Journal of Knowledge Information Technology and Systems(JKITS), Vol. 9, No. 5, pp. 607~615, October 2014

Journal of Knowledge Information Technology and Systems

ISSN 1975-7700

http://www.kkits.or.kr

The VOD Service is Based on Cloud File System Which is Used to Fixed-Node Server of Private Dedicated Network

Myeong-Ki Jung, Jae-Woong Kim*

Department of Computer Engineering, Kongju National University

ABSTRACT

A company or organization that provides a VOD service inside the media by using a private dedicated network, in preparation for the connection's rush to the service at the same time, is adopted how to get service from the relay server or the branch point or that and transfer the specific media to the relay ser ver. However, these methods have a problem that needs to be added the storage of the relay server, wh ich increases the number and size of the media. In order to solve this problem, the method to distribute to a plurality of relay servers that you specify only a portion by dividing the media is used. Also, in o rder to be able to reduce the load connected to the main streaming server, they allow the list of splittin g media files to be created and to be viewed by selecting the optimum relay server among a plurality o f relay servers, which are stored and divided into a base point of a branch point or that. Then it is pos sible to realize a VOD services with a stability, lower cost, and high efficiency.

KEYWORDS: Lowest Common Ancestor, HTTP Live Streaming, Relay Servers, VOD Streaming

ARTICLE INFO: Received 12 September 2014, Revised 10 October 2014, Accepted 10 October 2014.

1. 서 론

기업 또는 기관에서 VOD 스트리밍 솔루션을 도입하는 경우의 대부분은 VOD 스트리밍서버 한 개의 시스템만을 갖추고 있는 것이 일반적인 상황이다. 이러한 경우는 대개의 경우 활용도가 낮거나 VOD를 시청하는 시청자 수가 적은 경우이다. 그러나 시청해야 하는 인원이 서버나 네트워크 장비의 수용범위를 넘어가고, 어느 기간 내에 의무적으로 시청해야 하는 경우가 발생하면 많은 시청자가 접속하여 VOD를 제대로 시청할 수 없게 된다. 또

*Corresponding author is with the Department of Computer Engineering, Kongju National University, 275, Budae-dong, Seobuk-gu, Cheonan-si, Chungcheongnam-do, (1223-24, cheonandae-ro), 331-717, KOREA.
E-mail address: jwkim@kongju.ac.kr

한 VOD의 특성 상 순간적인 트래픽의 몰림이 아닌 해당 VOD의 영상 재생 시간만큼의 트래픽이 발생하게 되므로 기업 또는 기관의 네트워크 장비에 무리를 주게 되어 해당 기관 또는 기업의 실제 업무를 진행하지 못할 수 도 있다. 그래서 이런 집중 현상을 타개하기 위해 로드밸런싱을 이용하는 방법외의 여러 방법이 이용한다. 그 중에 한 가지는 각 지점 또는 지사 별로 중계 서버를 설치하여 시청 부하를 분산하는 방법을 사용하는 것이다. 이 방법의 특징은 메인 VOD 서버에 저장되어 있는 모든 동영상 파일을 각각의 지점/지사의 중계 서버에 저장하는 방식이 아닌 일부의 파일 만을 복제하는 것이 특징이다. 그러나 이러한 방식은 관리자가 복제해야 할 동영상 파일들을 지속적으로 관리하지 않는다면 중계 서버의 복제 저장 용량이 남아있지 않게 되어 다시 메인 VOD 서버로 트래픽이 집중 되는 현상이 나타날 수 있다. 이에 본 논문에서는 클라우드 파일 시스템을 중계 서버를 통해 구현하여 VOD 동영상 파일을 부분 및 다중 복제하여 트래픽의 중앙 집중을 막고 또한 해당 동영상의 무결성을 보장하는 방법을 설계, 구현하고자 한다.

2. 관련연구

2.1 클라우드 파일 시스템

클라우드 컴퓨팅에서 가장 중요한 요소는 분산파일시스템이다. 어떠한 분산파일시스템을 사용해야 하는지는, 분산파일시스템의 요구사항이라 할 수 있는 서버 및 네트워크 장비의 저비용 요구, 손쉽게 확장이 가능한지, 얼마나 안정적인지, 요구한 성능을 보장하는지, 시스템관리자가 얼마나 간편하게 사용가능한지, 데이터관리의 보안성 및 무결성을 확보할 수 있는지를 검토한 후 선택해야 한다. 현재 여러 분산파일시스템이 연구되어지고 있고 또한 사용되어지고 있으나 그중 현재 가장 대표적인 분산파일시스템은 Google에서 개발하여 발표한 Google File System[1](이하 GFS)과 아마존, IBM등 여러 IT 기업들의 클라우드 컴퓨팅 플랫폼의 기반이 되는, 오픈소스 소프트웨어 개발 프로젝트인 Hadoop 분산 파일 시스템[2](이하 HDFS) 등이 있다.

GFS는 <그림 1>에서와 같이 여러 클라이언트들이 접근하는, 모든 파일 시스템 메타데이터를 관리 및 제어를 담당하는 단일 마스터와 청크(chunk)라 불리는 64MB 크기의 단위로 나누어진 파일들을 분

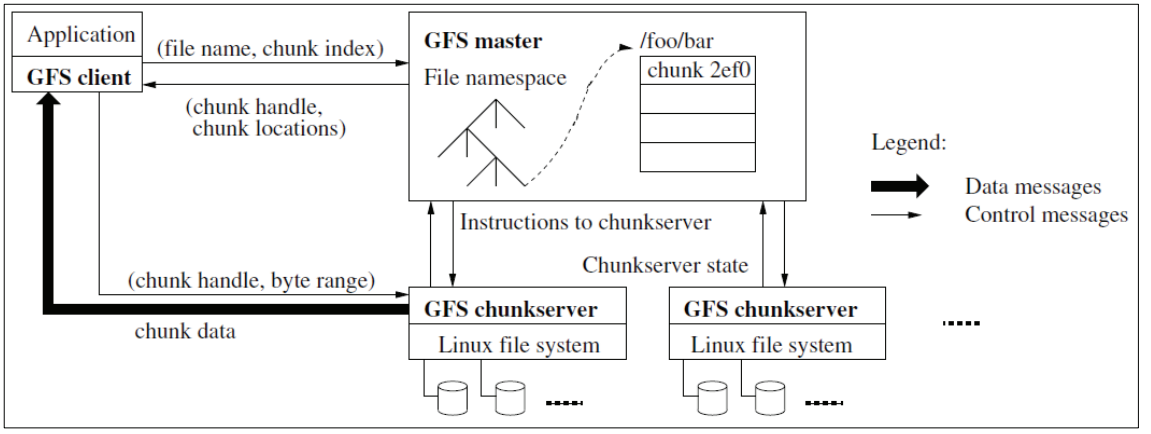

그림 1. GFS의 구조
Figure 1. GFS architecture

산 저장하는 여러 청크 서버들로 구성된다. 이렇게 구성된 GFS는 데이터 흐름과 메타데이터가 분리가 된 비대칭성 구조를 가지며, 이를 통해 확장성과 고성능을 보장하고 또한 장애에 적절히 대처할 수 있는 안정성을 확보하게 된다.

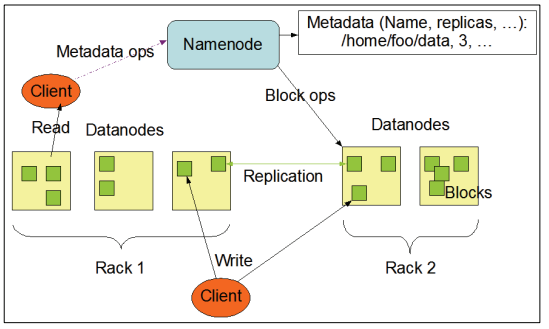

그림 2. HDFS의 구조
Figure 2. HDFS architecture

HDFS는 GFS와 이름만 다를 뿐 동일한 구조와 기능을 제공한다. <그림 2>에서 네임노드는 GFS의 마스터 역할을 하며, 데이터노드는 GFS의 청크서버와 기능이 동일하다. HDFS는 GFS를 기반으로 개발된 분산 파일 시스템이기 때문이긴 하나 플랫폼 간의 이식을 손쉽게 하기 위해 자바를 이용하여 구현된 접이 특징이다.

이 외에 Amazon S3 파일 시스템[3], 병렬 네트워크 파일시스템[4], CloudStore[5] 등이 있으며 국내에서는 ETRI에서 발표한 GLORY-FS 파일시스템[6]등이 있다.

이러한 분산 파일 시스템들은 주로 클라우드 컴퓨팅의 주요한 파일 시스템으로 채택되고 있어 클라우드 파일 시스템이라고도 한다.

2.2 최소 공통 조상 알고리즘

공통된 루트노드를 가진 두 노드간의 최소 공통 조상(Lowest Common Ancestor, 이하 LCA)을 찾는 문제는 Harel과 Tarjan이 해당 트리를 먼저 선 처리한 후 쿼리가 일정 시간에 응답 할 수 있다는 것을 보여주었다[7]. 이후 문자열 처리 및 계산이나 생물학 등 수많은 어플리케이션에서 트리형 구조를 가지는 알고리즘 문제를 해결하는데 사용되어지고 있다.

```
function TarjanOLCA(u)
    MakeSet(u);
    u.ancestor := u;
    for each v in u.children do
        TarjanOLCA(v);
        Union(u,v);
        Find(u).ancestor := u;
    u.colour := black;
    for each v such that {u,v} in P do
        if v.colour == black
            print "Tarjan's Lowest Common Ancestor of " + u +
                  " and " + v + " is " + Find(v).ancestor + ".";
```

그림 3. Tarjan LCA 알고리즘
Figure 3. Tarjan LCA Algorithm

<그림 3>은 트리 R에서 주어진 각각의 노드 P의 가장 최소의 공통 조상을 결정하는 알고리즘이다. 이 알고리즘의 경우에는 P를 사전에 지정하고. 분리 집합 가지(disjoint-set forest) 형태의 트리를 위해 MakeSet(), Union(), Find() 함수를 사용한다[8]. 또한 Range Minimum Query (이하 RMQ)를 이용하여 시간복잡도 O(sqrt(N))를 O(N)로 단순화 시키는 알고리즘도 연구되어 있다[9].

위의 알고리즘을 이용하여 출발 노드에서 도착 노드까지의 LCA 노드를 구하고 출발 노드에서 LCA 노드까지의 노드의 거리와 목적 노드와 LCA 노드까지의 거리를 합산하여 전체 거리를 구하여 사용한다.

3. 본론

본 논문에서 제안하는 방법은 사설전용망 내에 있는 Fixed-node 서버(이하 중계서버)를 클라우드 파일시스템을 응용하여 미디어들을 전용망 또는 중계서버들의 조건에 맞게 분산 배치하여 중계서버의 작은 저장 용량으로도 메인 서버의 스토리지 용량을 감당할 수 있게 한다. 또한 이렇게 분산된 파일들을 접근 시점에 맞게 중계서버 및 네트워크 상태를 고려하여 클라이언트에게 제공하여 서비스하는 방법을 제시한다.

3.1 미디어 배포 프로세스

동영상 미디어를 배포하기 위한 사전 준비로서 MPEG-4 형태로 인코딩하여 파일을 서버에 업로드하고 해당 파일들을 MPEG-DASH[10] 또는 HTTP Live Streaming[11](이하 HLS) 방식을 사용하기 위해 부분 파일(Segment file)로 분할한다.

HLS의 구조는 <그림 4>에서와 같이 카메라로 촬영한 영상을 서버에 전송하여 media encoder를 이용하여 MPEG-2 Transport Stream으로 인코딩을 한 후 stream segmenter를 이용하여 인코딩된 영상을 일정한 시간 간격으로 분할해 파일을 만든다. 또한 그 분할된 파일들을 접근하여 재생할 수 있도록 하기 위해 재생목록 메타데이터(파일 확장자명 m3u8)을 생성한다. 이렇게 분할된 미디어 파일들과 메타데이터 파일을 일반적인 웹서버에 저장한다. 이러한 미디어들을 클라이언트들이 접근하여 재생하고자 할 때에는 재생목록 메타데이터를 우선 다운로드한다. 다운로드된 재생목록 메타데이터 내의 분할 구성된 미디어 파일들의 정보를 이용하여 재생할 시점의 분할 동영상 파일을 다운로드하고 재생하면서 재생이 끊어지지 않게 다음 시점의 분할 동영상 파일을 다운로드하여 재생한다. 이러한 방식으로 분할된 동영상이 존재하지 않을 때까지 반복한다.

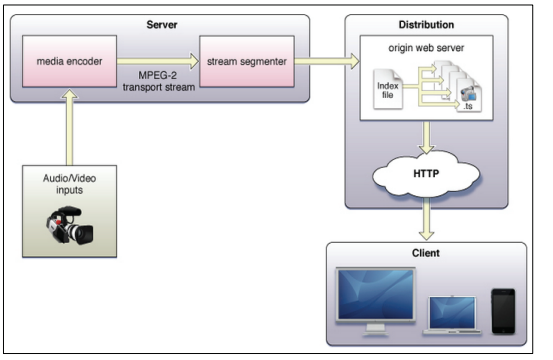

그림 4. HLS 구조
Figure 4. HLS architecture

본 논문에서는 VOD의 경우 재생 시점을 초 기준으로 이동할 수 있게 하기 위해 3초 단위로 분할하였다. 원래 1초 단위의 재생 시점 이동이 필요하나 3초 이하의 경우에는 너무 많은 분할 파일이 존재하고 오히려 효율이 나빠진다는 단점이 있기 때문이다.

```
result_RPC_ID ← get_RPC_ID_list();
count_RPC ← count_RPC( result_RPC_ID );
count_ts ← count_segment_file();
nScale ← get_backup_scale();
nCnt ← 0;
for ( i = 0; i < count_ts; i++ ) {
  for ( j = 0; j < nScale; j++) {
    segment_name ← filename + i + ".ts";
    save_segment( segment_name, result_RPC_ID );
    if ( nCnt > count_RPC -1 ) nCnt ← 0;
  }
}
```

그림 5. 분할 파일 분배 알고리즘
Figure 5. Segment file distribution Algorithm

분할 된 부분 파일은 <그림 5> 분할 파일 분배 알고리즘에 의해 분배될 중계 서버를 할당 받는다.

<그림 5>에서 result_RPC_ID는 알고리즘 실행 당시 활성화되어 있는 중계서버들의 ID 목록을 저장한다. 중계서버 목록의 생성은 미디어의 저장을 우선 시하므로 해당 중계서버의 스토리지 용량이 높은 것부터, 네트워크 트래픽은 낮은 것부터, CPU 사용률은 작은 것부터, 메모리 사용률은 작은 것부터 정렬하여 목록을 작성한다. count_ts는 분할된 파일의 총 개수를 저장하고, nScale은 분할된 미디어파일 각각을 몇 개의 중계서버에 배분할 것인지를 나타내는 사용자 입력 변수이다.

실제 구현에서는 중계서버의 상태를 특정시간마다 Simple Network Management Protocol(SNMP)를 이용하여 중계서버 ON/OFF, CPU 사용률, 네트워크 트래픽, 메모리 상태, 스토리지 용량 등을 메인 데이터베이스에 저장하게 된다. 이렇게 데이터베이스에 저장되어져 있는 중계서버 상태를 이용하여 목록을 생성하고 해당 목록을 중계서버 할당량(nScale)만큼씩 일괄 분배하여 다시 데이터베이스에 입력하게 된다. 이렇게 할당된 분할 파일들은 중계서버에서 주기적으로 메인 데이터베이스에 접속해서 중계서버가 가지고 있는 목록과 비교하여 해당 분할 파일들을 가져오게 된다. 이를 실시간으로 처리하지 않는 이유는 사설전용망 자체가 미디어의 분배 및 재생이 주 업무가 아니기 때문이다. 즉, 기업 또는 기관의 주요 업무에 지장을 초래하지 않게 하기 위해 실시간으로 복제가 되는 클라우드 파일시스템과는 다르게 설계 및 구현이 되었다.

<그림 6>은 실제 미디어 분할이 이루어지고 난 후에 서버 측 스크립트에서 각각의 중계서버로 분할 파일을 할당하는 실제 코드의 일부를 나타내고 있다. 그리고 <그림 6>에 추가된 주석인 "//" 에는 <그림 5> 분할파일 분배 알고리즘 상의 의사코드를 같이 병기 하였으므로 해당 의사코드가 실제 코드로 구현되는 것을 볼 수 있다.

```
// result_RPC_ID ← get_RPC_ID_list();
$query  = "SELECT SRM.RELAY_MASTER_SN, SRM.MST_IP
FROM SB_RELAY_MASTER SRM, SB_RELAY_STATUS SRS
WHERE SRS.ALIVE = 1 AND SRS.RELAY_MASTER_SN =
SRM.RELAY_MASTER_SN ORDER BY SRS.HDD ASC, SRS.CPU
DESC, SRS.MEMORY DESC";
$res_RPC = $DB2->result($query);

// count_RPC ← count_RPC( result_RPC_ID );
$nRPC = count($res_RPC);

// nScale ← get_backup_scale();
$query = "SELECT COMM_VALUE FROM SB_COMM_CODE
WHERE COMM_CODE='vod_scale'";
$res_scale = $DB2->result($query);
if ($DB2->rowCount() > 0)
  $nScale = $res_scale[0]['COMM_VALUE'];
else  $nScale = 3;

$nCnt = 0;
for ( $i = 0; $i < $cnt_ts; $i++ )
  for ( $j = 0; $j < $nScale; $j++ ) {
    // segment_name ← filename + i + ".ts" ;
    // save_segment( segment_name, result_RPC_ID )
    $query = "SELECT CASE WHEN MAX(
MEDIA_SEGMENT_LIST_SN ) IS NULL THEN 1 ELSE MAX(
MEDIA_SEGMENT_LIST_SN ) + 1 END FROM
SB_MEDIA_SEGMENT_LIST";
    $res_pkid = $DB2->result($query);
    $max_pkid = $res_pkid[0][0];
    $format  = "INSERT INTO SB_MEDIA_SEGMENT_LIST
(MEDIA_SEGMENT_LIST_SN, SEG_NAME, MVP_FILE_SN,
RELAY_MASTER_SN, SEG_OWNER) VALUES (%d, '%s_%07d.ts',
%d, %d, 0)";
    $query = sprintf( $format, $max_pkid, $nName, $i, $pkid,
$res_RPC[$nCnt++]['RELAY_MASTER_SN'] );
    $DB2->execute($query);
    if ( $nCnt > $nRPC-1 )
      $nCnt = 0;
  }
```

그림 6. 분할 파일 분배 PHP 스크립트
Figure 6. Segment file distribution PHP script

3.2 미디어 분할 파일 목록 (Playlist) 생성 프로세스

위 3.1절의 미디어 배포 프로세스에 의해 미디어 분할 파일들이 배포되어진 후에 클라이언트들에 의해 미디어 재생 요청을 받게 되면 미디어 플레이를 위한 재생목록을 만들어 전달해 주어야 한다. <그림 5>에서와 같이 한 개의 분할 파일이 nScale 개수만큼의 중계서버에 분산되어 있으므로 분산된 중계서버 중에서 클라이언트 입장에서 가장 최적화된 중계서버를 선택해야 한다. 해당 중계서버의 선택은 다음과 같은 프로세스를 가진다.

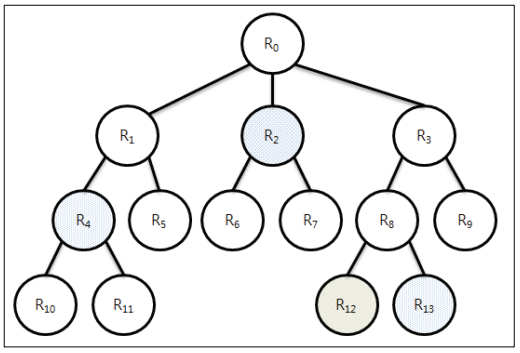

그림 7. 재생목록 생성 예시도
Figure 7. Playlist generation example diagram

우선 클라이언트로부터 요청되어진 미디어의 분할 파일 개수 n을 데이터베이스에서 읽어온다. 분할 파일 개수 n에 대하여 0부터 n-1까지 각각에 대하여(변수를 이후 s로 칭한다.) 다음의 과정을 반복하여 목록을 생성한다.

s에 해당하는 분할 파일에 대한 nScale개의 중계서버 목록을 데이터베이스에서 읽어온다. nScale이 3이고, <그림 7>에서와 같이 R_2, R_4, R_{13}에 분할 파일이 분산 저장되어 있다 가정한다면 각각의 선택된 중계서버(R_2, R_4, R_{13})를 클라이언트가 소속되어 있는 중계서버(R_{12})를 기준으로 최소공통조상 노드(이하 LCANi)를 검색하고 LCANi에서 R_{12}까지의 길이와 LCANi에서 각각의 선택된 중계서버까지의 길이를 합산한 결과 값을 계산한다.

또한 각 선택된 중계서버의 네트워크 트래픽 NT_2, NT_4, NT_{13}, 각 CPU 사용률 CU_2, CU_4, CU_{13}, 각 메모리 사용률 MU_2, MU_4, MU_{13}은 SNMP로 수집되어 데이터베이스에 저장된 자료를 읽어와 각각의 항목을 내림차순 정렬하여 가장 상위에 있는 중계서버를 선택한다.

```
$query = "SELECT RELAY_MASTER_SN FROM
SB_RELAY_MASTER WHERE MST_IP='$rpc_ip'";
$res_ip = $DB->result($query);
$client_rpc = $res_ip[0]['RELAY_MASTER_SN'];
$error_list = false;
header("Content-type: application/text");
header("Content-Disposition: attachment;
filename=index.m3u8");
$media =
file_get_contents("/home/servers/www/VOD/$nName/$nName.m
3u8");
for ($s = 0; $s < $nCnt; $s++) {
  $format = "SELECT RELAY_MASTER_SN FROM
SB_MEDIA_SEGMENT_LIST WHERE SEG_NAME='%s_%07d.ts'";
  $query = sprintf( $format, $nName, $s );
  $res_seg_rpc_list = $DB->result($query);
  $min_rpc_sn = CalcDistanceWithLCA( $client_rpc,
$res_seq_rpc_list );
  if ($min_rpc_sn == -1) { $error_list = true;  break; }
  $query = "SELECT MST_IP FROM SB_RELAY_MASTER
WHERE RELAY_MASTER_SN=$min_rpc_sn";
  $res_min_rpc_ip = $DB->result($query);
  $min_rpc_ip = $res_min_rpc_ip[0]['MST_IP'];
  $org_seg_name = sprintf( "%s_%07d.ts", $nName, $s );
  $media = str_replace( $org_seg_name,
"http://".$min_rpc_ip."/VOD/".$nName."/".$org_seg_name,
$media );
}
if ($error_list == false)  echo $media;
else {
  header("Content-type: video/mp4");
  header("Content-Disposition: attachment; $nName.mp4");
  $media = file_get_contents( "http://" . $rpc_ip . "/edu/" .
$nName .  ".mp4" );  echo $media; }
```

그림 8. 분할 파일 목록 생성 PHP 스크립트
Figure 8. Segment file play list generation PHP script

이 때 클라이언트가 요청한 시점에서 비활성화된 중계서버는 배제해야 한다. 이렇게 선택된 중계서버의 도메인 주소 또는 IP 주소를 HLS 파일내의 해당 분할 파일 명 앞에 HTTP 형식으로 변경한다. 이러한 과정을 해당 변수 s가 n-1에 도달할 때 까지 반복하여 HLS 목록을 완성한다.

분할 파일이 저장되어 있는 중계서버 목록이 완성되면 MPEG-DASH 파일 형태 또는 HLS 형태로 클라이언트에 전달하여 미디어를 재생하게 한다.

<그림 8>은 미디어 분할 파일 목록 생성을 위한 서버 측 실제 코드의 일부분을 나타내고 있다. <그림 8>내의 CalcDistanceWithLCA 함수가 LCA를 이용하여 클라이언트 중계서버와 분할파일이 저장되어 있는 중계서버와의 거리를 계산해내고, 계산되어진 거리 및 다른 속성 값들을 이용하여 클라이언트가 소속된 지점 또는 지사와 가장 가까운 거리의 중계서버를 반환하여 준다. 만일 분할 파일이 중계서버에 하나도 전달되어 있지 않거나 잘못 전달되어질 경우 “-1” 을 반환하여 메인 스트리밍서버 또는 메인 VOD서버에서 직접 미디어를 가져오게 되어 있다.

3.3 미디어 저장 효율 분석

효율 분석을 위해 실제 시험에서 중계서버는 66대, 동영상 미디어파일 1500개를 대상으로 하였다. 해당 동영상 파일은 비디오코덱은 H.264/AVC로 인코딩 하였고, 오디오 코덱은 AAC로 인코딩하였다. 각 미디어 파일의 해상도는 다양하며, 또한 각 코덱의 bit rate 역시 다양하다. 각 영상 파일의 분할 파일은 3초의 시간단위로 분할되었으나 I-frame 또는 Scene-cut 방식으로 분할하는 옵션이 적용 되었으므로 시간은 2초~6초 단위로 가변적이다. 이러한 미디어 파일을 중계서버에 <그림 5>의 nScale 변수를 3으로 하여 동일 분할 파일을 3개의 서버에 배분하였다.

3.1절의 미디어 배포 프로세스에 의해 실제 배포되었을 때 <표 1> 파일 크기에 따른 중계서버 저장 비율과 같이 나타난다. <표 1>에서와 같이 100개씩 나누어 측정한 결과 원래 동영상 파일의 크기 대비하여 5.03%에서 5.40%까지 중계서버 저장 비율이 나타났다.

표 1. 파일 크기에 따른 중계서버 저장 비율
Table 1. A relay storage rate per the file size.

파일 개수	파일 크기 (GB)	중계서버 평균용량 (GB)	파일 크기 대비 중계서버 저장 비율 (%)
100	7.04	0.35	5.03
200	18.26	0.96	5.27
300	35.65	1.9	5.34
400	49.22	2.64	5.36
500	60.44	3.25	5.37
600	77.25	4.16	5.38
700	87.15	4.69	5.38
800	98.06	5.28	5.39
900	106.62	5.75	5.39
1000	119.17	6.43	5.39
1100	126.82	6.84	5.39
1200	131.78	7.11	5.39
1300	138.4	7.47	5.40
1400	145.58	7.86	5.40
1500	158.24	8.51	5.38

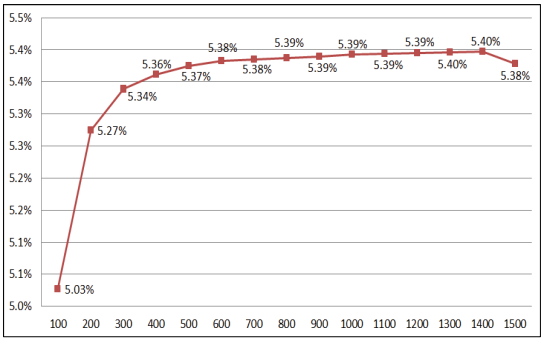

그림 9. 파일 개수별 중계서버 저장 비율
Figure 9 Relay storage file size ratio per number of files.

<그림 9>는 파일 개수별 중계서버 저장 비율의 추이를 나타내는 그래프이다. 그래프 상에서 600개 이상의 미디어 파일을 분할 배포할 경우에 미디어 파일의 상태에 따라 달라지겠지만 대체로 5.40%에 근접하여 비율이 안정적으로 유지하는 것을 볼 수 있다.

<그림 9>에서와 중계서버 저장 비율이 5.40%에 근접하므로 저장비율을 정수단위로 6%와 7%로 계산하면 <표 2>와 같이 메인 스토리지 크기별 중계서버 스토리지 용량을 산출할 수 있다.

<표 2>에서 보는 바와 같이 메인 스토리지 용량이 3072GB(3TB)인 경우 평균 215.04GB의 중계서버 스토리지 용량이 필요하고, 1024GB(10TB)인 716.80GB의 용량이 필요한 것으로 나타났다. 그러므로 중계서버 스토리지의 용량은 1TB가 넘지 않아도 메인 VOD서버의 용량이 10TB인 저장 용량을 본 논문에서 제시한 방법을 이용해 분산 저장할 수 있음을 알 수 있다.

표 2. 메인 스토리지 크기 별 중계서버 스토리지 크기
Table 2. A relay storage size per main storage size.

메인 스토리지	중계서버 스토리지 (6%)	중계서버 스토리지 (7%)
3072 GB	184.32 GB	215.04 GB
4096 GB	245.76 GB	286.72 GB
5120 GB	307.20 GB	358.40 GB
6144 GB	368.64 GB	430.08 GB
7168 GB	430.08 GB	501.76 GB
8192 GB	491.52 GB	573.44 GB
9216 GB	552.96 GB	645.12 GB
10240 GB	614.40 GB	716.80 GB

이러한 결과는 동영상 미디어들을 분산저장하지 않고 별도의 백업 저장 용량을 확보하거나 또는 중계서버의 용량을 메인 저장용량과 똑같이 가져갈 필요가 없음을 보여주고 있다.

4. 결 론

본론에서 미디어 파일을 MPEG-DASH 또는 HLS 방식으로 분할한 후 본 논문에서 제시한 미디어 배포 프로세스를 이용하여 각 중계서버에 분산 배포 하였다. 또한 분산된 분할 파일들을 이용해 메인 스트리밍서버에 부하 부담을 주지 않고 클라이언트 입장에서 최적 경로의 중계서버에 접근하여 미디어를 재생할 수 있게 하여주는 미디어 분할 파일 목록을 생성하여 안정적으로 미디어를 시청할 수 있다. 또한 이렇게 분할 파일을 이용할 경우 메인 스토리지 보다 상대적으로 작은 중계서버 스토리지 용량을 채택하여도 메인 스토리지의 용량을 대비할 수 있어 저비용, 고효율의 시스템을 구축할 수 있다.

References

[1] S. Ghemawat, H. Gobioff, and S. T. Leung, *The google file system*. In ACM SIGOPS Operating Systems Review. Vol. 37, No. 5, pp. 29-43, ACM, October, 2003.

[2] *HDFS*, http://hadoop.apache.org/core/docs

[3] *Amazon S3, loud computing storage for files, images, videos*. http://aws.amazon.com/s3/

[4] R. B. Ross, and R. Thakur. *PVFS: A parallel file system for Linux clusters*. In in Proceedings of the 4th Annual Linux Showcase and Conference. pp. 391-430, 2000.

[5] *Kosmix releases Google GFS workalike 'KFS' as open source*. http://www.skrenta.com/2007/09/kosmix_releases_google_gfs_wor.html

[6] M. H. Cha, S. M. Lee, K. S. Jin. Y. S. Min, and Y. K. Kim. *Design and implementation of a consistent update method for multiple file replicas in a distributed file syste*

m. In Advanced Communication Technology, 2008. ICACT 2008. 10th International Conference, Vol. 3, pp. 2063-2066, IEEE. February, 2008.

[7] H. N. Gabow, and R. E. Tarjan, *A linear-time algorithm for a special case of disjoint set union*. In Proceedings of the fifteenth annual ACM symposium on Theory of computing, pp. 246-251. ACM. December, 1983.

[8] *Tarjan's off-line lowest common ancestors algorithm*, http://en.wikipedia.org/wiki/Tarjan's_off-line_lowest_common_ancestors_algorithm

[9] *Range minimum query and lowest common ancestor*. http://community.topcoder.com/tc?module=Static&d1=tutorials&d2=lowestCommonAncestor

[10] T. Stockhammer, *Dynamic adaptive streaming over HTTP--: standards and design principles*. In Proceedings of the second annual ACM conference on Multimedia systems. pp. 133-144. ACM. February, 2011.

[11] K. Honkanen, *HTTP Live streaming*. 2011.

사설전용망내의 fixed-node 서버를 이용한 클라우드 파일시스템 기반 VOD 서비스

정명기, 김재웅

공주대학교 컴퓨터공학과

요 약

사설전용망을 이용하여 사내에 VOD 서비스하는 기관 또는 기업들에서는 동시에 많은 사용자들이 접속하기 위한 대책을 마련해야 한다. 그 방법으로는 중계서버에 특정 미디어를 전체 전송하고 해당 위치의 사용자들은 해당 위치의 중계서버에서 서비스를 받는 방법을 채택하고 있다. 그러나 이러한 방식은 미디어의 개수 및 크기가 많아지고 커지면 중계서버의 스토리지도 따라서 같이 증설해야 하는 문제점이 있다. 이러한 문제점을 해결하기 방법으로 클라우드 파일시스템의 개념을 응용하여 미디어를 분할하여 일부만을 지정된 여러 대의 중계서버에 배포하는 방식을 사용한다. 이렇게 하면 VOD 서버 저장 공간의 6~7% 정도의 저장 공간만을 가진 중계서버를 운용할 수 있다. 그리고 사용자가 소속되어 있는 해당 지점 또는 지사를 기준으로 하여 분할 및 저장되어 있는 여러 중계서버 중 최적의 중계서버를 최소공통조상 알고리즘 및 여러 속성 값을 이용하여 선택한다. 위의 과정을 모든 분할 파일에 대해 반복하여 목록을 생성하고 클라이언트에게 제공하여 VOD를 시청하게 한다. 이런 방식은 클라이언트가 VOD 서버에 집중하지 않고 여러 중계서버를 이용하게 됨으로 VOD서버의 접속부하를 줄일 수 있다. 이 방식을 이용하면 저비용, 고효율 및 안정성을 갖춘 VOD 서비스를 구현할 수 있다.

Myeong-Ki Jung received the bachelor's degree in the Department of Electric Engineering Education from the Chungnam National University in 1994. He received the M.S. degree in the Department of Computer Multimedia Engineering from Kongju National University in 2004. His current research Interests include software engineering, web engineering, IPTV. He is a regular member of the KKITS.

E-mail address: bright@kongju.ac.kr

Jae-Woong Kim received the bachelor's degree and the MS degree in the Department of Computer Engineering from the Jungang University in 1983 and 1988, respectively. He received the Ph.D. degree in the Department of Computer Engineering from Daejun University in 2000. He has been a professor in the Department of Computer Engineering at Kongju National University since 1992. His current research interests include software engineering. He is a life member of the KKITS.

E-mail address: yjkim@kongju.ac.kr