Cluster Comput (2014) 17:605–628
DOI 10.1007/s10586-014-0381-0

CloudDMSS: robust Hadoop-based multimedia streaming service architecture for a cloud computing environment

Myoungjin Kim · Seungho Han · Yun Cui ·
Hanku Lee · Hogyeon Cho · Sungdae Hwang

Received: 25 August 2013 / Revised: 6 February 2014 / Accepted: 7 May 2014 / Published online: 25 May 2014

Abstract The delivery of scalable, rich multimedia applications and services on the Internet requires sophisticated technologies for transcoding, distributing, and streaming content. Cloud computing provides an infrastructure for such technologies, but specific challenges still remain in the areas of task management, load balancing, and fault tolerance. To address these issues, we propose a cloud-based distributed multimedia streaming service (CloudDMSS), which is designed to run on all major cloud computing services. CloudDMSS is highly adapted to the structure and policies of Hadoop, thus it has additional capacities for transcoding, task distribution, load balancing, and content replication and distribution. To satisfy the design requirements of our service architecture, we propose four important algorithms: content replication, system recovery for Hadoop distributed multimedia streaming, management for cloud multimedia management, and streaming resource-based connection (SRC) for streaming job distribution. To evaluate the proposed system, we conducted several different performance tests on a local testbed: transcoding, streaming job distribution using SRC, streaming service deployment and robustness to data node and task failures. In addition, we performed three different tests in an actual cloud computing environment, Cloudit 2.0: transcoding, streaming job distribution using SRC, and streaming service deployment.

M. Kim · S. Han · Y. Cui · H. Lee (✉)
Department of Internet and Multimedia Engineering,
Konkuk University, Seoul, Korea
e-mail: hlee@konkuk.ac.kr

H. Cho · S. Hwang
Technical Support Team, Innogrid Co., Ltd., Seoul, Korea

Keywords Streaming service · Mobile media service · Cloud computing · Media transcoding · Distributed processing

1 Introduction

With the recent proliferation of rich social media across a variety of personal devices, considerable attention has shifted to the challenge of adaptively distributing and streaming multimedia content over the Internet. Among the emergent technologies, cloud-based media streaming, transcoding, and distributed storage have been the most noteworthy and influential. Before the advent of cloud computing technology, multimedia services employed traditional distributed and cluster-based computing approaches for media transcoding and streaming processing. However, supporting the quality of service (QoS) is difficult for these multimedia services using traditional approaches because of the explosive growth in mobile services [1] and multimedia traffic caused by the high resolution and capacity of recent multimedia formats.

In the areas of traditional multimedia transcoding and streaming, much research has focused on distributed and cluster-based video media approaches [2–6] for reducing the transcoding processing time to facilitate the delivery and transmission of multimedia to the end user while considering limited network traffic, as well as streaming job distribution. However, these approaches have several problems and limitations. First, these transcoding approaches only focus on obtaining computing resources for multimedia transcoding processes simply by increasing the number of cluster machines in a parallel and distributed computing environment. Second, during the recovery phase of a QoS-guaranteed streaming system, these approaches do not include load balancing, fault tolerance, or a data replication method that ensures data protection and expedites recovery

[7]. Thus, the absence of an automated recovery policy to prevent the loss of multimedia content and system faults in traditional streaming systems mean that these approaches cannot guarantee the reliability of a system when providing rich media services. Finally, most of the systems that use these approaches do not include or consider a media transcoding function for streaming services, thus vendors and developers find it difficult to simultaneously construct and develop transcoding and streaming modules for distributed and cluster environments.

To overcome these limitations and problems, many researchers and developers have adopted cloud computing technologies [8–12] for multimedia service owing to advantages such as a flexible dynamic IT infrastructure, QoS-guaranteed computing environments, cloud programming models [13], and configurable software services [14]. Cloud-based media streaming services that are distributed over the Internet have been released: iCloud (Apple), Cloud Player and Cloud Drive (Amazon), Azure Media Services (Microsoft), and Netflix. Cloud-based transcoding services such as Amazon Elastic transcoder, Zencoder, and Ankoder have been released [15].

Cloud-based technologies have emerged owing to the features of recent multimedia services: the heterogeneity of media, QoS, networks, and devices [16]. To support such features, media streaming, transcoding, and distribution must depend on massive—and massively scalable—computational resources: i.e., CPUs, memory, network bandwidth, and storage. Although cloud computing can provide these resources, in doing so, it also introduces a heavy burden on existing Internet infrastructure and cloud resources as well as a host of new challenges (e.g., cluster rebalancing, namespace management, data distribution/replication [17], auto-recovery, and fault tolerance), which are intensified by the massive swings in traffic associated with rich media streaming. Developers and service vendors have both found that these challenges are difficult to resolve, and they continue to hinder current media delivery systems.

To address these challenges, we propose a cloud-based distributed multimedia streaming service (CloudDMSS) system based on Hadoop [18,19], which is designed to run on the current cloud computing infrastructure. The capacities of CloudDMSS include the following:

(1) Transcoding of large volumes of media into the MPEG-4 video format for delivery to a variety of devices including PCs, smart pads, and phones;
(2) Reduction in the transcoding time by incorporating the Hadoop distributed file system (HDFS) for the storage of multimedia data and MapReduce for distributed parallel processing;
(3) Reduced content delay and traffic bottlenecks using a streaming job distribution algorithm;
(4) Improvement in the overall performance using dual-Hadoop clustering for each physical cluster;
(5) Efficient content distribution and improved scalability by adhering to Hadoop policies;
(6) Conducting the workflow of sequential tasks automatically during streaming service deployment.

In this study, we describe the design of the CloudDMSS architecture with an HDFS-based [20,21] distribution and storage function for source media files, a batch processing function to transcode a large number of media files, an automatic migration function for transcoding contents to content servers based on HDFS, and a streaming job distribution strategy. We designed and defined a set of systematic cloud-based multimedia streaming processing workflows. We focused on using CloudDMSS to conduct a workflow based on a dual-Hadoop cluster for each physical cluster using a variety of open sources and we developed a Web-based dashboard to support user interfaces by monitoring cloud resources. Experimental evaluations of CloudDMSS were conducted to verify its performance.

The remainder of this paper is organized as follows. Section 2 discusses relevant research on cloud-based streaming services. Section 3 describes the core architecture of CloudDMSS and the workflow of the streaming service deployment process, as well as presenting four robust algorithms that satisfy the design requirements of our service architecture. Section 4 explains the prototype of the proposed system and its configuration. In Sect. 5, we discuss the results of several experiments conducted using a 28-node cluster over a local testbed. In Sect. 6, we discuss performance evaluations conducted in an actual cloud environment. Section 7 presents our concluding remarks and plans for future work.

2 Related works

In recent years, many researchers have applied cloud computing technologies to rich media services in response to the explosion in demand. We consider the three aspects that are most relevant to our CloudDMSS system: Hadoop, media transcoding, and multimedia cloud computing.

2.1 Hadoop

Hadoop was inspired by Google's MapReduce and Google File System [22], and it 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, which is a distributed, scalable, and portable file system written in Java, like the Google file system (GFS); and (2) MapReduce, 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 suitable for various real-world tasks [5]. This framework also handles automatic scheduling, communication, and synchronization during the processing of huge datasets and it has a fault tolerance capacity. The MapReduce programming model is executed in two main steps called *mapping* and *reducing*. Mapping and reducing are defined by *mapper* and *reducer* functions. Each phase requires a list of key and value pairs as the input and output. In the *mapping* step, MapReduce receives the input dataset 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.

2.2 Media transcoding

The term media transcoding has been defined in many previous studies, such as [23,24]. In [14], multimedia information must be adapted to bring multimedia contents and service to numerous heterogeneous client devices while retaining the capacity for mobile usage, which is referred to as media transcoding technology.

Figure 1 shows the architecture of a legacy transcoding system. First, the client requests a transcoding function from a transcoding server. The transcoding sever reads the original media data from the media server and then proceeds to transcode the data depending on user requested resolution, bit-rate, and frame rate. The transcoding server then

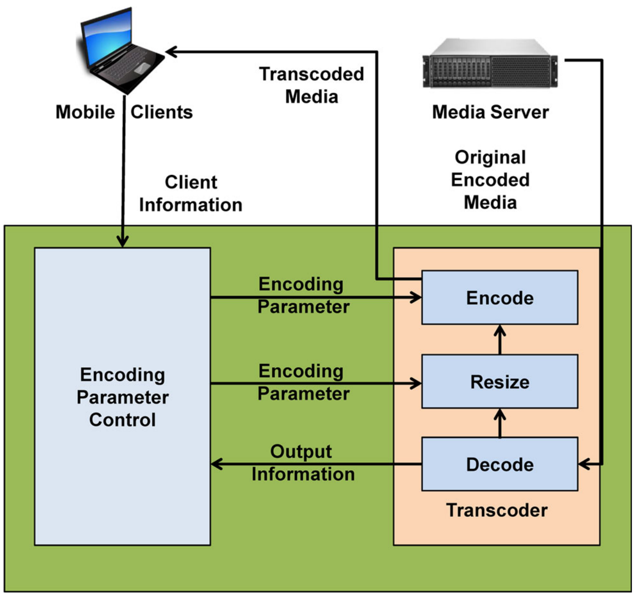


Fig. 1 Architecture of a legacy transcoding system [25]

sends the transcoded media data to the client [25]. However, this media transcoding processing imposes a heavy burden on the existing internet infrastructure and computing resources because more recent media files, such as video and image files, have changed to high capacity/high definition.

Therefore, many researchers have applied distributed and parallel computing to media transcoding methods. For example, Guo et al. [4] proposed a cluster-based multimedia web server, where they designed and implemented a media cluster that dynamically generates video units in order to satisfy the bit rate requested by many clients, as well as proposing seven load balance scheduling schemes for the MPEG transcoding service. Sambe et al. [26] designed and implemented a distributed video transcoding system with the capacity to transcode an MPEG-2 video file into diverse video formats with different rates. The main reason for transcoding a video file is that the transcoder chunks the MPEG-2 video file into small segments along the time axis, before transcoding them in a parallel and distributed manner.

Tian et al. [27] described a cluster-based transcoder that transcodes MPEG-2 format video files into MPEG-4 and H.264 format video files with a faster transcoding speed. This system comprises a master node and a number of worker nodes. The master node has six threads, a splitter, merger, sender, receiver, scheduler, and an audio transcoder.

2.3 Multimedia cloud computing

Multimedia traffic has increased dramatically over the Internet since the release of various personal devices and changes in the content of video and image files to high capacity and high definition. To support a high QoS for heterogeneous devices such as smartphones, personal computers, smart televisions, and smart pads, many researchers and developers have tried to apply cloud computing technologies to multimedia services. In this section, we introduce the basic concept of multimedia cloud computing and we consider some recent studies.

Zhu et al. [16] introduced the first principal concept of the multimedia cloud computing model, where they addressed multimedia cloud computing from multimedia-aware cloud (media cloud) and cloud-aware multimedia (cloud media) perspectives. Figure 2 shows the relationship between the media cloud and cloud media services. A multimedia-aware cloud perspective focuses on how the cloud can provide QoS for multimedia applications and services. A cloud-aware multimedia perspective focuses on how multimedia can perform content storage, processing, adaptation, rendering, and other functions in the cloud to best utilize cloud-computing resources, thereby delivering a high quality of experience for multimedia services.

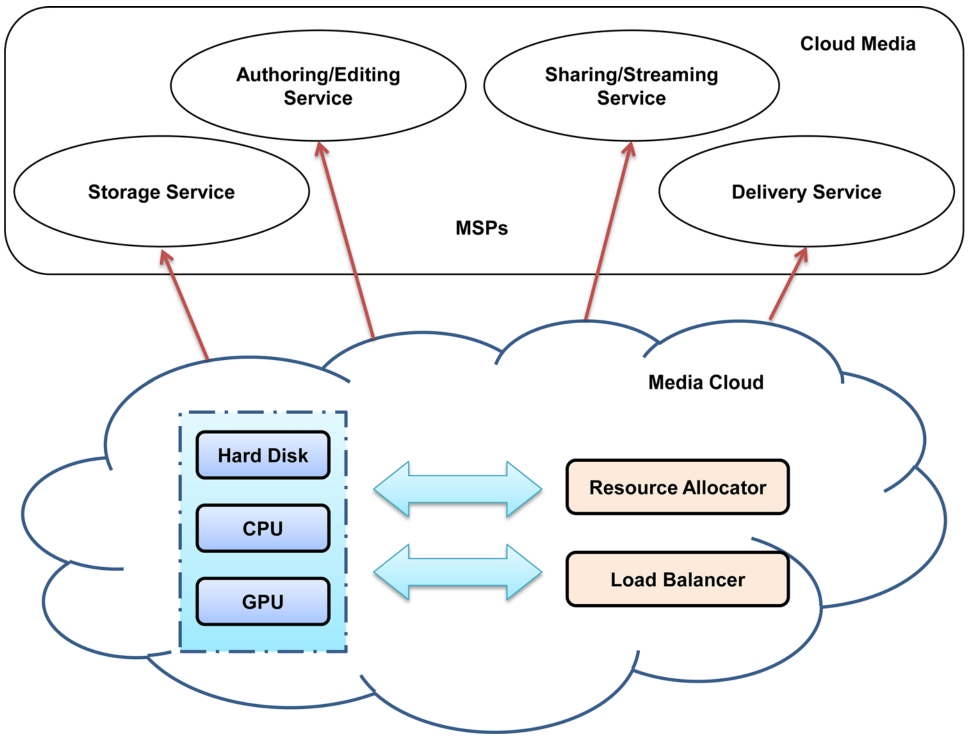


Fig. 2 Relationship between media cloud and cloud media services [16]

To the best of our knowledge, there have been few previous reports of multimedia cloud computing. Hui et al. [28] proposed *MediaCloud*, which is a layered architecture that defines a new paradigm for dealing with multimedia applications and services. The architecture comprises three layers—a *Media Service Layer*, a *Media Overlay Layer*, and a *Resource Management Layer*—and it addresses key challenges such as heterogeneity, scalability, and QoS provisioning. However, this architecture operates mainly at the conceptual level and it leaves most of the challenges of real-world implementation for future work [28]. By contrast, Luo et al. addressed the implementation challenge of QoS delivery over a virtualized infrastructure by presenting a practical architecture and mechanism for a private media cloud [29]. They described their system in terms of four major components: monitoring, load balancing, traffic management, and security. In the context of cloud-based streaming, Lee et al. [15] proposed a configuration scheme for connectivity-aware P2P networks based on algorithms for connectivity-aware mobile P2P network configuration and connectivity-aware P2P network reconfiguration. Chang et al. [30] described a cloud-based media streaming architecture that dynamically adjusts streaming services in response to mobile device resources, multimedia codec features, and the network environment. They also presented a design for a stream dispatcher component, including the real-time adaptation of codecs in response to client device profiling and a dynamic adjustment of multimedia streaming algorithm. Huang et al. [31] presented *CloudStream*, which is a cloud-based video proxy that is capable of delivering high-quality video streams by transcoding the original video in real time to a scalable codec, thereby allowing the adaptation of the stream to various network dynamics. They also proposed a multi-level transcoding parallelization framework with two mapping options: Hallsh-based mapping and lateness-first mapping.

3 Architecture and workflow of CloudDMSS

In this section, we describe the design strategy of the CloudDMSS system, including the overall workflow from uploading original multimedia content to supporting a streaming service for end users, without content delay and traffic bottlenecks. Figure 3 shows the fundamental concept of our service model.

Personal media data such as movies, music videos, and animations are distributed and stored on 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) that 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 guaranteed QoS by controlling the streaming servers that run on the streaming Hadoop clus-

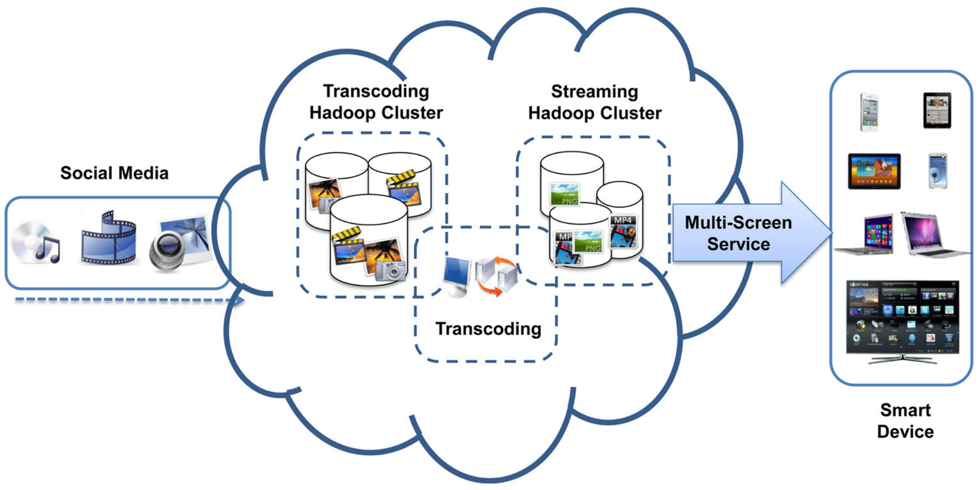


Fig. 3 Fundamental concept of the cloud-based distributed multimedia streaming service model (CloudDMSS)

ter. To reduce content delays and traffic bottlenecks, our service model utilizes a streaming job distribution algorithm, which balances and distributes the load of the streaming servers.

3.1 Overall system architecture

Our proposed CloudDMSS is designed to run on a Hadoop cluster for heterogeneous devices in a distributed manner. The overall system architecture is shown in Fig. 4. CloudDMSS has three main components: Hadoop-based distributed multimedia transcoding (HadoopDMT), Hadoop-based distributed multimedia streaming (HadoopDMS), and cloud multimedia management (CMM). The characteristics of our system are as follows. (1) Our system transcodes large amounts of media content into the MPEG-4 video format for delivery to a variety of devices, including PCs, smart pads, and phones. (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 delay and traffic bottlenecks using a streaming job distribution algorithm. (4) Dual-Hadoop clustering per physical cluster is used to improve the overall performance and 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.

3.2 CloudDMSS system architectural components

3.2.1 HadoopDMT

The main role of HadoopDMT is to transcode a various multimedia data stored on a transcoding Hadoop cluster into MPEG4, which is the standard format for media streaming to a variety of devices. HadoopDMT improves the quality and speed by using the HDFS [32] to store video data from many sources, MapReduce [32] for distributed parallel processing of these data, and Xuggler [33] to transcode the data. The capabilities of HadoopDMT are as follows. (1) HadoopDMT comprises a codec transcoding function with a configurable display size, codec method, and container format. (2) It focuses mainly on the batch processing of large video files collected over a fixed period of time, rather than processing small video files collected in real time. (3) HDFS is used to avoid the high cost of communicating video files during data transfer for distributed processing. HDFS is also used because the large chunk size (64 MB) policy is suitable for processing video files and the user-level distributed system. (4) HadoopDMT incorporates the load balancing, fault tolerance, and merging and splitting policies provided by MapReduce for distributed processing.

HadoopDMT [34] is divided into four main domains: video data collection domain (VDCD), HDFS-based splitting and merging domain (HbSMD), MapReduce-based transcoding domain (MbTD), and cloud-based infrastructure service domain (CbISD). Figure 5 shows the detailed structure of HadoopDMT.

First, the main contribution of VDCD is to collect the different types of original encoded video files created by media creators such as SNS providers, media sharing services, and personal users, as well as the storage of these files on the HDFS of the transcoding Hadoop cluster. It also collects the transcoded video datasets converted to a target format file by a transcoding processing step based on MapReduce in MbTD, and stores them on the HDFS. The period required to collect the original encoded video datasets can be set by administrators and users, depending on the dataset size and acquisition time. Second, the main role of HbSMD, which runs on HDFS, is to split collection of original video datasets

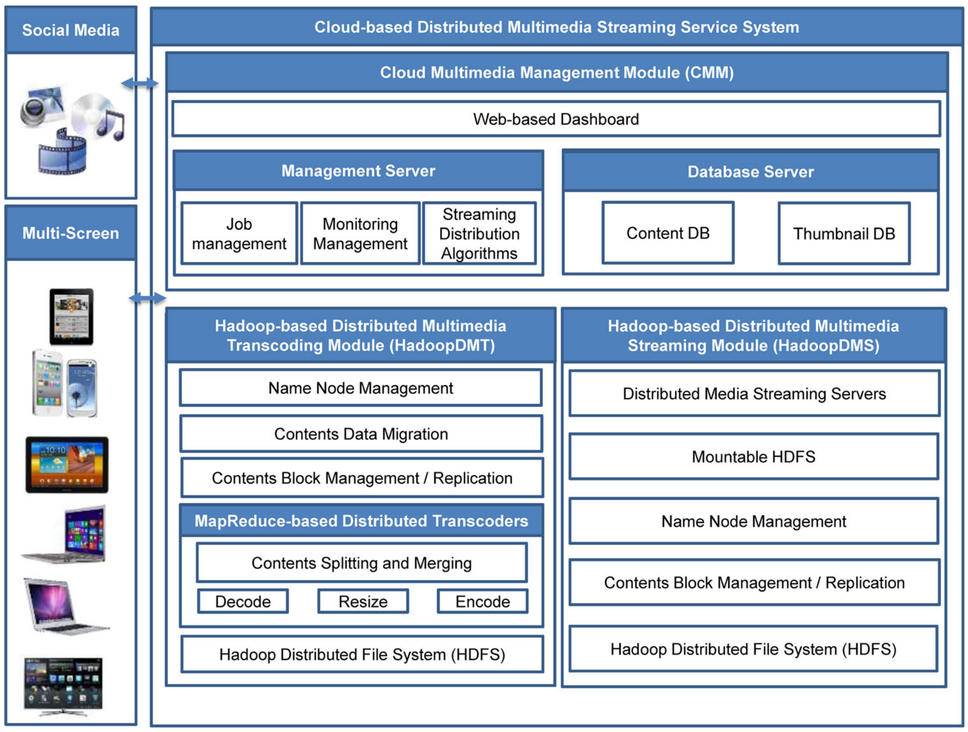

Social Media
Multi-Screen
Cloud-based Distributed Multimedia Streaming Service System
Cloud Multimedia Management Module (CMM)
Web-based Dashboard
Management Server
Job management
Monitoring Management
Streaming Distribution Algorithms
Database Server
Content DB
Thumbnail DB
Hadoop-based Distributed Multimedia Transcoding Module (HadoopDMT)
Name Node Management
Contents Data Migration
Contents Block Management / Replication
MapReduce-based Distributed Transcoders
Contents Splitting and Merging
Decode
Resize
Encode
Hadoop Distributed File System (HDFS)
Hadoop-based Distributed Multimedia Streaming Module (HadoopDMS)
Distributed Media Streaming Servers
Mountable HDFS
Name Node Management
Contents Block Management / Replication
Hadoop Distributed File System (HDFS)

Fig. 4 Overall architecture of cloud-based distributed multimedia streaming service system

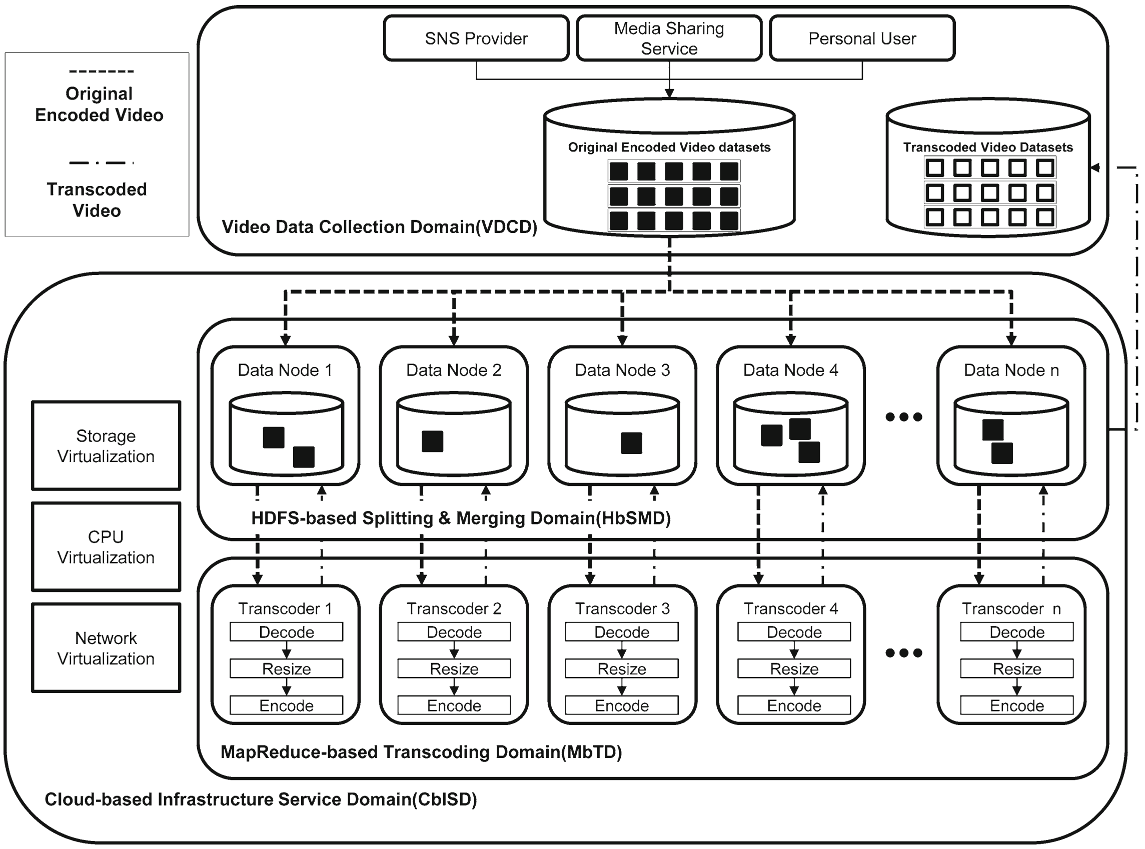


Fig. 5 Detailed structure of HadoopDMT

into blocks with a configured size and to automatically distribute all of the blocks over the cluster. In HbSMD, the default block size is set to 64 MB, but it can be changed to various other values by administrators and users, such as 16, 32, 128, 256 MB, etc. When a block is distributed, it is replicated at three data nodes according to the Hadoop distribution policy, thereby complying with the overall distributed processing procedure and enabling recovery from a system failure caused by data loss. The other role of HbSMD is to merge the blocks transcoded by transcoders in MbTD into target video files and to transmit the video files to VDCD. The number of block replicas is set to 1, 2, 4, 5, etc. Third, MbTD performs several tasks that transcode the distributed blocks in each data node using a MapReduce-based distributed transcoder module with Xuggler. A single MapReduce-based distributed transcoding task is managed as one MapReduce job. It is also scheduled by JobTracker in a name node and TaskTracker in a data node. First, JobTracker schedules and monitors the overall job registered in HadoopDMT. If a transcoding job is submitted to HadoopDMT, one JobTracker task is run. JobTracker calculates the number of distributed tasks (Map tasks) that are split into blocks. When a 10-GB dataset, i.e., 50×200 MB files, is transcoded with the default block size option of 64 MB, 200 Map tasks are generated. After the number of tasks has been determined, JobTracker assigns Map tasks to the TaskTrackers, and each TaskTracker performs the assigned tasks. As blocks are transcoded, each TaskTracker periodically sends heart bit method calls to the JobTracker to maintain the robustness against task failures. If a Map task fails, JobTracker detects the situation via the heart bit, and recovers the task failure by rescheduling and reassigning the remaining tasks, including failed tasks. Data node 1 and transcoder 1 are located in the same physical machine. First, the transcoders implement the decoding step. Next, the resizing step is implemented if the users and administrators require a change in the resolution of a video file. If such a change is not required, the transcoders skip this step. The transcoders encode the decoded blocks into a target file based on the requirements of the user. Finally, CbISD offers infrastructure services in a cloud computing environment via server, storage, CPU, and network virtualization techniques. Because of the massive storage space and enormous computing resource requirements of such systems, small service vendors are unable to afford the cost of building them. When users require logical computing resources to build and implement this system, CbISD automatically deploys a virtualized cluster environment. CbISD allows users to select a specific configuration of memory, CPU, storage, and the number of clusters. In addition, it provides the easy installation and configuration environment of HDFS and MapReduce, which require little effort from the user. In this study, we present the idea and concept of CbISD, but its implementation is not considered.

The other role of HadoopDMT is to automatically migrate the transcoded media content stored on the HDFS of the transcoding Hadoop cluster to the HDFS of a streaming Hadoop cluster in HadoopDMS. In addition, HadoopDMT extracts the thumbnail images of transcoded content and transmits the extracted thumbnail images to HadoopDMS. After the content migration and thumbnail extraction tasks have been completed, HadoopDMT deletes the original media content and transcoded content stored on the HDFS of the transcoding Hadoop cluster in order to release storage space for the next transcoding task.

3.2.2 HadoopDMS

HadoopDMS, which runs on the streaming Hadoop cluster, comprises HDFS-based data nodes that act as content storage servers, a name node that acts as a namespace server, and streaming severs. Our system uses a dual Hadoop cluster on one physical cluster, including the transcoding Hadoop cluster, in HadoopDMT and the streaming Hadoop cluster in HadoopDMS, which balances the task loads such as transcoding and streaming. The quality of the streaming service is not guaranteed if both a transcoding task, which requires intensive computing resources, and a streaming task are conducted at the same time.

The first role of HadoopDMS is to store the transcoded contents migrated from HadoopDMT on data nodes of the streaming Hadoop cluster in a distributed manner and to provide the capacity for content replication and recovery if data node failures and loss of content occur. 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 loss of content. A streaming service system that uses the traditional approach does not include content replication management and automated recovery policies, so the system reliability and seamless streaming service cannot be guaranteed. To overcome these problems, HadoopDMS includes intelligent content replication and recovery policies.

Algorithm 1 is the content replication policy. First, HadoopDMS splits the content into blocks with the configured size (default = 64 MB) and it then creates two replicas of each original block. In HadoopDMS, the default block replication is set to 3 (one original block and two replicas of original block). However, this can be changed by administrators and users because the block replication factor affects the system performance, depending on the system specifications and physical system configuration. In the performance evaluation section (Sect. 5), we describe the optimal Hadoop options, including the block replication factors, and block size options. In particular, we experimentally verify that the performance is better when the block replication factor value is set to ≥ 3 (3, 4, 5). The worst performance is obtained when the value is set to 1, because new blocks with problems should be copied and transferred to a new data node on HDFS if task and disk failures and data loss occur. To maintain the robustness of the system to task failures, CloudDMSS uses two replicas as the default option. CloudDMSS is designed to be deployed on low-cost hardware and commodity hardware, thus our system focuses on high fault-tolerance rather than incurring the cost of building numerous hard disks for replicas. After the replicas have been created, they are distributed on the data nodes while complying with the Hadoop storage policy. The name node of HadoopDMS then collects metadata, including the status information for data nodes, location information of stored content, and the content file names required to maintain and control the streaming Hadoop cluster and multimedia content.

Algorithm 1 Algorithm for content replication

```
Data : mf: migrated content file in the HadoopDMS

 1: while system available
 2: input (mf)
 3: split mf into blocks of 64 MB;
 4: while create block of mf
 5:   if number of blocks ==3 then
 6:     break;
 7:   else if the number of blocks > 3 then
 8:     while check the number of blocks
 9:       if number of blocks ==3 then
10:         break;
11:       else
12:         delete one block;
13:       end if
14:     end while
15:   else
16:     while check the number of blocks
17:       if number of blocks ==3 then
18:         break;
19:       else
20:         copy and make one block;
21:       end if
22:     end while
23:   end if
24: end while
25:end input
26:end while
```

When data loss and data node failures occur, the name node in HadoopDMS detects the situation based on the status information, which is collected periodically by the node, and it performs an automatic recovery scheme using Algorithm 2. Algorithm 2 facilitates system recovery in the event of data loss and data node failures. First, the name node (nn) sends a check packet (cp) to all of the data nodes to detect data loss, before comparing the number and name of the blocks in the block information (bi) of the name node with those of the check packet (cp) generated by each data node. If the name node and data node have different bi, a new content replication task is performed by Algorithm 1 in each node. After checking the data loss in each data node, nn sends a heart bit (hb) to all of the data nodes to detect any failures. If nn receives the hb generated by each data node within 5 min, there is no system failure on the node. Otherwise, the policy for recovery from system failure is executed. The procedure of the recovery policy based on nn is as follows. (1) nn reconfigures the cluster with data nodes, excluding the failed node, and conducts a new content replication task. (2) The new bi is updated in nn. (3) nn automatically detects the cause of the system failure and recovers the failed node automatically via remote inspection. (4) After the completion of recovery, nn restarts the overall system and reconfigures the cluster to include the recovered data node. (5) nn repeats (1) and (2).

Algorithm 2 Algorithm for system recovery

```
Data: nn: name node
Data: $node_n$: data $node_1$, data $node_2$ ....data $node_n$
Data: hb: heart bit for checking the active status of $node_n$
Data: cp: check packet to check blocks of $node_n$
Data: bi: block information including the number and
      name of blocks requested by check bit

 1:while system available
 2: for (each node $_n$)
 3:   nn sends cp to node $_{n;}$
 4:   if receive cp then
 5:     if block information of nn != bi of $node_n$ then
 6:       do content replication algorithm;
 7:   else
 9:       do checking hb;
10:     end if
11:   end if
12: end for
13: for(each node $_n$)
14:   nn sends hb to node $_n$;
15:     if the time to receive hb from $node_n$ $\geq$ 5 min then
16:       exclude node $_n$ in cluster;
17:       do replication algorithm;
18:       update bi in nn;
19:       if receive hb from all nodes by checking hb again
20:         include node $_n$ in cluster;
21:         do replication algorithm;
22:         update bi in nn;
23:       end if
24 :    end if
25: end for
26:end while
```

The second role of HadoopDMS is to provide users with the thumbnail images extracted from HadoopDMT so that they can easily search and select media content on the Web interface. These thumbnail images are stored on a specific data node and registered on a thumbnail DB of the DB server in CMM to facilitate their management.

3.2.3 CMM

As a core part of CloudDMSS, CMM controls and manages a various tasks created during each phase of the streaming service deployment process, while it also monitors the overall system usage and status information related to streaming processes, such as uploading, transcoding, and content migration for users and administrators.

Our 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 management server is scheduled using Algorithm 3.

Algorithm 3 Algorithm for the management server

```
Data: DMT: HadoopDMT component
Data: DMS: HadoopDMS component
Data: request_ID: id for requesting a transcoding task
Data: tc : transcoded content
Data: et: extracted thumbnail image
Data: ci: information with the name and location of tc

 1: while system available
 2:   if request for transcoding then
 3:     transcoding_ready_task(request_ID){
 4:       setup ssh shell connection;
 5:       make a new folder for transcoded contents;
 6:     }
 7:     while conduct transcoding tasks
 8:       if error detected (request_ID) then
 9:           stop_transcoding_task(request_ID);
10:           delete_transcoded contents_task(request_ID);
11:           conduct_transcoding_task(request_ID)
12:           break;
13:       end if
14:       if task completed then
15:           thumbnail_extraction_task(tc);
16:           migration_task(tc→DMS)
17:           registration_task(et, ci→database server)
18:       end if
19:     end while
20:   end if
21:   if request for multimedia streaming then
22:     do streaming resource-based connection algorithm;
23:   end if
24: end while
```

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. A streaming job distribution scheme is carried out by round robin (RR) [35], least connection (LC) [35], and streaming resource-based connection (SRC) algorithms. 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 utilization rate and network transmission rate, 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 algorithms consider the CPU, RAM, and streaming transmission rate usage of servers generated from each streaming server. The Linux command *mpstat* is used to generate statistics about the CPU usage servers. The *free* command is used to inquire about the RAM usage and */proc/net/dev* is used to inquire about the streaming transmission rate. Algorithm 4 uses SRC in CloudDMSS.

Algorithm 4 Algorithm for SRC

```
Data: $ss_n$_ID: Id for streaming server $_1$, server $_2$ ….sever$_n$
          streaming server $_n$
Data: $ssu_n$: the system usage rate of $ss_n$
Data: $scu_n$: the CPU usage rate of $ss_n$
Data: $sru_n$: the RAM usage rate of $ss_n$
Data: $str_n$: streaming transmission rate of $ss_n$
Data: request_ID: id for requesting streaming service

 1: while system available
 2: if request for streaming service then
 3:   for(each $ss_n$_ID)
 4:       calculate_system usage($ss_n$_ID){
 5:       $ssu_n = scu_n + sru_n$;
 6:       $str_n$ = current $str_n$ – $str_n$ before 1sec
 7:       }
 8:   end for
 9:   select streaming server(requst_ID){
10:     if the number of the smallest ssu ==1 then
11:         set server(request_ID, the smallest ssu);
12:         start_streaming_service (request_ID, content);
13:     else
14:         set server(request_ID, the smallest str);
15:         start_streaming_service(request_ID, content);
16:     end if
17:     }
18:   end if
19:end while
```

The database server module manages and registers the thumbnail images in the thumbnail DB and content information, including the file name and location, in the content DB. Users can easily access streaming services by utilizing the DB information on our Web interface.

3.3 Workflow of CloudDMSS

In this section, we define the workflow of the sequential tasks in CloudDMSS during the streaming service deployment process in the overall system. Figure 6 shows the overall workflow of CloudDMSS. The CloudDMSS workflow is divided into two parts: content transcoding processing and distributed streaming processing.

3.3.1 Content transcoding processing workflow

We introduce the workflow required for content transcoding processing to stream content to heterogeneous devices. Figure 7 shows a diagram of the workflow of content transcoding processing. Users and administrators can access our user interface via CMM to upload their own original multimedia content. After an upload task is complete, they select the user-requested options (resolution, frame rate, bit rate, codec, and container) for transcoding and then request a content transcoding task from a MapReduce-based distributed transcoder of CMM. Immediately after the task is requested, CMM prepares a preprocessing procedure for the task. The preprocessing procedure comprises two steps. First, CMM creates an SSH shell in both HadoopDMT and CMM to allow remote command execution and secured content migration between both components. Second, the file names of the transcoded and original contents are the same, so CMM creates a new folder to store the transcoded contents to avoid any content loss due to file redundancy. After the preprocessing procedure is complete, a management server module in CMM performs the transcoding task by operating the MapReduce-based distributed transcoder in a distributed manner. Subsequently, the management server module commands a thumbnail extraction task from a thumbnail image extractor in HadoopDMT, and the extractor extracts thumbnail images of the transcoded content using FFmpeg supported by Xuggler [33] libraries. In the next step, the transcoded content and extracted thumbnail images are migrated to HadoopDMS via a content migrator in HadoopDMT. After migration, the original multimedia content, transcoded content, and thumbnail images are deleted by the management server, which then registers the thumbnail images to a thumbnail database in a CMM database server, as well as content information, including the file name and location, to a content database in the same server.

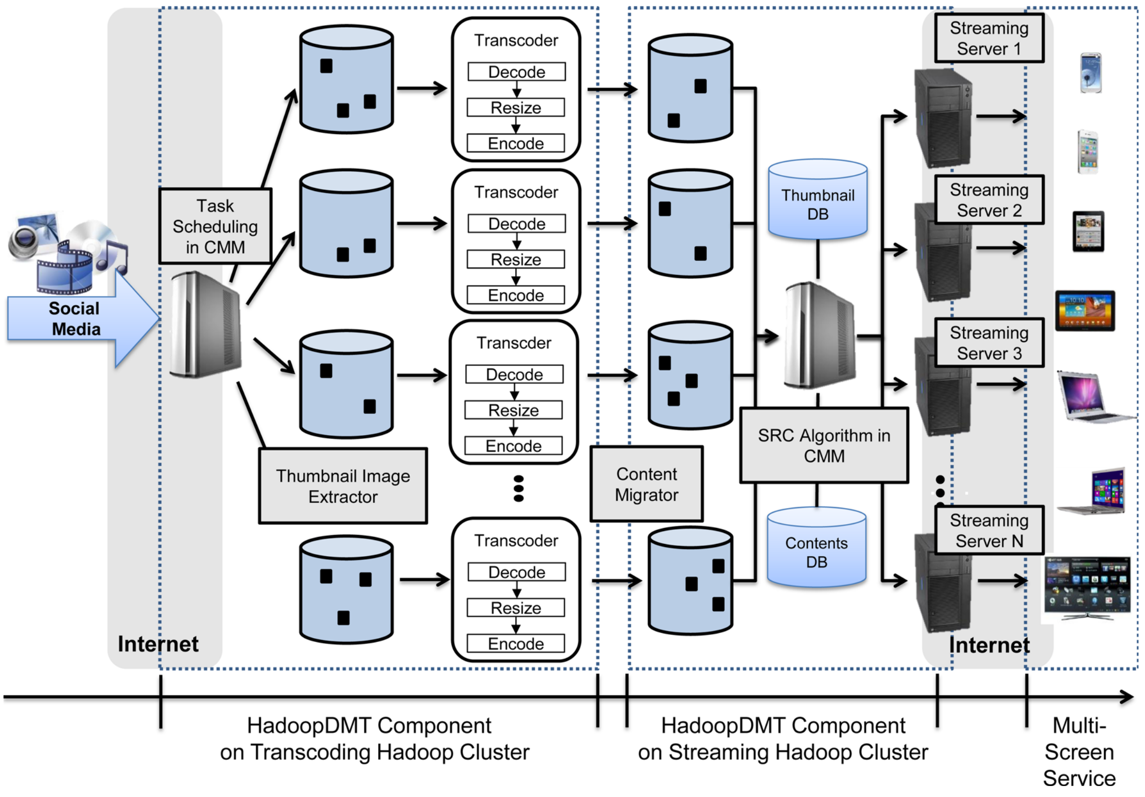


Fig. 6 Overall workflow of CloudDMSS

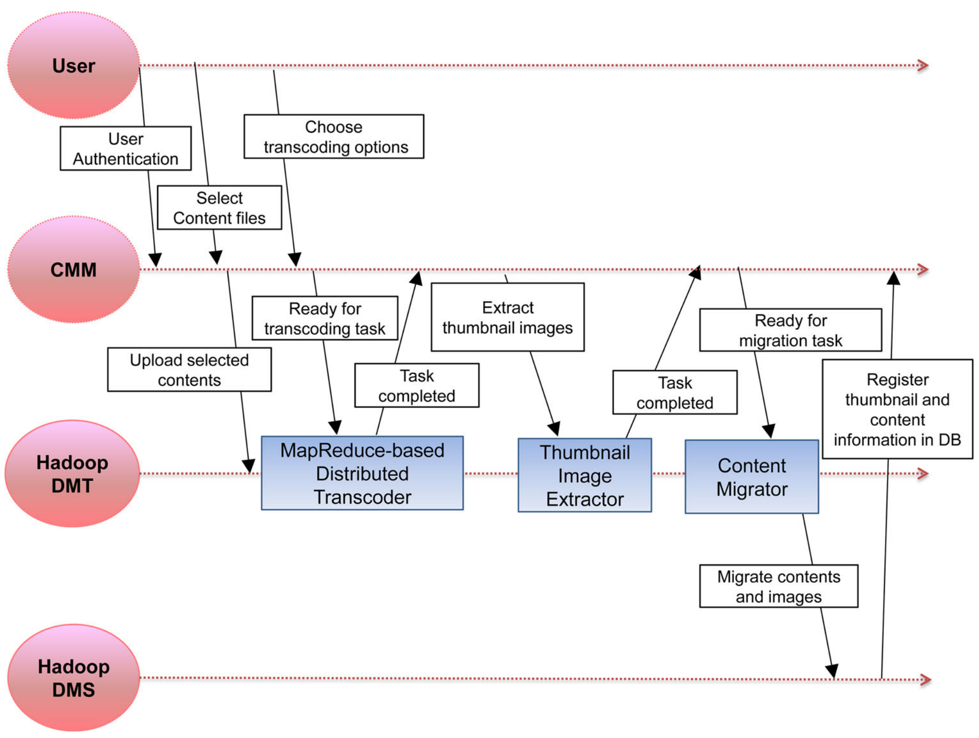

User
User Authentication
Select Content files
Choose transcoding options
CMM
Upload selected contents
Ready for transcoding task
Task completed
Extract thumbnail images
Task completed
Ready for migration task
Register thumbnail and content information in DB
Hadoop DMT
MapReduce-based Distributed Transcoder
Thumbnail Image Extractor
Content Migrator
Migrate contents and images
Hadoop DMS

Fig. 7 Diagram showing the workflow of content transcoding processing

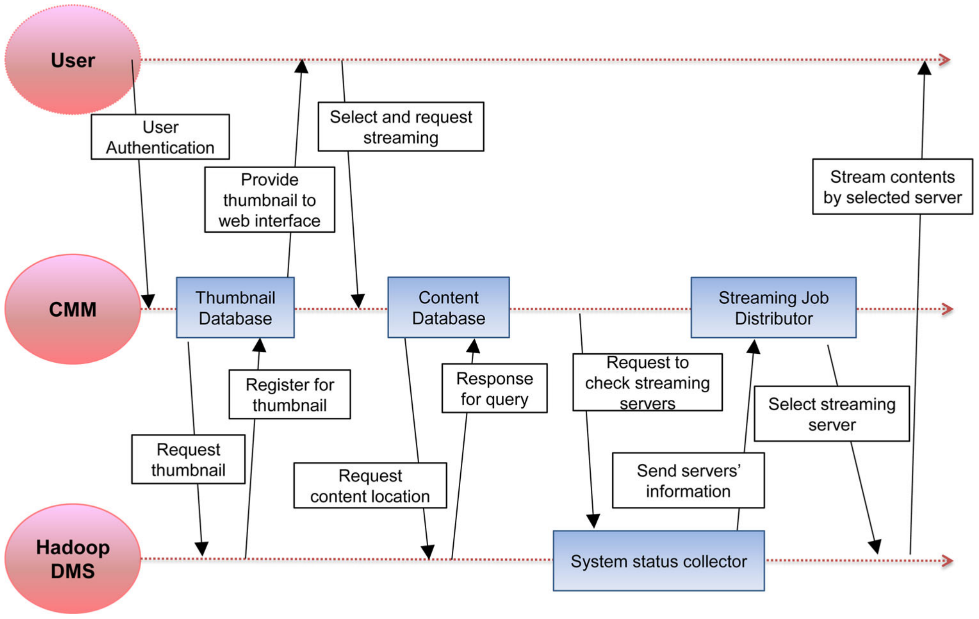


Fig. 8 Diagram showing the workflow of distributed streaming processing

3.3.2 Distributed streaming processing workflow

The distributed streaming processing workflow with a streaming job distribution scheme is the most important part of the overall processing flow. Figure 8 illustrates the distributed processing workflow. Thumbnail images are updated periodically in a Web interface and the metadata of the thumbnail images are registered each time a transcoding task is completed. When users select and request media content by clicking a thumbnail image, CMM obtains the location and file name of the selected media via a content database. CMM then requests the status of streaming servers via a system status collector and selects the optimal streaming server from those in HadoopDMT based on the SRC algorithm, as described in Sect. 3.2.3. The content streaming service begins after the optimal streaming server is selected.

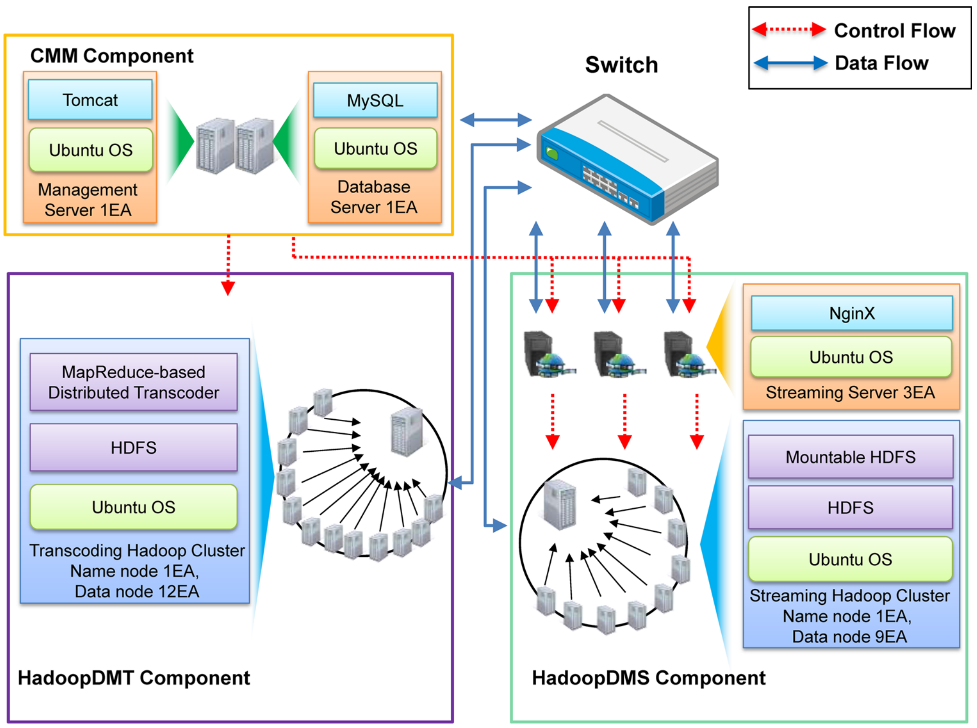


Fig. 9 Detailed cluster configuration of CloudDMSS

4 Implementation and prototype

We designed a robust CloudDMSS based on Hadoop clustering to provide a seamless streaming service with adequate QoS over the current Internet. We focused on developing a system that automatically conducts the entire workflow of streaming tasks, including uploading, transcoding, migration, thumbnail extraction, content registration, and streaming job distribution, by a single click of the content upload button on the Web interface provided. We implemented a prototype of the overall system architecture running on a real cloud cluster to validate the feasibility of our service architecture according to the design issues described above.

4.1 Cluster configuration and implementation

Figure 9 shows the hardware configuration of CloudDMSS. In our prototype implementation of CloudDMSS, we constructed our own cluster servers in a cloud computing environment, which comprised 28 nodes in total. Each node consisted of Linux OS (Ubuntu 10.04 LTS) running on two Intel Xeon quad-core 2.13 GHz processors with 4 GB registered ECC DDR memory and 1 TB SATA-2 disk storage. All of the nodes were interconnected via 100 Mbps Ethernet adapters. To distribute the loads of transcoding and streaming tasks, we divided the cluster into a dual Hadoop cluster: a physical cluster that included a transcoding Hadoop cluster with 13 nodes and a streaming Hadoop cluster with 10 nodes. To implement the CMM component, one node was designated as our management server running Tomcat 7.0 and a second node was designated as a database server running on MySQL, which included a content database and thumbnail database. The management server used JSP to implement the Web-based dashboard, swfupload 2.2.0.1 libraries [36] 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. The HadoopDMT component comprised one name node and 12 data nodes running on HDFS in the transcoding Hadoop cluster. The component used Hadoop 1.0.4 to store original multimedia content, Java 1.6.0_39 (64-bit), and Xuggler 3.4.1012 (64-bit) to implement a MapReduce-based distributed transcoder. The HadoopDMT component had 13 nodes: three streaming server nodes running on NginX [37] and 10 content storage servers. The content storage servers comprised one name node and nine data nodes running on HDFS in the streaming Hadoop cluster. Our software specifications for HadoopDMT included an H.264 streaming module 2.2.7 for implementing 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.

4.2 Prototype

The output from the CloudDMSS prototype is shown in Figs. 10, 11 and 12. The interface provided by the Web-based dashboard module of CMM was designed and implemented for

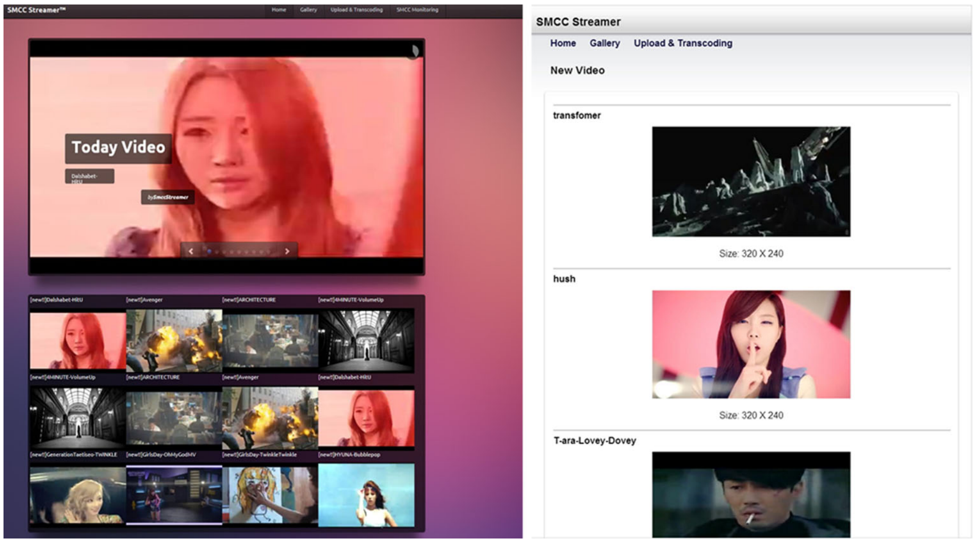
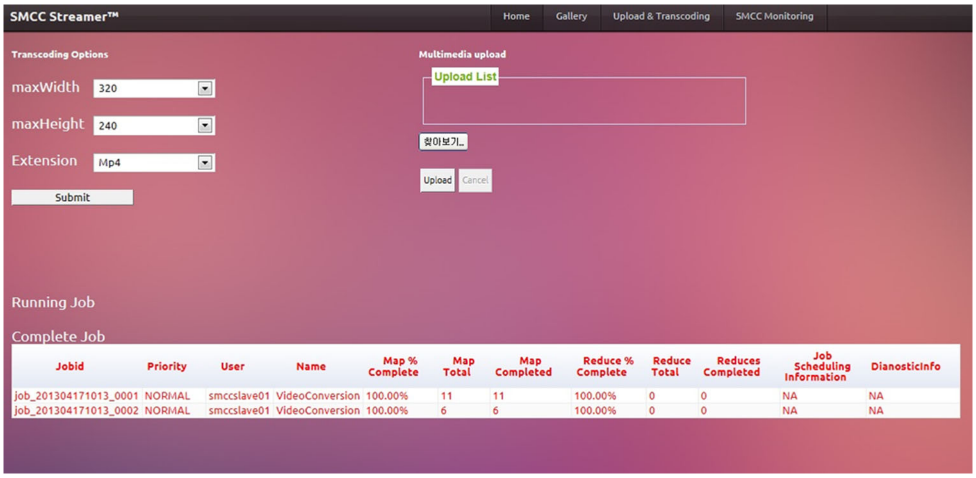


Fig. 10 Front page of the Web-based dashboard for streaming transcoded content in a PC-based Web browser (*left*) and a mobile-based Web browser (*right*)



maxWidth 320
maxHeight 240
Extension Mp4

Running Job

Complete Job

Jobid	Priority	User	Name	Map % Complete	Map Total	Map Completed	Reduce % Complete	Reduce Total	Reduces Completed	Job Scheduling Information	DianosticInfo
job_201304171013_0001	NORMAL	smccslave01	VideoConversion	100.00%	11	11	100.00%	0	0	NA	NA
job_201304171013_0002	NORMAL	smccslave01	VideoConversion	100.00%	6	6	100.00%	0	0	NA	NA

Fig. 11 Web-based dashboard for transcoding tasks and resources

use by various Web browsers such as IE, Firefox, Chrome, and Safari, rather than requiring specific browsers. Figure 10 shows the front page of the Web-based dashboard for streaming transcoded content in a PC-based Web browser (left) and mobile-based Web browser (right), which allows users to select thumbnail images extracted from the transcoded content and menus for uploading, transcoding, and monitoring via the Web interface.

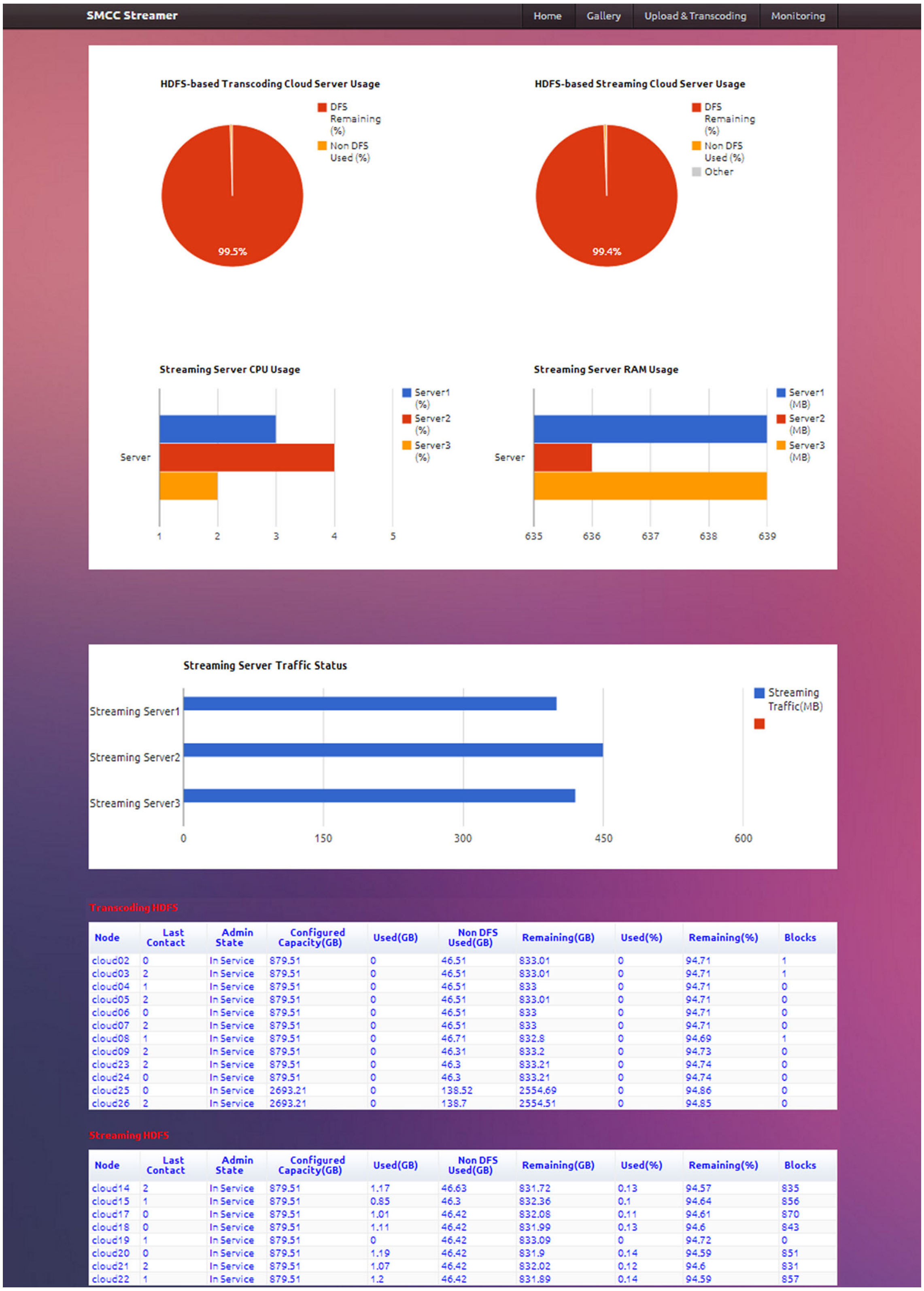
Figure 11 shows a screenshot of the web page used to manage transcoding tasks. Users and administrators can use this page to upload original content, select transcoding options (e.g., resolution, format, and codec), and stream the content to other users. Users can also monitor the progress of ongoing transcoding tasks.

Figure 12 shows a screenshot of the web page used to monitor the status of cluster servers. Users and administrators can use this page to monitor HDFS storage usage in each cluster and the streaming server usage (CPU, RAM, and network traffic) of HadoopDMS in real time.

5 Performance evaluation in a private cloud computing environment

5.1 Experimental environment description

We designed a CloudDMSS architecture, which can be deployed in a private cloud environment. We deployed 28 nodes using a local testbed, i.e., a physical cluster that comprised one management server, one database server, three streaming servers, 13 transcoding servers running on the transcoding Hadoop cluster, and 10 content storage servers running on the streaming Hadoop cluster. Section 4.1 describes the cluster configuration and software specifications in more detail. We conducted three different performance tests to validate the performance of the proposed CloudDMSS. First, to verify the transcoding performance of the transcoding Hadoop cluster, we measured the total transcoding time required to transcode large video files into target files. We compared our Hadoop-based transcoding



Transcoding HDFS

Node	Last Contact	Admin State	Configured Capacity(GB)	Used(GB)	Non DFS Used(GB)	Remaining(GB)	Used(%)	Remaining(%)	Blocks
cloud02	0	In Service	879.51	0	46.51	833.01	0	94.71	1
cloud03	2	In Service	879.51	0	46.51	833.01	0	94.71	1
cloud04	1	In Service	879.51	0	46.51	833	0	94.71	0
cloud05	2	In Service	879.51	0	46.51	833.01	0	94.71	0
cloud06	0	In Service	879.51	0	46.51	833	0	94.71	0
cloud07	2	In Service	879.51	0	46.51	833	0	94.71	0
cloud08	1	In Service	879.51	0	46.71	832.8	0	94.69	1
cloud09	2	In Service	879.51	0	46.31	833.2	0	94.73	0
cloud23	2	In Service	879.51	0	46.3	833.21	0	94.74	0
cloud24	0	In Service	879.51	0	46.3	833.21	0	94.74	0
cloud25	0	In Service	2693.21	0	138.52	2554.69	0	94.86	0
cloud26	2	In Service	2693.21	0	138.7	2554.51	0	94.85	0

Streaming HDFS

Node	Last Contact	Admin State	Configured Capacity(GB)	Used(GB)	Non DFS Used(GB)	Remaining(GB)	Used(%)	Remaining(%)	Blocks
cloud14	2	In Service	879.51	1.17	46.63	831.72	0.13	94.57	835
cloud15	1	In Service	879.51	0.85	46.3	832.36	0.1	94.64	856
cloud17	0	In Service	879.51	1.01	46.42	832.08	0.11	94.61	870
cloud18	0	In Service	879.51	1.11	46.42	831.99	0.13	94.6	843
cloud19	1	In Service	879.51	0	46.42	833.09	0	94.72	0
cloud20	0	In Service	879.51	1.19	46.42	831.9	0.14	94.59	851
cloud21	2	In Service	879.51	1.07	46.42	832.02	0.12	94.6	831
cloud22	1	In Service	879.51	1.2	46.42	831.89	0.14	94.59	857

Fig. 12 Web-based dashboard used to monitor the status of cluster servers

approach with a traditional parallel processing approach: the Media Encoding Cluster [38]. Second, to validate the performance of the SRC algorithm, we compared our system with RR- and LC-based systems in terms of the network transmission rate based on the three streaming servers. Third, we measured the total execution time required for each task in the streaming service deployment process from uploading the original content files to service deployment, as reflected in our Web interface. We divided the streaming service deployment process into 10 steps. Finally, to demonstrate the reliability and robustness of CloudDMSS, we performed a set of experiments to evaluate the behavior of the transcoding process when data node and TaskTracker (Map task) failures occurred.

5.2 Media transcoding performance

In the first experiment, we measured the total transcoding time required to complete the transcoding of a video dataset into a specific target format. We present the results of several experiments conducted using our transcoding Hadoop cluster with 13 computational nodes (one master node and 12 data nodes) and describe the optimal Hadoop options for our system configuration. The parameters for each original and target transcoded video file are listed in Table 1. In the transcoding test, we used six types of video datasets, including several 200 MB video files, which are listed in Table 2.

The following default Hadoop options were used in the transcoding experiment. (1) The number of block replications was set to 3. (2) The block size was set to 64 MB. To verify the efficiency of our system, we conducted three sets of experiments to test: (1) the effects of changes in cluster size on the performance speedup; (2) the effects of different Hadoop options with various block sizes, i.e., 32, 64, 128, 256, and 512; and (3) the effects of different Hadoop options with various block replication factors, i.e., 1, 2, 3, 4, and 5.

Table 1 Parameters for each original and transcoded video file

Parameter	Original video file	Transcoded video file
Codec	Xvid	MPEG-4
Container	AVI	MP4
Size	200 MB	60 MB
Duration	3 min 19 s	3 min 19 s
Resolution	1280 × 720	320 × 240

Table 2 Video datasets used in the performance evaluations

Video dataset						
Size of file (GB)	1	2	4	8	10	20
Number of files	5	10	20	40	50	100

The objective of the first set of experiments was to measure the total transcoding time and speedup with various cluster sizes, i.e., 1, 2, 4, 8, 10, and 12 data nodes, using the Hadoop default options. The speedup (SU) is used to evaluate the effects of parallelism and it is defined as: SU (n) = transcoding time on 1 node/transcoding time on n nodes.

Table 3 shows the transcoding time and speedup results with various cluster sizes. Figures 13 and 14 also show the transcoding time as functions of the cluster size and cluster. According to Table 3, our system obtained excellent transcoding times with very large video files. For example, with 12 data nodes, our system required approximately 2,624 s (ca 44 min) to complete the transcoding process for a 20 GB video dataset with the default Hadoop options.

The SU results indicate the following: (1) our system delivered excellent performance in terms of its parallel and distributed characteristic; (2) the SU performance was better with the 8, 10, and 20 GB datasets compared with the 1, 2, and 4 GB datasets, which suggests that the performance of our system increases with the size of the dataset.

In the second set of experiments, we measured the total time elapsed using different Hadoop options for the block size (default: 64 MB) and block replication factor (default: 3). Hadoop processes large portions of datasets in a parallel and distributed manner after the datasets are split into block sizes of 64 MB. However, users and programmers can change the block size options to improve the data processing performance, depending on the size and type of unstructured data. Furthermore, when large portions of datasets are stored in HDFS, HDFS splits the dataset into fixed size blocks to facilitate rapid searching and processing. With the default Hadoop option for block replication, the replicated data is stored on three data nodes of HDFS to rebalance, move copies around, and continue data replication if system errors occur, such as disk failures or network connection problems. Thus, to verify whether block replication and size factors affected the performance, we measured the time required to complete the media transcoding processing. Five block size options were used in the experiments, i.e., 32, 64, 128, 256, and 512 MB. Five block replication factor values were used, i.e., 1, 2, 3, 4 and 5. Table 4 lists the transcoding times (in s) required with different block sizes and block replication factor values.

Table 4 and Figs. 15, 16 show clearly that the performance of our system was best when the block size was set to 256 and 512 MB, or when the block replication factor was set to three. Thus, it can be concluded that the block size option should be set to a value greater than or close to the original file size to ensure the best distributed video transcoding performance in our system. One video dataset had a file size of 200 MB, thus 256 or 512 MB block sizes provided better performance during transcoding processing. The perfor-

Table 3 Total transcoding time (s) and speedup (SU (n)) with various cluster sizes

No. servers	1	2		4		8		10		12	
Dataset size (GB)	Time	Time	SU (2)	Time	SU (4)	Time	SU (8)	Time	SU (10)	Time	SU (12)
1	1,522	913	1.67	473	3.62	326	4.67	192	7.93	184	8.27
2	2,909	1,702	1.71	781	3.72	475	6.12	355	8.19	349	8.34
4	5,921	3,456	1.71	1,533	3.86	882	6.71	644	9.19	627	9.44
8	11,899	6,785	1.72	3,053	3.89	1632	7.29	1,292	9.21	1,105	10.74
10	14,852	8,612	1.72	3,815	3.90	1989	7.47	1,575	9.43	1,341	11.07
20	29,875	17,235	1.73	7,920	3.88	3791	7.88	3,031	9.86	2,624	11.38

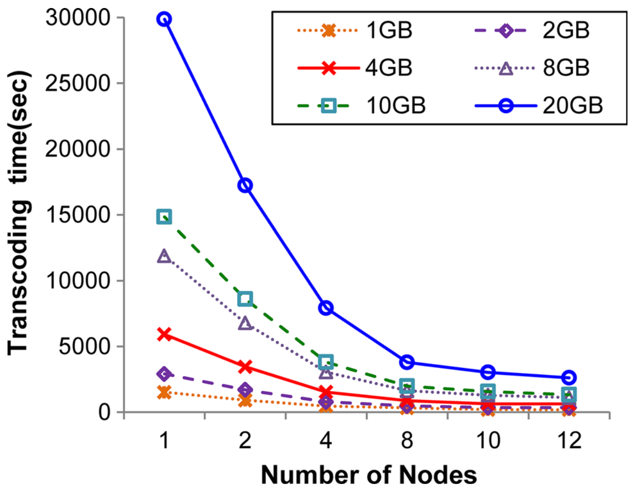


Fig. 13 Transcoding time versus the cluster size

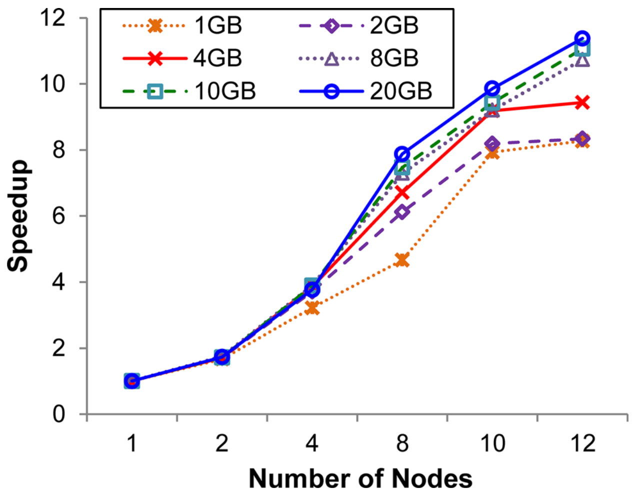


Fig. 14 Speedup versus the cluster size

mance of the transcoding process increased when the block size reached 512 MB because there was a correlation between the file size and block size. HadoopDMT split the input file into blocks of the configured size when the file size was greater than the block size. If the file size was equal to or less than the block size, a block was generated that was equal to the file size. Seven, four, and two blocks were created when the block size was set to 32, 64, and 128 MB, respectively. One file size block was generated when the block size was set to 256 and 512 MB. The number of tasks was determined by the number of divided blocks. The total transcoding time increased if the number of blocks increased because the scheduling time required to process the tasks generated was higher. In addition, the performance was degraded because of the increased delay time required to merge transcoded blocks after transcoding each block. Furthermore, the results showed that the block replication factor should be set to three, which provided the best performance and allowed the distributed systems to process massive media files in a reliable and robust manner, in terms of the recovery from system failure when data loss occurred and when a node failed.

In the third set of experiments, we compared the performance of our Hadoop-based transcoding system with that of the Media Encoding Cluster [38], which is a traditional frame-based parallel transcoding approach. The Media Encoding Cluster is written in C and C++, and it was the first open-source project for frame-based encoding in a distributed and parallel manner that used commercial hardware to reduce the encoding time for a file. To encode original media files into target files, our Hadoop-based transcoding approach splits media files into fixed blocks whereas the Media Encoding Cluster splits media files into frame units.

We tested and compared both approaches using the same video datasets in the same cluster environment. First, we tested the Hadoop-based transcoding approach with 13 nodes. Second, we tested the Media Encoding Cluster approach by configuring the method in our same cluster environment (13 nodes) after shutting down the system configuration of the Hadoop-based transcoding approach. Table 5 lists the total transcoding time results obtained using the two versions of video transcoding. According to Table 5, the Hadoop-based transcoding approach delivered better performance than the Media Encoding Cluster in terms of the execution time for all datasets. The Media Encoding Cluster delivered lower performance than our module because

Table 4 Total transcoding time required for various block sizes and block replications (s)

Dataset size (GB)	Block size option					Block replication				
	32 MB	64 MB	128 MB	256 MB	512 MB	1	2	3	4	5
1	338	178	179	180	181	194	182	178	180	188
2	492	341	242	188	185	361	355	341	338	335
4	961	618	383	248	252	632	625	618	622	625
8	1,831	1,080	621	332	328	1,097	1,089	1,080	1,083	1,086
10	2,277	1,311	749	459	468	1,324	1,315	1,311	1,282	1,280
20	4,422	2,544	1,387	738	752	2,680	2,551	2,544	2,463	2,467

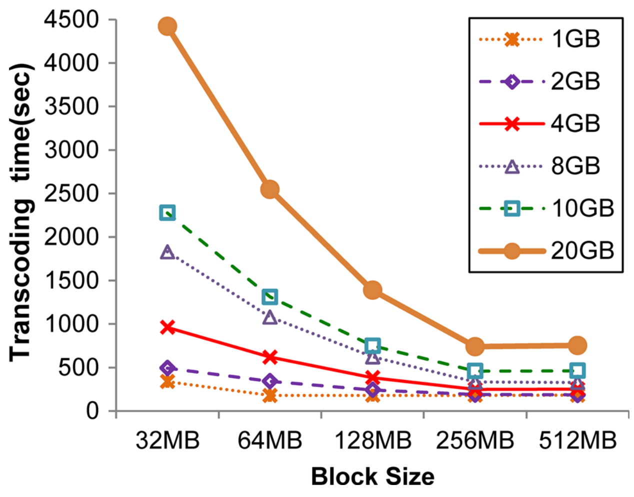


Fig. 15 Total transcoding times (in s) with various block size factors

it incurred high overheads due to the splitting and merging steps applied to the original media files. Our approach split the original video files into 64 MB blocks, which were merged after the transcoding process in MbTD, whereas the Media Encoding Cluster split the original video files into a significantly larger number of frame units compared with the Hadoop-based transcoding module's blocks and it merged the chunked frames into target video files. For a 1 GB dataset (200 MB, 29 frames, 3 min 19 s), our module created 20 chunked blocks of 64 MB, whereas the Media Encoding Cluster produced approximately 29,000 chunked frames.

5.3 Streaming job distribution performance experiment

We proposed a SRC algorithm to balance and distribute the streaming tasks of streaming servers. We evaluated the performance using 10 content storage servers running on the streaming Hadoop cluster, three streaming servers running on NginX, and one management server. Each streaming server had a bandwidth of 100 Mbps. The management server for the Web interface and the SRC algorithm were implemented with JSP in a Tomcat7 environment, and the streaming performance testing tool used to calculate the average transmission rate per streaming server was developed in Java. A dataset was used that included 4 MB MP4 files transcoded from our transcoding task. In the performance evaluation, we simulated 600 virtual users that accessed three streaming systems by applying three algorithms: RR, LC, and SRC. We calcu-

Fig. 16 Total transcoding times (in s) with various block replication factors

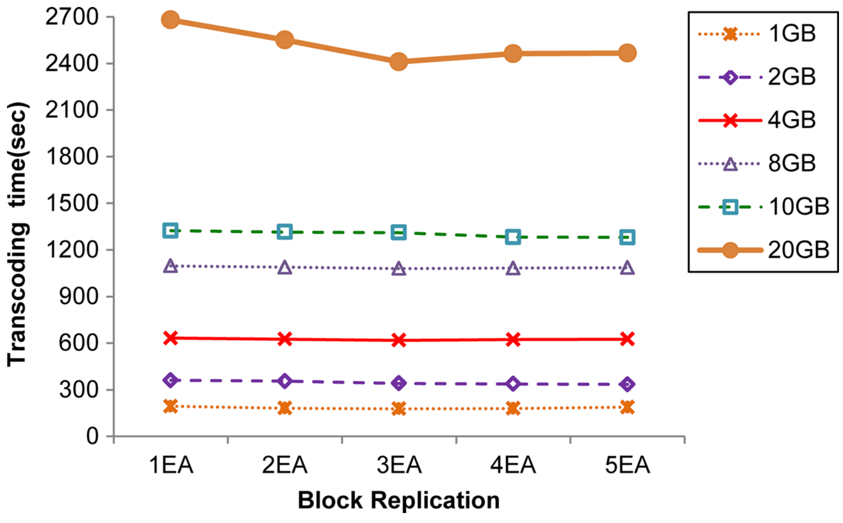


Table 5 Total transcoding times (in s) for both transcoding tasks

Data set size (GB)	The media encoding cluster-based transcoding time (s)	Hadoop-based transcoding time (s)
1	196	178
2	374	341
4	722	618
8	1,482	1,080
10	1,844	1,311
20	3,784	2,544

Table 6 Comparison of the average transmission rate per streaming server using each algorithm-based system

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

lated the overall transmission rate of each streaming server and the average transmission rate per second. Table 6 shows the performance results. The SRC algorithm delivered the best performance in terms of the average transmission rate compared with both the RR and LC algorithms.

5.4 Streaming service deployment with workflow experiment

We defined and described a workflow based on sequential tasks for the streaming service deployment process. To measure the total time required for the deployment process, we divided the workflow of a practical streaming service into 10 steps, from uploading to the deployment of the streaming content on the Web interface. Three datasets were used in this experiment: 1G, 2G, and 4G. Table 7 shows the time required for each task in the deployment process. The total times required for 1G, 2G, and 4G in the deployment process were 288.58 s (approximately 5 min), 546.67 s (9 min), and 997.19 s (16 min), respectively. According to Table 7, the transcoding task required the most time to execute, followed by the uploading task. However, even if the size of the dataset increased, there were no significant differences in the execution times of the streaming tasks. Thus, we can reduce the total execution time for the deployment process by improving the performance of the uploading and transcoding tasks.

5.5 Evaluation of the robustness of CloudDMSS

In this study, we proposed a robust Hadoop-based multimedia streaming service architecture that handles node failures (Sect. 3.2.2) and task failures (Sect. 3.2.1). To demonstrate the reliability and robustness of CloudDMSS, we performed a set of experiments to evaluate the behavior of transcoding process when data node and job (Map task) failures occurred. We experimentally verified the robustness of CloudDMSS in our local testbed (13 nodes) and we used two datasets (10 and 20 GB) with the default Hadoop options described in Sects. 5.1 and 5.2.

First, we measured the lost computing time by artificially injecting data node failures while the normal transcoding process was being performed. The lost computing time was calculated by subtracting the transcoding time with node failures from the normal transcoding time when the number of node failures was zero. If a data node failure occurred, the name node in CloudDMSS detected the situation and performed automatic recovery, including system reconfiguration and block relocation processes using Algorithms 1 and

Table 7 Time required for each task in the streaming service deployment process

Task		Time (s)		
		1G	2G	4G
Upload original contents to HadoopDMT component		93	187	358
Connect SSH shell between CMM and HadoopDMT, and set a folder for transcoding		0.04	0.04	0.04
Transcoding	Set MapReduce job scheduling	9	9	9
	Conduct Map and Reduce task for transcoding	178	341	618
	Clean up Hadoop job step	6	6	6
Extract thumbnail images for transcoded contents		0.66	1.26	2.85
Migrate transcoded contents and extracted thumbnail images to HadoopDMS		0.66	1.12	2.01
Delete original and transcoded contents, and thumbnail images in HadoopDMT		0.21	0.22	0.24
Register thumbnail images and content information in database server of HadoopDMS		0.01	0.03	0.05
Deploy streaming contents for the streaming service on the Web interface		1	1	1
Total		288.58	546.67	997.19

Table 8 Lost computing time caused by data node failures in the local testbed

Number of node failures	Dataset							
	10 GB				20 GB			
	Total blocks	Lost blocks	Time (s)	Lost time (s)	Total blocks	Lost blocks	Time (s)	Lost time (s)
0	600	0	1,331	0	1,200	0	2,544	0
1	600	48	1,480	149	1,200	94	2,880	336
2	600	92	1,590	259	1,200	181	3,155	611
3	600	151	1,735	404	1,200	278	3,386	842
4	600	186	1,914	583	1,200	342	3,543	999
5	600	219	2,175	844	1,200	456	3,875	1,331

Table 9 Amount of lost computing time caused by job (TaskTracker) failures in the local testbed

Number of Task-Tracker failures	Dataset							
	10 GB				20 GB			
	Map tasks	Failed tasks	Time (s)	Lost time (s)	Map tasks	Failed tasks	Time (s)	Lost time (s)
0	200	0	1,331	0	400	0	2,544	0
1	200	2	1,428	97	400	2	2,722	178
2	200	4	1,511	180	400	4	2,841	297
3	200	6	1,600	269	400	6	2,905	361
4	200	8	1,622	291	400	8	2,983	439
5	200	10	1,747	416	400	10	3,064	520

2. Table 8 shows the lost computing time caused by data node failures. The normal times required to transcode 10 and 20 GB datasets were 1,311 and 2,544 s. When a data node failure was generated among 13 nodes, the losses were 48 of 600 blocks with the 10 GB dataset and 94 of 1,200 blocks with the 20 GB dataset, including replicas. Thus, the lost computing times with the 10 and 20 GB datasets were calculated as 149 and 336 s, respectively, with one data node failure. With four data node failures, i.e., one-third of the 13 nodes, the recovery times for the 10 and 20 GB datasets using CloudDMSS were calculated as approximately 10 and 17 min. Based on this evaluation of the data node failures, we experimentally verified that our system delivered excellent performance and it was robust to node failures.

Second, we measured the lost computing time after artificially injecting TaskTracker failures. Our system performed a MapReduce-based distributed transcoding process, which was scheduled by JobTracker and TaskTracker, as described in Sect. 3.2.1. When a 10 GB dataset that included 50 × 200 MB files was transcoded using the default block size option (64 MB), 200 Map tasks were generated. After determining the number of tasks, JobTracker assigned the Map tasks to TaskTrackers and each TaskTracker performed its assigned tasks. If TaskTracker failed, CloudDMSS compensated for the task failures by rescheduling and reassigning the remaining tasks, including the failed tasks. Table 9 shows the lost computing time caused by TaskTracker failures in the local testbed. According to Table 9, when four TaskTrackers were generated, the lost computing times with the 10 and 20 GB datasets were 291 and 439 s, respectively. Thus, the recovery times for task failures with the 10 and 20 GB datasets using our system were approximately 5 min and 7 min, respectively, in our local testbed. According to the performance evaluation based on TaskTracker failures, our system was robust to task failures.

6 Performance evaluation in a public cloud computing environment

6.1 Experimental environment description

We conducted a performance evaluation using our local testbed environment in a LAN environment. Our cloud computing environment did not consider the unpredictable and unstable network traffic generated by commercial and public cloud computing environments, such as Amazon EC2 and Rackspace. Thus, to demonstrate the validity of CloudDMSS in commercial and public cloud computing environments over WAN, we performed several performance evaluations

Table 10 Comparison of the overall system configurations in Cloudit 2.0 and the local testbed

Type	Cloudit 2.0 (50 nodes)	Local testbed (28 nodes)
Management server	1 EA	1 EA
Database server	1 EA	1 EA
Transcoding server	28 EA	13 EA
Streaming server	10 EA	3 EA
Content server	10 EA	10 EA

in an actual cloud environment, Cloudit 2.0 [39], which is operated by Innogrid. Table 10 compares the overall system configurations of CloudDMSS in Cloudit 2.0 and that of our local testbed.

We deployed 50 virtual machines (50 VMs) provided by Cloudit 2.0, i.e., one management server, one database server, 10 streaming servers, 28 transcoding servers, and 10 content storage servers. Each VM comprised Linux OS (Ubuntu 12.04 LTS) running on four virtual cores with the equivalent of Intel Xeon quad-core 2.13 GHz processors, 8 GB of memory, and 100 GB of disk space on a shared hard drive. Cloudit 2.0 utilized Xen [40] virtualization software. The other environment, including the software specification (refer to Sect. 4) and the algorithms, was the same as the local testbed environment. We also used seven types of datasets, i.e., the six datasets listed in Table 2 and an additional 50 GB dataset.

6.2 Performance evaluations

The first set of experiments was similar to the performance evaluation conducted in the first part of Sect. 5.2. We measured the total transcoding time and SU with various cluster sizes, i.e., 1, 4, 8, 12, 16, 20, 24, and 28 data nodes, using the default Hadoop options. Table 11 and Fig. 17 show the total transcoding times with various cluster sizes. The SU results indicated the following. First, although the SU results were slightly different depending on the size of the dataset, our system delivered good performance in terms of its parallel and distributed characteristics up to 12 nodes. However, the performance improved only slightly 16 nodes to 28 nodes. For example, the results with 4, 8, and 12 nodes and the 20 GB dataset showed that the performance improved by approximately 4, 8, and 9 times compared with one node. By contrast, the results with 16, 20, 24, and 28 nodes improved only slightly to about 9.8, 10.6, 11.8, and 12.4 times, respectively. Second, our system had a higher capacity for effective distributed processing as the size of the dataset increased. There were performance improvements of about 57 % for 4 GB, 78 % for 10 GB, and 98 % for 50 GB compared with the transcoding of 1 GB using 28 nodes. Finally, although commercial cloud computing services run on unpredictable and unstable networks with VM I/O traffic, our CloudDMSS

Table 11 Total transcoding time (s) and speedup with various cluster size

Cluster size	Dataset size						
	1 GB	2 GB	4 GB	8 GB	10 GB	20 GB	50 GB
Total transcoding time (s)							
1	1,491	3,032	5,897	11,781	14,126	29,199	81,610
4	467	762	1,516	2,965	3,710	7,860	16,440
8	306	465	826	1,569	1,890	3,651	8,730
12	252	427	642	1,330	1,643	3,212	7,818
16	245	351	621	1,285	1,580	2,985	7,190
20	232	342	609	1,215	1,463	2,762	6,883
24	227	332	589	1,057	1,236	2,472	6,315
28	225	325	565	1,022	1,194	2,358	6,204
Speedup							
1	1	1	1	1	1	1	1
4	3.19	3.98	3.89	3.97	3.81	3.71	4.96
8	4.87	6.52	7.14	7.51	7.47	7.99	9.35
12	5.92	7.1	9.19	8.86	8.6	9.09	10.44
16	6.09	8.64	9.5	9.17	8.94	9.78	11.35
20	6.43	8.87	9.68	9.7	9.66	10.57	11.86
24	6.57	9.13	10.01	11.15	11.43	11.81	12.92
28	6.63	9.33	10.44	11.53	11.83	12.38	13.15

provided a stable and effective distributed transcoding service in an actual cloud computing environment based on all the performance evaluations.

In the second experiment, we experimentally verified the performance of streaming job distribution using the same software specifications and simulation conditions described in Sect. 5.3. The only difference from the previous evaluation was the system configuration. To demonstrate the scalability of our system, we tested the performance using a streaming Hadoop cluster that comprised 10 streaming servers and 10 content storage servers. Each streaming server had a bandwidth of 100 Mbps. We calculated the overall transmission rate for each streaming server and the average transmission rate per second with a bandwidth of 100 Mbps. Table 12 shows that the performance levels were similar to the results in Table 6. Our SRC algorithm delivered better performance in terms of the average transmission rate compared with the RR and LC algorithms. The total transmission rate per second with the three streaming servers on the local testbed was about 10 MB while the rate per second in the 10 servers of Cloudit 2.0 was about 20 MB. Thus, our CloudDMSS effectively distributed the streaming jobs requested by numerous users in an actual cloud computing environment with a similar performance to that on the local testbed.

In the final experiment, we measured the total time required for the workflow of the sequential tasks, as defined and described in Sect. 3.3. We also performed an experiment

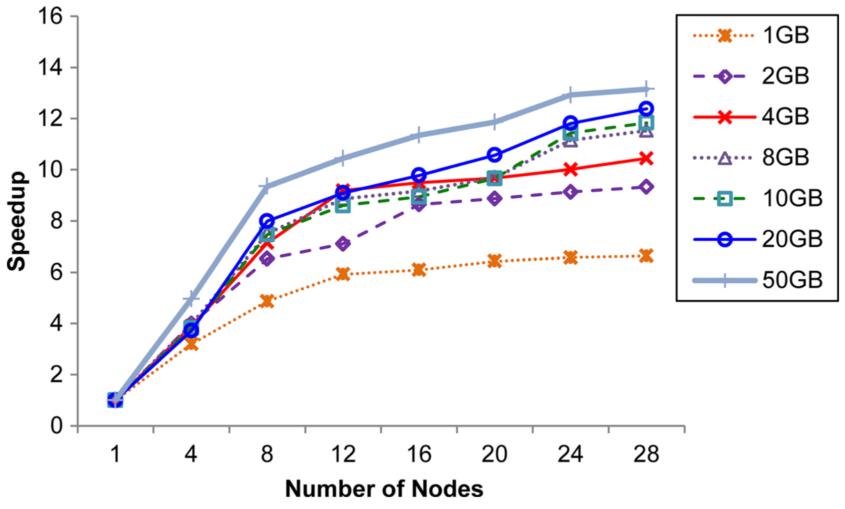


Fig. 17 Speedup versus cluster size

Table 12 Comparison of the average transmission rates of each streaming server using each algorithm-based system

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

using three datasets, i.e., 1G, 2G, and 4G. The overall system configuration was the same as that of the local testbed, except for the transcoding servers. To facilitate a comparison with the results obtained in the same environment, as described in Sect. 5.4, we deployed 13 VMs as transcoding servers among 28 VMs running on the transcoding Hadoop cluster in CloudDMSS. According to Table 13, the total times required by 1G, 2G, and 4G for the overall deployment process were 357.93 s (approximately 6 min), 590.23 s (10 min), and 1,022.14 s (17 min), respectively. These results were very similar to those reported in Sect. 5.4, which shows that our CloudDMSS performs well in public cloud and actual cloud environments. The times required to perform the deployment process for the three datasets differed by approximately 1 min, respectively, compared with the results obtained using the local test bed. These differences in the execution time were attributable to the delay times caused by the upload and transcoding tasks

Table 13 Time required for each task during the streaming service deployment process

Task		Time (s)		
		1G	2G	4G
Upload original contents to HadoopDMT component		98	220	372
Connect SSH shell between CMM and HadoopDMT and set a folder for transcoding		0.08	0.08	0.08
Transcoding	Set MapReduce job scheduling	9	9	9
	Conduct Map and Reduce task for transcoding	243	352	631
	Clean up Hadoop job step	6	6	6
Extract thumbnail images for transcoded contents		0.09	0.5	0.18
Migrate transcoded contents and extracted thumbnail images to HadoopDMS		0.53	1.42	2.64
Delete original and transcoded contents, and thumbnail images in HadoopDMT		0.22	0.22	0.23
Register thumbnail images and content information in database server of HadoopDMS		0.01	0.01	0.01
Deploy streaming contents for streaming service in the Web interface		1	1	1
Total		357.93	590.23	1022.14

in the overall process. The upload task required more time than that on the local testbed because there was an unexpected network bottleneck in the real public cloud environment. Furthermore, the transcoding task required more time in the real cloud environment than the local testbed because of an I/O virtualization bottleneck due to the sharing of physical resources by many users.

The three different experiments conducted using Cloudit 2.0 demonstrate that our CloudDMSS performed well in terms of the transcoding process, the average transmission rate, and the deployment process for the streaming service in the local testbed environment and in the commercial and public cloud computing environment over WAN with unpredictable and unstable network traffic.

7 Conclusion and future work

In this study, we focused on designing a robust cloud-based distributed multimedia streaming service with transcoding, which we call CloudDMSS. CloudDMSS is based on Hadoop clustering and it provides a seamless streaming service with suitable QoS over the current Internet. Our system was developed to perform the workflow of the streaming service deployment process automatically, including uploading, transcoding, migration, thumbnail extraction, content registration, and streaming job distribution, after a single click of a content upload button on the provided Web interface. In this study, we describe the design of CloudDMSS and its workflow, as well as four important algorithms for content replication, system recovery on HadoopDMS, management for CMM, and SRC for streaming job distribution. We determined a suitable hardware configuration for the current cluster environment based on a dual Hadoop cluster and we report on the implementation of our system.

To validate the performance of the proposed CloudDMSS with the job distribution scheme using the SRC algorithm, we conducted four different tests on a local testbed to evaluate the transcoding task, streaming job distribution conducted by SRC, streaming service deployment, and the robustness to data node and task failures. We demonstrated the excellent performance of our system and identified the optimal Hadoop options for media transcoding. When the block size option was set to a value greater than or close to the original file size and the block replication factor value was set to 3, our system delivered good performance for media transcoding processes. We also confirmed that the SRC algorithm had a better average transmission rate than the RR and LC algorithms for the streaming job distribution task. Finally, the streaming service deployment experiment showed that the total execution time required for the deployment process can be decreased by reducing the uploading and transcoding execution times. To demonstrate the validity and scalability of our system in a commercial cloud computing environment, we also conducted several experiments in Cloudit 2.0, which were similar to the experiments conducted on the local testbed. These experiments verified that CloudDMSS performed well in transcoding, streaming job distribution using SRC, and streaming service deployment.

In future research, we plan to improve the transcoding performance by developing improved content splitting and merging algorithms on the Hadoop cluster. In addition, we plan to implement our system on commercial cloud services such as Amazon EC2, Rackspace, and KT ucloud.

Acknowledgments This research was supported by the MSIP (Ministry of Science, ICT & Future Planning), Korea, under the ITRC (Information Technology Research Center) support program (NIPA-2013-H0301-13-3006) supervised by the NIPA (National IT Industry Promotion Agency)

References

1. Dirk, W., Andreas, E., Alexandra, C.: An ecosystem for user-generated mobile services. J. Converg. **3**(4), 43–48 (2012)
2. Barlas, G.: Cluster-based optimized parallel video transcoding. Parallel Comput. **38**(4–5), 226–244 (2012)
3. Ratakonda, K., Turaga, D.S., Lai, J.: QoS support for streaming media using a multimedia server cluster. In: IEEE GLOBECOM
4. Guo, J., Chen, F., Bhuyan, L., Kumar, R.: A cluster-based active router architecture supporting video/audio stream transcoding service. In: Proceedings of Parallel and Distributed Processing Symposium (2003)
5. Seo, D., Lee, J., Kim, Y., Choi, C., Jung, I.: Load distribution strategies in cluster-based transcoding servers for mobile clients. In: Proceedings of ICCSA, pp. 1156–1165 (2006)
6. Strufe, T.: llmStream: efficient multimedia streaming in decentralized distributed systems. In: IFMIP, pp. 63–68 (2004)
7. Kim, M., Han, S., Cui, Y., Lee, H., Jeong, C.: A Hadoop-based multimedia transcoding system for processing social media in the PaaS platform of SMCCSE. KSII Trans. Internet Inf. Syst. **6**(11), 2827–2848 (2012)
8. Buyya, R., Yeo, C.S., Venugopal, S., Broberg, J., Brandi, I.: Cloud computing and emerging IT platforms: vision, hype, and reality for delivering computing as the 5th utility. Future Gener. Comput. Syst. **25**(6), 599–616 (2009)
9. Vouk, M.A.: Cloud computing: issues, research, and implementations. In: ITI, pp. 31–40 (2008)
10. Armbrust, M., Fox, A., Griffith, R., Joseph, A.D., Katz, R., Konwinski, A., Lee, G., Patterson, D., Rabkin, A., Stoica, I., Zaharia, M.: A view of cloud computing. Commun. ACM **53**(4), 50–58 (2010)
11. Grossman, R.L.: The case for cloud computing. IT Prof. **11**(2), 23–27 (2009)
12. Zhang, Q., Cheng, L., Boutaba, R.: Cloud computing: state of the art and research challenges. J. Internet Ser. A **1**(1), 7–18 (2010)
13. Pan, Y., Zhang, J.: Parallel programming on cloud computing platforms: challenges and solutions. J. Converg. **3**(4), 23–28 (2012)
14. Lei, Z.: Media transcoding for pervasive computing. In: ACM Conference on Multimedia Workshop, pp. 459–460 (2001)
15. Lee, H.S., Lim, K.H., Kim, S.J.: A configuration scheme for connectivity-aware mobile P2P networks for efficient mobile cloud-based video streaming services. Clust. Comput. 1–12 (2013)

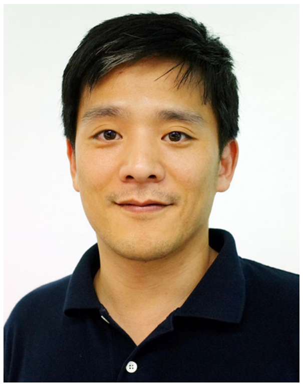
16. Zhu, W., Luo, C., Wang, J., Li, S.: Multimedia cloud computing. IEEE Signal Process. Mag. **28**(3), 59–69 (2011)
17. Park, S., Jung, I.Y., Eom, H., Yeom, H.Y.: An analysis of replication enhancement for a high availability cluster. J. Inf. Process. Syst. **9**(2), 205–216 (2013)
18. Apache Hadoop. http://hadoop.apache.org/. Accessed 5 May 2013
19. Joey, J.: Introduction to Hadoop.http://i.dell.com/sites/content/business/solutions/whitepapers/en/Documents/hadoop-introduction.pdf. Accessed 2 Aug 2013
20. HDFS architecture guide. http://hadoop.apache.org/docs/stable/hadoop-project-dist/hadoop-hdfs/HdfsDesign.html. Accessed 18 Aug 2013
21. Shvachko, K., Kuang, H., Radia, S., Chansler, R.: The Hadoop distributed file system. In: MSST
22. Ghemawat, S., Gobioff, H., Leung, S.T.: The Google file system. Oper. Syst. Rev. ACM **37**(5), 29–43 (2003)
23. Poellabauer, C., Schwan, K.: Energy-aware media transcoding in wireless systems. In: IEEE PerCom, pp. 135–144 (2004)
24. Roy, S., Shen, B., Sundaram, V.: Application level hand off support for mobile media transcoding sessions. In: Workshop on Network and Operating Systems, pp. 95–104 (2002)
25. Liao, X., Jin, H.: A new distributed storage scheme for cluster video server. J. Syst. Archit. **51**(2), 79–94 (2005)
26. Sambe, Y., Watanabe, S., Yu, D., Nakamura, T., Wakamiya, N.: High-speed distributed video transcoding for multiple rates and formats. IEICE Trans. Inf. Syst. **E88–D**(8), 1923–1931 (2005)
27. Tian, Z., Xue, J., Hu, W., Xu, T., Zheng, N.: High performance cluster-based transcoder. In: Proceedings of ICCASM, pp. 248–252 (2010)
28. Hui, W., Lin, C., Yang, Y.: MediaCloud: a new paradigm of multimedia computing. KSII Trans. Internet Inf. Syst. **6**(4), 1153–1170 (2012)
29. Luo, H., Egbert, A., Stahlhut, T.: QoS architecture for cloud-based media computing. In: ICSESS, pp. 769–772 (2012)
30. Chang, S.Y., Lai, C.F., Huang, Y.H.: Dynamic adjustable multimedia streaming service architecture over cloud computing. Comput. Commun. **35**(15), 1798–1808 (2012)
31. Huang, Z., Mei, C., Li, L.E., Woo, T.: CloudStream: delivering high-quality streaming videos through a cloud-based SVC proxy. In: Proceedings of IEEE INFOCOM, pp. 201–205 (2011)
32. Dean, J., Ghemawat, S.: MapReduce: simplified data processing on large clusters. Commun. ACM **51**(1), 107–113 (2008)
33. Xuggler Java Library. http://www.xuggle.com/xuggler/index. Accessed 2 May 2013
34. Kim, M., Cui, Y., Han, S., Lee, H.P.: Towards efficient design and implementation of a Hadoop-based distributed video transcoding system in cloud computing environment. J. Multimed. Ubiquitous Eng. **8**(2), 213–224 (2013)
35. Piorkowski, A., Kempny, A., Hajduk, A., Strzelczyk, J.: Load balancing for heterogeneous web servers. CN, pp. 189–198 (2010)
36. Swfupload. https://code.google.com/p/swfupload/. Accessed 10 Aug 2013
37. NginX. http://nginx.org/. Accessed 10 Aug 2013
38. Media Encoding Cluster Project. http://www.encodingcluster.com. Accessed 20 July 2013
39. Cloudit 2.0. http://www.cloudit.co.kr/. Accessed 13 Jan 2014
40. Barham, P., Dragovic, B., Fraser, K., Hand, S., Harris, T., Ho, A., Pratt, I., Warfield, A.: Xen and the art of virtualization. In: Proceedings of ACM SOSP, pp. 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 realtime 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.

Yun Cui 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 an associate 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.



Hogyeon Cho is a CEO at Innogrid co., ltd. He received a B.E. degree in Naval Architecture and Ocean Engineering from Seoul National University in 2005. Before joining Innogrid, he had a lot of experience in planning, marketing and sales. He joined Innogrid in 2007 as a director of Cloud Business Department, and has directed various projects e.g. the buildup of WMO (World Meteorological Organization) Global Information System Center and the development of Cloudit^{TM}, the only total cloud computing solutions in Korea. He was appointed to the CEO in March, 2014.

Sungdae Hwang received a B.E. degree in Information and Communication Engineering from Suwon University in 2001. He has performed a lot of tasks concerning servers, storage, and network engineering, and he managed large-scale infrastructure at Interpark INT. At present, he works for Innogrid co., ltd., and is now a team manager at the cloud computing engineering team.