTP, RTSP, or RTMP. In general, a specific streaming protocol is selected according to the type of target devices when a streaming system is constructed. However, an HTTP-based streaming protocol is the best and easiest because it allows streaming services to be provided to most devices.

5.2.3 HDFS-based Content Storage Server Module (HSCSM)

The main role of HSCSM is to store transcoded content on data nodes in a distributed manner and to provide the capacity for content replication and recovery if data node failures or content loss occurs. The most important factor when providing a streaming service is the maintenance of a seamless streaming service, which requires that users cannot recognize data node failures and the loss of content. A streaming service system that uses the traditional approach does not include content replication management and automated recovery policies, thus the system reliability and seamless streaming service cannot be guaranteed. HSCSM overcomes these limitations by including the core function and policy of Hadoop. In addition, mountable HDFS [24] is applied to our system to allow HDFS to be mounted on a local file system. Thus, users can access a distributed file system with the complexity of the use as a local file system by using general commands.

6 Performance Evaluation

The proposed streaming task distribution method with the SRC algorithm was implemented in the distributed streaming system. In this section, we present our

verification of the systems capacity for streaming task distribution when users generated numerous requests for streaming services.

6.1 Methodology and Results with a Local Testbed Environment

We tested the streaming task distribution function using the first prototype of our distributed streaming system. We simulated the systems performance in two different environments: our local testbed and a public cloud computing environment.

Table 3: Average transmission throughput per streaming server using each algorithm-based system

Streaming Server	RR-based System	LC-based System	SRC-based System
Server 1	2.80 MB/s	2.76 MB/s	3.25 MB/s
Server 2	2.84 MB/s	2.66 MB/s	3.37 MB/s
Server 3	2.80 MB/s	2.64 MB/s	3.32 MB/s
Average	2.81 MB/s	2.69 MB/s	3.31 MB/s

First, the system configuration described in Section 5 was used as the local testbed. To validate the performance of the SRC algorithm, we compared our system with RR- and LC-based systems in terms of the network transmission throughput generated by three streaming servers in the SSM. Each streaming server had a bandwidth of 100 Mbps. The performance testing tool used to calculate the average transmission throughput per streaming server per second was developed in Java. A dataset that included 4 MB MP4 files transcoded by our HadoopDMT was used. We created 600 virtual users and simulated the users as they accessed three streaming system based on three algorithms, RR, LC, and SRC. In the next step, we calculated the overall transmission throughput for each streaming server per second.

Table 3 shows the average transmission throughput for each streaming server per second. The experimental results verified that the system with the SRC algorithm delivered the best performance in terms of the average transmission throughput compared with both the RR- and LC-based systems. The high streaming throughput per second demonstrated that the streaming task distribution method with the SRC algorithm was faster than the other algorithms because it rapidly processed the 600 requests by distributing the requests to each streaming server in an efficient manner.

Table 4: Average transmission throughput per streaming server using each algorithm-based system

Streaming Server	RR-based System	LC-based System	SRC-based System
Server 1	1.18 MB/s	1.68 MB/s	1.95 MB/s
Server 2	1.31 MB/s	1.89 MB/s	1.92 MB/s
Server 3	1.23 MB/s	1.68 MB/s	1.93 MB/s
Server 4	1.24 MB/s	1.62 MB/s	2.14 MB/s
Server 5	1.22 MB/s	1.54 MB/s	2.24 MB/s
Server 6	1.27 MB/s	1.77 MB/s	1.99 MB/s
Server 7	1.31 MB/s	1.54 MB/s	2.10 MB/s
Server 8	1.37 MB/s	1.55 MB/s	1.95 MB/s
Server 9	1.22 MB/s	1.67 MB/s	2.12 MB/s
Server 10	1.18 MB/s	1.60 MB/s	1.98 MB/s
Average	1.25 MB/s	1.65 MB/s	2.03 MB/s

6.2 Methodology and Results with a Commercial Cloud Computing Environment

The main objective of our system is to provide an architecture that is designed for streaming services, which can be deployed in an actual cloud computing environment. However, the local cloud computing environment in our LAN environment did not replicate the unpredictable and unstable network traffic generated by public cloud computing environments such as Amazon EC2 and Rackspace. Therefore, to demonstrate the validity of our system for streaming task distribution in a commercial environment over WAN, we conducted a performance test using an actual cloud computing environment, Cloudit 2.0 [19], which is operated by Innogrid. In this performance test, we used the same software configuration, dataset, and simulation methodology described in Section 6.1. However, the system and hardware specifications differed from those described in Sections 5 and 6.1. We used 21 virtual machines (21 VMs), which were deployed from Cloudit 2.0 for one management server, 10 streaming servers, and 10 content storage servers. Each VM comprised Linux OS (Ubuntu 12.04 LTS) running on four virtual cores with the equivalent of an Intel Xeon quad-core 2.13 GHz processor, 8 GB of memory, and 100 GB of disk space on a shared hard drive. Cloudit 2.0 utilizes Xen [1,25] virtualization software.

We calculated the overall transmission throughput for each streaming server per second with a bandwidth of 100 Mbps. The performance test results are shown in Table 4. Our SRC algorithm delivered better performance in terms of the average transmission throughput when compared with the RR-based and LC-based algorithms. The total transmission throughput per second with the three streaming servers on the local testbed was about 10 MB,

whereas the total throughput per second using the 10 servers on Cloudit 2.0 was about 20 MB. Thus, our system with the SRC algorithm effectively distributed the streaming jobs requested by numerous users in an actual cloud computing environment with a similar performance level to that on the local testbed.

7 Conclusion and Future Research

In this study, we proposed a streaming task distribution method based on the SRC algorithm in a cloud computing environment, and we demonstrated the feasibility of this distributed streaming system by implementing a prototype in local and private cloud computing environments. The main benefit of our method is that it improves the capacity for streaming task distribution in CloudDMSS by reducing content delays and network traffic. Previous methods that employ conventional algorithms, such as RR and LC, are not suitable for distributing requested tasks to distributed servers in general streaming systems. Thus, we enhanced the efficiency of streaming task distribution by introducing the SRC algorithm, which considers the key elements that affect the processing of streaming services, i.e., the physical resource usage (CPU and RAM) and streaming network traffic.

To demonstrate the feasibility of a distributed streaming system with the SRC algorithm in CloudDMSS, we implemented the prototype in private and public computing environments. In addition, to verify the improved task distribution performance, we compared our system with RR- and LC-based systems in terms of the network transmission throughput generated by each streaming server in both cloud computing environments. The performance tests showed that the system with the SRC algorithm delivered the best performance in each environment in terms of effectively distributing the streaming tasks for numerous requests when compared with conventional approaches.

In future research, we plan to improve the method based on the SRC algorithm by considering additional elements, such as the disk I/O generated for HDFS. In addition, we plan to implement a fully functional CloudDMSS with a distributed transcoding process in a commercial cloud computing environment, such as Amazon EC2 or Rackspace.

Acknowledgement

This research was supported by the MSIP(Ministry of Science, ICT & Future Planning), Korea, under the ITRC(Information Technology Research Center) support program (NIPA-2014-H0301-14-1001) supervised by the NIPA(National IT Industry Promotion Agency).

References

[1] N. Fallenbeck, H.J Picht, M. Smith, B. Freisleben, Xen and the art of cluster scheduling, VTDC 2006, article no.4299349 (2006).
[2] A. McAfee, E. Brynjolfsson, Big data: the management revolution, Harvard Business Review, **90**, 60-66 (2012).
[3] Z. Tian, J. Xue, W. Hu, T. Xu, N. Zheng, High performance cluster-based transcoder, In: Proceedings of ICCASM 2010, **2**, 248-252 (2010).
[4] M. Kim, S.H Han, J.J Jung, H, O. Choi, A robust cloud-based service architecture for multimedia streaming using Hadoop, **274**, 365-370 (2014).
[5] H. Nansook, L. Dongsun, S. Dongmahn, J. Inbum, K. Yoon, Load distribution method and admission control for streaming media QoS in distributed transcoding servers, In: Proceeding of ICCSA 2007, article no.4301122, 39-45 (2007)
[6] K.J Ma, R. Bartos, S. Bhatia, A survey of schemes for Internet-based video delivery, Journal of Network and Computer Applications, **34**, 1572-1586 (2011).
[7] C. Li, G. Peng, K. Gopalan, T.C Chiueh, Performance guarantees for cluster-based Internet services, 3rd IEEE/ACM International Symposium on Cluster Computing and the Grid, 276-283 (2003).
[8] Y. M Teo, R. Ayani, Comparison of load balancing strategies on cluster-based web servers, Simulation, **77**, 185-195 (2001).
[9] A. Piorkowski, A. Kempny, A. Hajduk, J. Strzelczyk, Load balancing for heterogeneous web servers, Communications in Computer and Information Science, **79**, 189-198 (2010).
[10] A. Mourad, H. Liu, Scalable web server architectures, In: Proceedings of IEEE Symposium on Computers and Communications, 12-16 (1997).
[11] K. Shvachoko, H. Kuang, S. Radia, R. Chansier, The Hadoop file system, In: Proceedings of MSST 2010, article no.5496972 (2010).
[12] A. Karun, K. Chitharanjan, A review on hadoop-HDFS infrastructure extensions, In: Proceedings of ICT 2013, 132-137 (2013).
[13] J. Dean, S. Ghemawat, MapReduce: Simplified data processing on large clusters, Communication of the ACM, **51**, 107-113 (2008).
[14] D. Werth, A. Emrich, A. Chapko, An ecosystem for user-generated mobile services, **3**, 43-48 (2012).
[15] G. Barlas, Cluster-based optimized parallel video transcoding, Parallel Computing, **48**, 226-44 (2012).
[16] S.H Kim, K. Kim, W. Ro, Offloading of media transcoding for high-quality multimedia services, IEEE Transactions on Con. Elec., **58**, 691-699 (2012).
[17] M.A Weifeng, M. Keji, Research on java imaging technology and its programming framework, Lecture Notes in Electrical Engineering, **72**, 61-68 (2010).
[18] J. Guo, F. Chen, L. Bhuyan, R. Kumar, A cluster-based active router architecture supporting video/audio stream transcoding service, In: Proceedings of Parallel and Distributed Processing Symposium, 44-51 (2003).

[19] Cloudit 2.0, http://www.cloudit.co.kr
[20] D. Seo, J. Lee, Y. Kim, C. Choi, H. Choi, I. Jung, Load distribution strategies in cluster-based transcoding servers for mobile clients, Lecture Notes in Computer Science, **3983**, 1156-1165 (2006).
[21] Q. Shao, T. Yang, W. Hou, The design of high available single sign-on server of Nginx-based, Applied Mechanics Materials, **241**, 2411-2416 (2012).
[22] S. Ghemawat, H. Gobioff, S. T Leung, The google file system, Operating Systems Review, **37**, 29-43 (2003).
[23] M. Fan, H. Fan, H. Feng, S. Li, The development of OGC WCS server to support MODIS data format, In: Proceedings of SPIE, **7498**, article no.74980M (2009).
[24] B. Bockelman, Using Hadoop as a grid storage element, Journal of Physics, **180**, article no.012047 (2009).
[25] P. Barham, B. Dragovic, K. Fraser, S. Hand, T. Harris, A. Ho, R. Neugebauer, I. Pratt, A. Warfield, Xen and the art of virtualization, Operating Systems Review, **37**, 164-177 (2003).

Myoungjin Kim received an M.S. degree from Konkuk University, Seoul, Korea, in 2009. Currently, he is a Ph.D. student in the department of Internet and Multimedia Engineering at the same university, and he is also an assistant researcher at the Social Media Cloud Computing Research Center. His research interests include distributed computing, distributed real-time programming, MapReduce, media transcoding and cloud computing.

Seungho Han is an M.S. course student at the department of Internet and Multimedia Engineering, Konkuk University. He is also an assistant researcher at the Social Media Cloud Computing Research Center. His current research interests include universal plug and play, cloud computing, Hadoop, and ubiquitous cities.

Cui Yun received an M.S degree from the department of Internet and Multimedia Engineering at Konkuk University, Korea,. At present, he is a Ph.D. student and an assistant researcher at the Social Media Cloud Computing Research Center. His current research interests include cloud computing, social network services, home network services, and distributed computing.

Hanku Lee is the director of the Social Media Cloud Computing Research Center and a professor at the division of Internet and Multimedia Engineering, Konkuk University, Seoul, Korea. He received a Ph.D. degree in Computer Science from Florida State University, USA. His recent research interests include cloud computing, distributed real-time systems, distributed computing, and compilers.