트랜스코딩 작업의 분배를 활용한 저전력 트랜스코딩 서버 설계 및 구현

Design and Implementation of Low-Power Transcoding Servers Based on Transcoding Task Distribution

이다영, 송민석[1)]
Dayoung Lee, Minseok Song

(22212) 인천광역시 미추홀구 인하로 100 인하대학교 컴퓨터공학과
cecci08@naver.com, mssong@inha.ac.kr

요 약

동적 적응 스트리밍 서버는 일시에 많은 양의 트랜스코딩 연산을 처리하기 때문에 높은 프로세서 전력을 소모한다. 많은 연산량을 위하여 다중 프로세서 구조가 필요하고, 이에 대한 효과적인 트랜스코딩 태스크 분배가 필요하다. 본 논문에서는 2 티어 (프론트엔드 노드 (frontend node)와 백엔드 노드 (backend node)) 트랜스코딩 서버의 전력 상한을 보장하고 스트리밍 되는 비디오의 인기도 및 품질을 고려한 트랜스코딩 서버의 설계 및 구현 방법을 제안한다. 이를 위하여 1) 각 백엔드 노드에 트랜스코딩 태스크 분배, 2) 백엔드 노드에서의 태스크 스케줄링, 3) 프론트엔드와 백엔드 노드 통신 기법들을 구현하고, 테스트베드를 구축하였다. 실제 테스트베드에서의 예상 소모 전력과 실제 소모 전력을 비교하는 실험을 진행함으로써 본 시스템의 효용성을 확인했다. 또한 본 시스템이 각 노드의 부하를 감소시킴으로써 트랜스코딩에 사용되는 전력 및 시간 최적화가 가능함을 보였다.

Abstract

A dynamic adaptive streaming server consumes high processor power because it handles a large amount of transcoding operations at a time. For this purpose, multi-processor architecture is mandatory for which effective transcoding task distribution strategies are essential. In this paper, we present the design and implementation details of the transcoding workload distribution schemes at a 2-tier (frontend node and backend node) transcoding server. For this, we implemented four schemes: 1) allocation of transcoding tasks to appropriate back-end nodes, 2) task scheduling in the back-end node and 3) the communication between front-end and back-end nodes. Experiments were conducted to compare the estimated and the actual power consumption in a real testbed to

※ 이 논문은 2018년도 과학기술정보통신부의 재원으로 한국연구재단-차세대정보·컴퓨팅기술개발사업의 지원을 받아 수행된 연구임 (NRF-2017M3C4A7080248). 이 논문은 2018년도 정부(교육부)의 재원으로 한국연구재단의 지원을 받아 수행된 기초연구사업임 (NRF-2018R1D1A1B07050614)

1) 교신저자

www.kci.go.kr

논문투고일자 : 2019. 06. 27 심사일자 : 2019. 07. 11 게재확정일자 : 2019. 08. 13

verify the efficacy of the system. It also proved that the system can reduce the load on each node to optimize the power and time used for transcoding.

키워드: DASH 서버, 프로세서 전력 관리, 트랜스코딩 작업 분배, 로드 밸런싱
Keyword: DASH server, power management, transcoding task distribution, Load balancing

1. 서론

최근 인터넷 및 스마트 기기 기술의 발전을 통해 Youtube, Netflix, Twitch 등에서 제공하는 비디오 스트리밍 서비스가 대중화되었다.

이러한 스트리밍 업체들은 일반적으로 동적 적응 스트리밍(DASH, Dynamic Adaptive Streaming over HTTP)을 사용한다[1,4]. 동적 적응 스트리밍은 비디오를 미리 세그먼트로 분할한 후, 여러 개의 비트율 버전을 가지고 있다가 사용자의 네트워크 환경에 맞추어 가장 적합한 비트율 버전을 제공하므로 많은 횟수의 비디오 트랜스코딩이 요구된다. 이때 고도의 연산이 집약되어 있는 트랜스코딩의 수행에는 많은 전력이 소모되므로, 서버는 상당한 전력 비용을 필요로 한다. 비디오 스트리밍 서비스의 인기가 점점 증가함에 따라 처리해야할 비디오의 양이 증가하여, 전력비용의 최적화는 서비스 제공 업체에게 있어 가장 중요한 문제 중 하나이다.

최근 여러 비디오의 요인에 따라 비트율을 선택하여 트랜스코딩하는 연구가 진행되었다. 그러나 실제 서버환경에 대한 고려가 없어 테스트베드 구축을 통한 실제 전력 측정이 이루어지지 않았으며, 트랜스코딩 작업 분배를 위한 로드 밸런싱이 필요했으나 이에 대한 고려는 이루어지지 않은 채 한 번에 프로세서에게 모든 작업을 할당하는 문제가 있었다.

이를 위해서는 사용자 인기도 및 각 세그먼트의 품질 비교를 바탕으로 제외할 수 있다. 또한 트랜스코딩에 사용될 소스의 비트율에 따라 트랜스코딩에 사용되는 전력이 차이가 있으므로 트랜스코딩 작업에 대한 스케쥴링이 필요하다. 그러나 기존의 연구는 비트율 버전의 선택만 고려의 대상이었으며, 많은 경우 스케쥴링 없이 원본에서 바로 트랜스코딩되었기 때문에 전력의 낭비가 있었다. 즉 실제로 진행하는 트랜스코딩 시스템 및 스케쥴링에 대한 고려가 미흡하다.

그러므로 실제 환경에서의 전력 최적화를 고려한 트랜스코딩을 위해 서버 시스템에게 요구되는 사항은 다음과 같다. ① 트랜스코딩할 버전의 선택 및 노드 별 태스크 분배 ② 각 비트율 버전을 트랜스코딩하는 작업에 대한 스케쥴링 ③ 클라이언트의 비트율 요청 및 응답을 위한 통신이다.

본 논문에서는 이를 위하여 트랜스코딩 작업의 분배를 이용한 저전력 트랜스코딩 서버 시스템을 제안한다.

2. 관련 연구

스트리밍 서버 및 비디오 트랜스코딩에 사용되는 자원을 줄이기 위해 다양한 연구가 진행되었다.

스트리밍 서버의 자원 관리를 위해, Madumitha와 그 외 연구진은 클라우드 서버에서 비디오 콘텐츠를 다운로드하는 동안 기존 지연 시간을 줄임과 최적화된 할당을 통해 각각 다른 디바이스에게 비디오를 제공할 수 있는 스케쥴링 기법을 제안했다[7]. Aparicio-Pardo와 그 외 연구진은 라이브 비디오 스트리밍의 시간에 따라 변화하는 요구 사항을 만족시키기 위해 클라우드 컴퓨팅 인프라를 제공하고 QoE의 평균값을 극대화하는 알고리즘 및 데이터 집합을 제안했다[5]. Kim과 그 외 연구진은 8K의 해상도를 가진 360° VR (Virtual Reality) 서비스가 소비자들의 인터넷 대역폭의 한계로 인해 많은 클라이언트에게 제공이 어려우므로 이를 해결하기 위해 분산 컴퓨팅 기반의 트랜스코딩 플랫폼

www.kci.go.kr

을 제안하였다[8]. 조대현 및 그 외 연구진은 비디오에 대한 인기도가 동적으로 변하는 클러스터 기반의 서버 환경에서 클러스터 비디오 서버의 성능을 최대화하는 버퍼 공유를 통해 요청 분배 방법을 제안했다[13]. 오주병과 그 외 연구진은 스마트 TV 및 N스크린을 위하여 기존의 리소스들을 재활용 하여 사용자에게 비디오 서비스를 제공하는 기법을 제안하였다[14].

비디오 트랜스코딩과 관련된 자원을 관리하기 위하여, Li와 그 외 연구진은 인터넷에서 원본 비디오를 다운로드 후 필요에 따라 트랜스코딩하고 이를 뷰어에 전달함으로써 트랜스코딩을 위한 복잡한 계산을 클라우드 기반의 서비스에 의해 빠르게 처리하는 트랜스코딩 시스템을 개발했다[6]. Zhao과 그 외 연구진은 여러 환경에 대비하기 위하여 비디오를 세그먼트로 분할하고, 비디오의 인기도에 따라 다양한 트랜스코딩 버전을 제공함으로써 자원 및 서버의 용량을 줄이는 VoD 시스템을 제안했다[3]. Barlas는 이기종 플랫폼에서 CBR 및 VBR 미디어 모두를 최적화 하기 위한 클러스터 기반의 대규모 병렬 트랜스코딩을 위한 분석적 접근 방법을 제안하였다[2]. Wei와 그 외 연구진은 트랜스 코딩 매개 변수의 프로파일링을 통해 다양한 환경을 지원함으로써 CPU 코어 수 및 트랜스 코딩 시간 등의 자원을 최적하는 클라우드 기반의 온라인 비디오 트랜스 코딩(COVT) 시스템을 제안했다[15]. Jin과 그 외 연구진은 이들은 에지 서버의 캐싱, 트랜스 코딩 및 대역폭 비용의 최적화를 토대로 한 주문형 비디오 스트리밍 서비스를 제안했다[16]. 또한 백치선과 그 외 연구진은 서버의 부하량을 예측하여, QoS를 최적화 하는 트랜스코딩 서버를 위한 관리 기법을 제안했다[12].

트랜스코딩에서의 프로세서의 전력 효율을 집중적으로 고려하기 위한 연구도 진행되었다. Kuang과 그 외 연구진은 프로세서 오퍼레이팅 레벨을 조정함으로써 트랜스코딩 전력 효율을 조절하는 멀티코어 서버 시스템을 제안했다[9]. Song과 그 외 연구진은 각 프로세서의 주파수와 작업의 데드라인을 만족하는 작업 할당을 통해 전력 소모를 최소화하기 위한 시스템을 제안하였다. 해당 시스템은 프론트 엔드 노드가 각 이기종 CPU를 담당하는 각 백엔드 노드를 제어하는 형태이다[10].

그러나 위의 연구는 트랜스코딩할 버전의 선택을 통한 전력 최적화를 고려하지는 않았다. Lee와 그 외 연구진은 DASH 환경에서의 사용자 QoE를 극대화 시키면서 트랜스코딩에 사용되는 전력 소모를 최소화하기 위한 휴리스틱 기반의 알고리즘을 제안하였다[11]. 하지만 실제 트랜스코딩 서버 환경이 고려되지 않아 여러 대의 프로세서가 사용되는 환경에 대한 고려가 없었으며, 실험 역시 시뮬레이션을 돌렸을 뿐 실제 서버 환경에서 사용되지는 않았다.

3. 비트율 버전 선택 방법

본 논문의 비트율 버전 선택 알고리즘은 Lee와 그 외 연구진이 기존에 제안한 알고리즘 [11]을 기반으로 하여 새로운 구현 방식을 제안한다. 구현에 사용한 알고리즘의 원리를 간단히 정리하면 아래와 같다 [11].

N^{vid}는 전제 비디오의 개수를 의미하며, N_i^{seg}는 비디오 i의 세그먼트 개수를 의미한다. N^{ver}은 전체 비트율 버전의 개수이며 원본 버전을 포함한 숫자이다. 가장 낮은 비트율 버전은 1이고, 원본 버전은 N^{ver}이다. 비디오 $i\,(i=1,\ldots,N^{vid})$의 세그먼트 $j\,(j=1,\ldots,N_i^{seg})$의 버전 $k\,(k=1,\ldots,N^{ver})$의 사용자 인기도를 $p_{i,j,k}$라고 하자. 또한 여기서 비디오 i의 세그먼트 j의 버전 k의 QoE (quality-of-experience) 값을 $Q_{i,j,k}$라고 하자. $Q_{i,j,k}$는 SSIM 도구와 같은 QoE 측정 도구로 얻은 값을 변환하여 얻으며 <표 1>에서 확인 가능하다.

www.kci.go.kr

〈표 1〉 SSIM 도구로 측정값의 QoE 값 변환

SSIMi,j	Qi,j	등급
SSIMi,j ≥ 0.99	5	excellent
0.95 ≤ SSIMi,j 〈0.99	25SSIMi,j - 19.75	good
0.88 ≤ SSIMi,j 〈0.95	14.29SSIMi,j - 9.57	fair
0.5 ≤ SSIMi,j 〈0.88	3.03SSIMi,j + 0.48	poor
SSIMi,j 〈0.5	1	bad

비트율 버전 선택 알고리즘은 서버 관리자가 설정한 트랜스코딩 시간 한도 $C^{\lim}$내에서 전체 스트리밍된 비디오의 사용자 인지 품질을 최대로 하는 거을 목표로 한다. 그러기 위해서는 비교적 높은 사용자 인기도와 QoE를 가진 버전을 선택해야 하며, 이를 위하여 영화 i 의 j 세그먼트의 k 버전을트랜스코딩할 때의 이득 $G_{i,j,k}$를 정의하였으며 식 1과 같다.

$$G_{i,j,k} = p_{i,j,k} Q_{i,j,k} \tag{1}$$

$X_{i,j,k}$는 비디오 i의 세그먼트 j의 버전 k가 트랜스코딩되는지 여부를 나타낸다.

$$X_{i,j,k} \in 0,1 \tag{2}$$

$C_{i,j,k}$는 비디오 i의 세그먼트 j의 버전 k의 트랜스코딩에 사용되는 시간을 의미할 때, 비트율 버전 선택 문제를 식으로 나타내면 아래와 같다.

$$\begin{aligned} &\text{Maximize} \sum_{i=1}^{N^{vid}} \sum_{j=1}^{N_i^{seg}} \sum_{k=1}^{N^{ver}} X_{i,j,k} G_{i,j,k} \\ &\text{subject to} \sum_{i=1}^{N^{vid}} \sum_{j=1}^{N_i^{seg}} \sum_{k=1}^{N^{ver}} X_{i,j,k} C_{i,j,k} \le C^{\lim} \end{aligned} \tag{3}$$

P^{trans}가 트랜스코딩에 사용되는 전력일 때, 소모되는 에너지, $E^{\lim}$은 $E^{\lim} = P^{trans} \times C^{\lim}$으로 계산된다. 그러므로 해당 알고리즘은 $C^{\lim}$를 결정함으로써 에너지 한도 $E^{\lim}$를 결정할 수 있다 [11].

상기 문제를 해결하는 휴리스틱 알고리즘이 [11]에서 제안되었으며, 본 구현에서는 해당 알고리즘을 활용하여 다중 프로세서 환경에서 실제 설계 구현하였다.

4. 설계

4.1 구조

본 서버 시스템은 최대한 낮은 전력에서, 프로세서의 부하를 줄이는 것을 목적으로 한다. 이를 구현하기 위해 시스템을 멀티 노드로 구성하였으며 각 노드는 PC이다. 즉 본 서버 시스템은 각자의 프로세서를 가진 여러 대의 PC로 구성되었다. 각 노드는 전체적인 작업 관리 및 클라이언트의 통신을 담당하는 한 개의 프론트엔드 노드와, 실제 트랜스코딩을 진행하는 N^{back}개의 백엔드 노드로 구성된다. 또한 모든 노드들은 NFS(Network File System)를 통해 연결되어 있으며, 지속적으로 계속 동기화되어 모든 노드는 항상 같은 데이터를 가지고 있다.

N^{job}은 트랜스코딩 시 특정 비디오 세그먼트를 선택된 비트율 버전으로 트랜스코딩하기 위한 작업의 개수이다. 프론트엔드 노드는 i번째 작업 $\tau_i (i = 1, \ldots, N^{job})$을 각 백엔드 노드에 할당한다.

본 시스템의 일반적인 동작 순서는 아래와 같다.

① 비트율 버전 선택 : 비디오가 업로드 되면, 프론트엔드 노드는 각 세그먼트마다 사용될 비트율 버전을 선택한다.

② 트랜스코딩 작업 분배 : 프론트엔드 노드는 ①에서 선택된 비트율 버전을 트랜스코딩하는 작업 τ_i를 생성하고, 각 작업을 각 백엔드 노드에 분배한 후 할당한다.

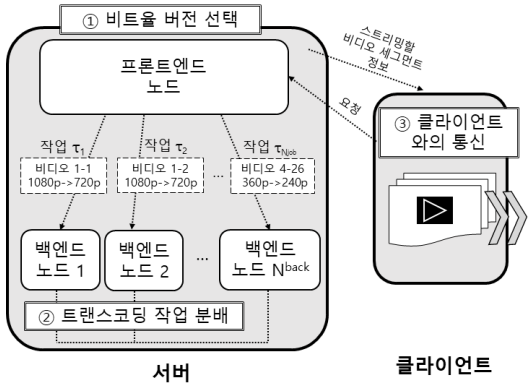
③ 클라이언트와의 통신 : 프론트엔드 노드는 통신 스레드를 통해 클라이언트에게 스트리밍 요청을 받는다. 프론트엔드 노드는 클라이언트에게 스트리밍할 비트율 버전이 무엇인지 전송한다. 이 때 해당 버전이 없다면 그 하위 버전을 전송한다. 만약 전송 가능한 버전이 없

www.kci.go.kr

다면 마지막 버전을 트랜스코딩 한 후 전송한다. 클라이언트는 이에 맞추어 사용자에게 스트리밍한다.

반드시 ①, ②번이 끝난 이후에만 ③번이 가능한 것은 아니다. 만약 백엔드 노드가 비디오 트랜스코딩을 진행 중일 때, 클라이언트의 요청에 의해 추가적인 트랜스코딩이 필요할 시, 프론트엔드 노드는 백엔드에 할당된 트랜스코딩의 작업들의 예상소요시간이 가장 적은 백엔드 노드를 선정하여 트랜스코딩 작업 τ_i을 할당함으로써 추가 트랜스코딩을 시행한다. 이 때 추가적으로 트랜스코딩된 세그먼트는 바로 스트리밍 되어야하기 때문에 다른 작업보다 우선된다.

(그림 1)은 본 시스템의 간략한 구조 및 전반적 진행을 보여준다.



(그림 1) 본 시스템의 간략한 구조 및 전반적 진행

4.2 프론트엔드 노드

프론트엔드 노드는 백엔드 노드에게 소켓 통신을 통한 명령을 보내는 방식으로 각 백엔드 노드의 움직임을 제어한다. 예를 들어 각 프론트엔드 노드는 각 백엔드 노드에게 트랜스코딩 태스크를 할당하기 위해 각 백엔드 노드가 읽을 공유 파일을 생성하여 할당된 작업의 리스트를 저장한다. 그리고 소켓으로 명령을 보냄으로써 각 백엔드 노드가 해당 공유파일을 읽어오게 한다.

프론트엔드 노드는 모든 백엔드 노드의 관리 및 제어, 비트율 버전 선택 및 분배, 클라이언트의 요청 관리를 진행한다. 이를 구현하기 위해 프론트엔드 노드는 총 세 가지 종류의 스레드를 가지고 있으며 이는 아래와 같다.

① 트랜스코딩 비트율 선택 및 노드 별 작업 분배 스레드 : 해당 스레드는 트랜스코딩할 비트율 버전을 선택하고 트랜스코딩 작업을 각 노드에 분배하는 로드 밸런싱 스레드이다.

② 각 백엔드 노드와의 통신 스레드 : 해당 스레드는 백엔드 노드에 할당된 버전을 트랜스코딩하기위한 작업을 생성하고 각 백엔드에 할당한다.

③ 클라이언트와의 통신 스레드 : 클라이언트의 요청에 대응하여 클라이언트가 스트리밍할 버전을 알려주는 스레드이다. 클라이언트가 요청한 버전이 없다면, 그 아래의 버전을 알려준다.

이러한 트랜스코딩 태스크의 로드 밸런싱을 위한 알고리즘은 서버 관리자에 의해 여러 가지가 선택될 수 있다. 우리는 3장에서 설명한 방법을 사용하였다.

4.3 백엔드 노드

서버를 구성하는 모든 백엔드 노드는 프론트노드와 소켓과 공유 파일 갱신을 통해 통신하게 된다. 모든 백엔드 노드는 프론트엔드 노드에게 트랜스코딩 작업을 할당받고 작업의 순서를 스케쥴링 후 수행하는 역할을 가지고 있다. 이를 위하여 작업할 작업을 담아두는 큐를 가지고 있으며, 할당된 작업을 스케쥴링 한 후 큐에 담는다.

할당된 작업은 프론트엔드 노드에서 해당 노드의 번호로 생성한 공유 파일을 읽어옴으로써 확인할 수 있으며, 추가 트랜스코딩이 발생 시 프론트 엔드에 의해 각 백엔드 노드는 해당 공유파일을 갱신하여 현재 대기 중인 작업 현황을 알린다.

이를 구현하기 위해 백엔드 노드 역시 두 가지의 스레드를 가지고 있으며 이는 아래와 같다.

① 트랜스코딩 스레드 : 실제로 트랜스코딩을 담

www.kci.go.kr

당하는 스레드이다. 프론트엔드에 의해 트랜스코딩할 작업이 할당되지 않으면 일시 정지 상태로 있으나, 프론트엔드 노드에 의해 트랜스코딩할 작업이 할당되면 깨어난다.

② 프론트엔드 노드와의 통신 스레드 : 프론트엔드에 의해 트랜스코딩 명령을 받거나, 공유 파일 갱신 명령을 받는다. 트랜스코딩 명령을 받을 경우 트랜스코딩 태스크를 깨운다.

5. 구현

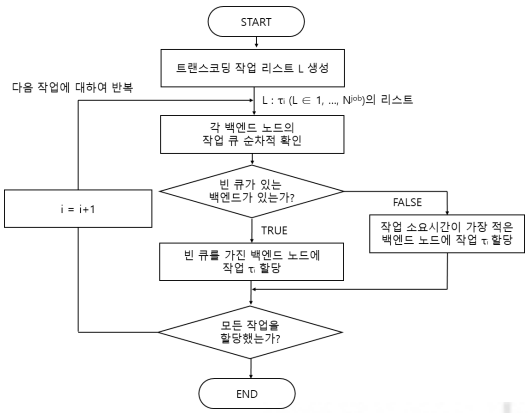
5.1 프론트노드의 백엔드 작업 할당

본 시스템은 프론트엔드 노드로부터 여러 개의 백엔드 노드에 트랜스코딩 작업을 할당하는 시스템이므로, 할당 알고리즘이 중요하다. 프론트엔드 노드는 각 백엔드 노드에 작업을 할당하기 위해, 각 노드별 트랜스코딩 소요시간을 함께 계산한다. 프론트엔드 노드가 백엔드 노드에 작업을 할당하는 방법은 다음과 같다.

① 해당 백엔드 노드의 작업 큐를 확인하고, 큐가 비어있는 백엔드에 할당한다.

② 만약 비어있는 백엔드 노드가 없다면, 각 백엔드 노드의 트랜스코딩 소요시간을 확인하고 이가 가장 적은 백엔드 노드에 작업을 할당한다.

(그림 2)는 프론트엔드 노드의 트랜스코딩 작업 할당 순서도이다.



(그림 2) 프론트엔드 노드의 작업 할당 순서도

생성된 작업을 각 백엔드 노드에 할당한 후 프론트엔드 노드는 해당 노드의 번호로 할당한 작업의 정보들을 파일 형태로 저장한다. 저장하는 정보는 <표 2>와 같다.

〈표 2〉 각 백엔드 노드의 작업 할당을 위해 파일로 저장되는 작업의 상세 정보

τ_i 의 정보	설명
$N_{\tau_i}^{vid}$	작업 τ_i 에 할당된 비디오의 번호
$N_{\tau_i, j}^{seg}$	작업 τ_i 에 할당된 비디오의 세그먼트의 번호
$V_{\tau_i}^{src}$	작업 τ_i 의 트랜스코딩 소스로 사용하는 세그먼트의 비트율 버전의 번호
$V_{\tau_i}^{tgt}$	작업 τ_i 의 트랜스코딩 결과로 얻을 세그먼트의 비트율 버전의 번호

5.2 백엔드 노드의 작업 스케쥴링

프론트엔드 노드로부터 작업을 할당받은 백엔드 노드는 어떤 작업을 먼저 수행할 것인지에 대한 스케쥴링을 실시한다. 작업 τ_i 의 대상인 비디오의 번호는 $N_{\tau_i}^{vid}$ 이고, $N_{\tau_i, j}^{seg}$는 해당 비디오의 몇 번째 세그먼트인가를 의미한다. 트랜스코딩 소스로 삼는 비트율 버전은 $V_{\tau_i}^{src}$이고, 트랜스코딩의 결과, 즉 타겟으로 설정한 비트율 버전은 $V_{\tau_i}^{tgt}$이다. 이 때 $V_{\tau_i}^{src}$과 $V_{\tau_i}^{tgt}$는 반드시 1부터 N^{ver}사이의 값이다. 또한 작업 τ_i의 결과물로 나올 세그먼트의 예상 사용자 접근률을 사용자 인기도 p_{τ_i}라고 정의한다.

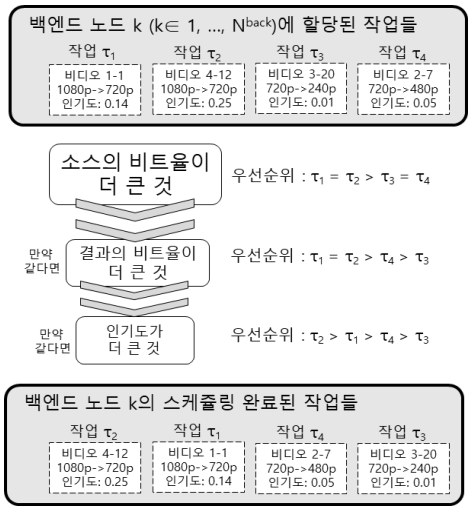
예를 들어 본 시스템이 N^{ver}=5 인 환경에서 1080p가 5번, 360p가 3번으로 정의되었다고 가정하자. 작업 τ_i가 1번 비디오의 13번 세그먼트의 1080p 버전을 360p로 트랜스코딩 할 경우, $N_{\tau_i}^{vid}$ =1, $N_{\tau_i, j}^{seg}$=13, $V_{\tau_i}^{src}$=5, $V_{\tau_i}^{tgt}$=3이다. 또한 1번 비디오의 13번 세그먼트의 360p의 예상 사용자 접근률이 5%라면 p_{τ_i}=0.05이다.

본 시스템은 $V_{\tau_i}^{src}$가 높은 작업에게 더 높은 우선순위를 준다. 만약 $V_{\tau_{i-1}}^{src}$와 $V_{\tau_i}^{src}$가 같은 경우, $V_{\tau_i}^{tgt}$가 높은 작업이 우선순위가 더 높다. 만약

www.kci.go.kr

$V_{\tau_{i-1}}^{tgt}$ 와 $V_{\tau_i}^{tgt}$ 가 같은 경우, 예상 사용자 인기도 p_{τ_i} 가 높은 작업이 우선순위를 높게 주는 형태로 스케쥴링 한다. 우선순위가 높은 작업이 먼저 트랜스코딩된다.

(그림 3)은 백엔드 노드의 스케쥴링 예시이며, 작업의 우선순위를 어떻게 결정하는지 확인할 수 있다.



(그림 3) 백엔드 노드의 스케쥴링 예시

본 스케쥴링의 이유는, 트랜스코딩 소스로 사용되는 버전의 비트율과 결과로 얻고자 하는 버전의 비트율의 차이가 적을수록 전력이 적게 소모되기 때문이다. 트랜스코딩의 결과물을 더 낮은 비트율 버전의 트랜스코딩 소스로 사용하여 전체 전력을 줄일 수 있다.

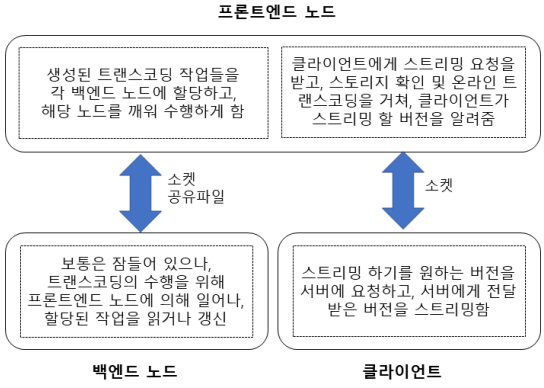
만약 백엔드 노드가 할당된 작업들을 아직 처리하고 있는 상황에 프론트엔드 노드로부터 추가 작업이 할당될 경우, 백엔드 노드는 남은 작업들과 추가된 작업을 합쳐 새롭게 스케쥴링 한다.

5.3 통신

프론트엔드 노드는 백엔드 노드의 관리 및 제어 및 클라이언트와의 통신을 통하여 트랜스코딩을 진행한다. 프론트엔드 노드는 백엔드와 클라이언트와의 소켓을 기반으로 통신하나 세부적인 방법이 다르다.

① 프론트엔드-백엔드 통신 : 백엔드 노드의 트랜스코딩 스레드는 기본적으로 트랜스코딩할 작업이 할당되지 않으면 일시 정지 상태로 있다. 프론트엔드 노드는 트랜스코딩 작업 할당을 위한 공유파일을 생성하는 동시에 동시에 소켓으로 명령을 보낸다. 명령을 받은 백엔드 노드는 트랜스코딩 태스크를 깨우며, 깨어난 트랜스코딩 스레드는 공유파일을 읽는다. 프론트 노드는 트랜스코딩 태스크에게 공유 파일을 만들게 하여 작업의 상황을 공유할 수 있다. 즉 프론트엔드-백엔드와의 통신은 소켓과 공유파일을 함께 사용하는 방식이다.

② 서버(프론트엔드)-클라이언트의 통신 : 서버는 클라이언트로부터 소켓통신을 통해 스트리밍 요청을 받는다. 클라이언트는 서버에 스트리밍 되기를 원하는 비디오 번호, 세그먼트 번호, 비트율 버전 세 가지의 정보를 서버에 전송한다. 이 정보를 받은 프론트엔드 노드는 클라이언트의 요청에 따라 추가 트랜스코딩을 진행할 수 있으며, 5.1에 설명된 할당 방식을 통해 작업 큐가 비어있거나, 트랜스코딩 소요시간이 가장 적은 백엔드 노드에 할당하고, ①의 내용대로 백엔드와 통신하여 추가트랜스코딩을 진행한다.



(그림 4) 프론트엔드를 중심으로 진행되는 통신

www.kci.go.kr

(그림 4)는 본 시스템에서 프론트엔드를 중심으로 일어나는 통신에 대하여 간략하게 표현하였다.

6. 실험

6.1 실험 환경

본 논문에서 제시하는 시스템의 노드는 Intel i7-4790으로 구성되었다. 영상은 ① 당갈 ② 라라랜드 ③ 바후발리 ④ 아일라 ⑤ 너의 이름은 총 5가지를 사용하였다. 각각의 비디오를 10초의 세그먼트로 나눈 후, 복제하여 영상 당 100개의 세그먼트를 사용하고, 총 500개의 세그먼트를 사용하였다.

실험에 사용된 각 비디오의 트랜스코딩 시간은 아래의 <표 3>과 같으며 본 시스템을 이용하여 트랜스코딩했을 경우의 시간이다.

〈표 3〉 각 비디오의 트랜스코딩 시간

소스 해상도 (kbps)	결과 해상도 (kbps)	비디오				
		①	②	③	④	⑤
1080p (10000)	720p (4000)	3.52	3.07	4.52	4.00	4.21
	480p (2000)	2.05	1.85	2.64	2.60	2.33
	360p (1000)	1.38	1.16	1.50	1.65	1.59
	240p (500)	1.05	0.95	1.33	1.23	1.19
720p (4000)	480p (2000)	1.70	1.93	2.25	1.99	2.09
	360p (1000)	1.24	1.12	1.47	1.36	1.44
	240p (500)	0.85	0.98	1.19	0.94	1.27
480p (2000)	360p (1000)	1.07	1.04	1.41	1.20	1.33
	240p (500)	0.69	0.68	0.69	0.68	0.70
360p (1000)	240p (500)	0.69	0.61	0.68	0.65	0.64

비디오 세그먼트의 접근률은 zipf 분포를 따르며, θ = 0.271이다. 스트리밍 시간은 총 24시간이고, 1초당 2건의 요청이 올 경우에 대하여 실험을 수행하였다.

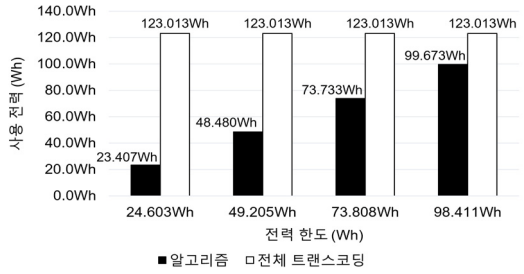
본 시스템은 FFmpeg 4.1.1를 이용하여 각 비트율 버전의 트랜스코딩을 실시하였다. 각 실험은 3번씩 진행하여 평균값을 구했다.

6.2 비트율 버전 선택의 신뢰성 확인

해당 실험은 본 시스템이 내장된 비트율 버전 선택을 확인하기 위해 진행되었다. 비트율 버전 선택이 의도대로 진행되는지 확인하기 위해, 백엔드 노드가 1개일 때 전체 버전을 트랜스코딩 할 때의 20%, 40%, 60%, 80%의 전력을 사용할 경우에 대하여 예상한 값이 나오는 지를 전력 측정계를 통해서 확인하였다.

〈표 4〉 전력 한도 설정에 따른 스트리밍된 QoE 값

전체 버전의 트랜스코딩 기법	알고리즘에 사용한 에너지 한도			
	24.603Wh	49.205Wh	73.808Wh	98.411Wh
4.698	4.558	4.664	4.69	4.698



(그림 5) 전력 한도 설정에 따른 전체 평균 사용 전력

(그림 5)와 <표 4>는 설정된 전력 한도에 따라 실제로 측정된 사용 전력 및 스트리밍 된 QoE 값을 나타내었다. 우리는 해당 실험의 결과를 통해 아래의 3가지를 알 수 있었다.

① 전력 한도와 실제 사용 전력의 근접도 : 전력 한도는 각각 전체를 트랜스코딩 했을 때의 20%, 40%, 60%, 80%이며, 각각 실제 사용된 전력은 설정한 전력한도 대비 −4.861%, −1.474%, −0.101%, 1.283%의 차이가 있다. 모두 전력 한도에 근접한 것을 확인할 수 있다.

20%의 경우 설정된 전력한도 값이 적어서 비율로 봐서는 차이가 크게 보이나, 절대적인 실제 사용

www.kci.go.kr

전력과 전력한도의 차이값은 −1.196Wh이며, 실제로 40% 이상부터는 비율로도 오차가 거의 없는 것으로 나타난다.

② 스트리밍 된 QoE 값 : 전체를 트랜스 코딩했을 때 스트리밍된 QoE과 전력 한도에 맞춰 트랜스코딩했을 때의 스트리밍된 QoE를 비교 시 −2.994%, 0.730%, −0.178%, −0.008%의 차이를 보인다. 이는 전체를 트랜스코딩 할 경우의 QoE 값에 근접한다. 특히 20%의 케이스를 제외하고 40% 이상부터는 차이가 거의 없음을 확인할 수 있다.

③ 트랜스코딩 소요 시간 : 각각 평균 13분 20초, 26분 22초, 39분 3초, 52분 58초가 사용되었다. 전체 버전을 트랜스코딩 했을 때 1시간 5분 45초가 걸리는 것에 비해 설정한 전력 한도가 줄어들수록 트랜스코딩에 사용된 시간도 줄어들어 스트리밍 서비스를 준비하는 시간이 더욱 빨라지게 된다.

해당 시스템은 전력 및 트랜스코딩 소요시간과 스트리밍 되는 QoE를 효과적으로 최적화 하는 것을 확인할 수 있었다. 그러므로 해당 시스템은 의도한 대로 진행되었으며 그러므로 본 시스템은 신뢰할 수 있다.

6.3 N^{back}에 따른 결과 차이 확인

〈표 5〉 전력 한도 설정에 따른 사용 전력

N^{back}	실험 회차			
	1	2	3	평균
1	23.43Wh	23.39Wh	23.40Wh	**23.41Wh**
2	23.06Wh	23.06Wh	23.05Wh	**23.06Wh**
3	23.02Wh	23.04Wh	23.03Wh	**23.03Wh**
4	23.00Wh	23.00Wh	23.04Wh	**23.01Wh**

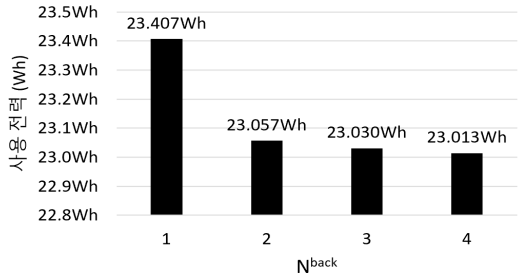
〈표 6〉 전력 한도 설정에 따른 트랜스코딩 시간(초)

N^{back}	실험 회차			
	1	2	3	평균
1	812.728	792.997	791.757	800.827
2	784.635	784.446	784.567	784.549
3	786.33	787.86	785.585	786.592
4	790.871	784.149	793.352	789.457

우리는 백엔드 노드 개수 N^{back}의 변화에 따른 차이를 확인하기 위하여 백엔드 노드를 1~4개 사용할 때에 대한 실험을 실시하였으며, 전력 한도는 백엔드 노드가 1개일 때 전체를 트랜스코딩 했을 때의 소모 전력의 20%으로 고정하여 실험을 진행하였다.

<표 5, 6>은 백엔드 노드 개수에 따른 사용 전력과 트랜스코딩 시간의 측정 결과이다. 우리는 해당 실험을 통해 아래의 2가지를 알 수 있었다.

① 사용 전력의 차이 : 사용 전력은 N^{back}이 증가할 때 마다 감소하는 것을 보였다. 특히 백엔드 노드가 1개에서 2개로 증가했을 때 전력이 제일 많이 감소했다(그림 6).



(그림 6) N^{back}에 따른 평균 사용 전력

이러한 사용 전력의 차이가 유의한 지 살펴보기 위해 ANOVA 검정을 실시한 결과, F비는 383.596, F 기각치는 4.066이므로 F 비 > F 기각치이다. 즉 영가설이 기각되어 통계적으로 차이가 유의하다. 그러므로 백엔드 노드 개수와 트랜스코딩 소요시간은 차이가 있으며, 이는 N^{back}이 늘어날수록 각 노드의 부하가 줄어들기 때문으로 보인다.

② 트랜스코딩 소요 시간 : 각 트랜스코딩 소요시간의 차이가 유의한지 살펴보기 위해 ANOVA 검정을 실시한 결과, F 비는 3.09, F 기각치는 4.066이므로 F 비 < F 기각치이다. 즉 영가설이 채택되어 통계적으로 차이가 유의하지 않다. 그러므로 백엔드 노드 개수와 트랜스코딩 소요시간은 차이가 없다고 할 수 있다.

www.kci.go.kr

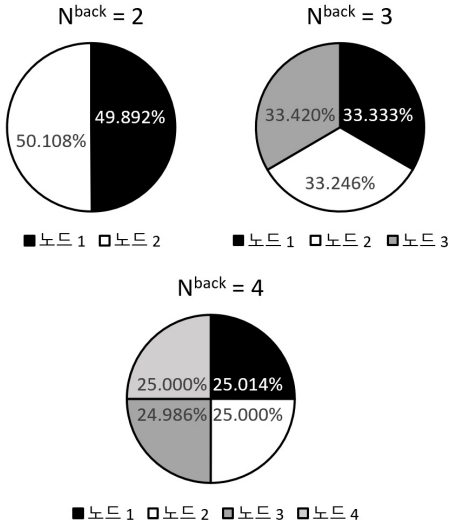
6.4 노드 별 작업 분배에 대한 신뢰성 확인

〈표 7〉 각 백엔드 노드 별 사용 전력

N^{back}	실험 회차			
	1	2	3	평균
2	11.51Wh	11.50Wh	11.50Wh	11.50Wh
	11.55Wh	11.56Wh	11.55Wh	11.55Wh
3	7.67Wh	7.68Wh	7.68Wh	7.68Wh
	7.66Wh	7.66Wh	7.65Wh	7.66Wh
	7.69Wh	7.70Wh	7.70Wh	7.70Wh
4	5.76Wh	5.75Wh	5.76Wh	5.76Wh
	5.75Wh	5.75Wh	5.76Wh	5.75Wh
	5.75Wh	5.76Wh	5.74Wh	5.75Wh
	5.74Wh	5.74Wh	5.78Wh	5.75Wh

우리는 N^{back}가 1 이상일 때, 프론트엔드 노드가 각 백엔드 노드에 작업 분배를 골고루 하였는지 확인 하고자 하였다. 전력 한도는 백엔드 노드가 1개일 때 전체를 트랜스코딩 했을 때의 소모 전력의 20%으로 고정하여 실험을 진행하였다.

<표 7>은 각 N^{back}가 2~4일 때, 각각 노드가 사용한 전력이다.



(그림 7) 각 노드 별 사용 전력의 비율

(그림 7)은 N^{back}가 2~4일 때, 각각의 노드가 사용한 전력의 비율을 나타낸 것이다. 전력이 각 노드에 골고루 분배되었음을 확인 할 수 있다.

7. 결론

본 논문에서는 트랜스코딩 태스크의 분배를 활용한 저전력 트랜스코딩 서버 시스템을 제안했다. 현재 스트리밍 서비스 및 동적 적응 스트리밍은 상당히 대중화 되었으며, 이를 위해서는 다양한 비트율의 세그먼트들의 트랜스코딩 연산이 필요하다. 그러나 트랜스코딩은 상당히 복잡한 연산을 사용하므로 많은 전력을 사용한다. 또한 트랜스코딩 서버의 작업 분배를 위한 로드 밸런싱 및 트랜스코딩 스케쥴링은 상당히 중요하다.

우리는 실제 트랜스코딩 서버 환경을 고려하기 위해 트랜스코딩 작업을 분배하고 스케쥴링하기 위한 시스템을 제작하였다. 해당 시스템은 프론트엔드 노드와 다수의 백엔드 노드로 구성된다. 프론트엔드 노드는 백엔드 노드의 관리 및 제어를 통하여 백엔드에 트랜스코딩 작업을 할당하고, 클라이언트의 요청에 대응하여 스트리밍할 버전을 전송한다.

설계한 시스템을 테스트하기 위해, 실제로 트랜스코딩 작업을 분배하고 스케쥴링함으로써 트랜스코딩을 진행하는 테스트베드를 구축하였다. 실험은 총 세 가지로 첫 번째 실험은 설정된 전력 한도에 따른 전력 결과 차이를 비교하고, 실제로 사용된 전력이 설정된 전력한도에 근접함을 확인했다. 두 번째 실험은 N^{back}의 숫자가 늘어날수록 전체 사용 전력이 줄어드는 것을 확인했다. 마지막 실험은 N^{back}가 1 이상 일 때, 트랜스코딩 작업이 각 백엔드 노드에 골고루 분산됨을 확인할 수 있었다. 우리는 실험을 통해 각 노드의 부하를 분산시킬 수 있으며, 본 시스템이 단일 노드 서버에 비해 부하가 적을 뿐만 아니라 트랜스코딩에 사용되는 시간 역시 줄어듦을 확인했다.

본 논문의 트랜스코딩 시스템은 트랜스코딩 태스크를 분배함으로써 서버를 이루는 각 노드의 부하를 줄이면서 트랜스코딩에 사용되는 전력 및 시간

www.kci.go.kr

을 최적화할 수 있음을 보여준다. 차후 연구에서는 CPU 뿐만 아니라 GPU등 이질 프로세서에서의 사용 전력 및 오버헤드를 고려한 서버 시스템을 설계하고 비트율 선택 및 작업 할당을 수행할 것이다.

▌참고문헌

〔1〕 T. Stockhammer,"Dynamic Adaptive Streaming over HTTP Standards and Design Principles", In Proceedings of the ACM Multimedia Systems Conference, pages 133 - 144, 2011.

〔2〕 Gerassimos Barlas,"Cluster-based optimized parallel video transcoding", Parallel Computing, 38(4-5):226-244, April - May 2012.

〔3〕 H. Zhao, Q. Zheng, W.Zhang, B.Du and H.Li, "A segment-based storage and transcoding trade-off strategy for multi-version vod systems in the cloud".IEEE Transactions on Multimedia, 19(1):149 - 159, Sep. 2016.

〔4〕 L. Toni, R. Aparicio-Pardo, G. Simon, A. Blanc and P. Frossard,"Optimal set of video representations in adaptive streaming", In Proceedings of the ACM Multimedia Systems Conference, pages 271 - 282, Mar. 2014.

〔5〕 R. Aparicio-Pardo, K. Pires, A. Blanc and G. Simon,"Transcoding live adaptive video streams at a massive scale in the cloud", In Proceedings of the ACM Multimedia Systems Conference, pages 49 - 60, Mar. 2015.

〔6〕 Z. Li, Y. Huang, G. Liu, F. Wang, Z. Zhang, and Y. Dai, "Cloud transcoder: Bridging the format and resolution gap between internet videos and mobile devices", In Proceedings of the ACM NOSSDAV. pages 33 - 38. Jun. 2012.

〔7〕 R. Madumitha, V. Keerthana, V. Dhivya and S. P. Tamizh Selvi,"Cloud Scheduling in Video Transcoding", In Proceedings of the International Conference on Technical Advancements in Computers and Communications, pages 102-105 Apr. 2017.

〔8〕 Y. Kim, J. Hu and J. Jeong ,"Distributed Video Transcoding System for 8K 360° VR Tiled Streaming Service", In Proceedings of the International Conference on Information and Communication Technology Convergence, pages 592-595, Oct. 2018.

〔9〕 J. Kuang, D. Guo and L. Bhuyan ,"Power optimization for multimedia transcoding on multicore servers", In Proceedings of the ACM/IEEE Symposium on Architectures for Networking and Communications Systems, pages 1-2, Oct. 2010.

〔10〕 M. Song, Y. Lee, J. Park, "A segment-based storage and transcoding trade-off strategy for multi-version vod systems in the cloud", ACM Transactions on Multimedia Computing, Communications, and Applications, 11(2):45, Feb. 2015.

〔11〕 Jungwoo Lee, Hwangje Han and Minseok Song,"Scheduling a Video Transcoding Server to Save Energy", In Proceedings of the International Symposium on Multimedia, pages 334-337, Dec. 2017.

〔12〕 백치선, 박준석,"비디오 트랜스코딩 서버의 QoS 관리 기법", 한국차세대컴퓨팅학회 논문지, 제 9권 제 4호, pp.49-56, 2013. 8.

〔13〕 조대현, 최승락, 이기용,"클러스터 비디오 서버에서 버퍼 공유를 최대화하는 동적 요청 분산 기법", 한국차세대컴퓨팅학회 논문지, 제 8권 제 2호, pp.47-61, 2012. 4.

〔14〕 오주병, 권오석,"스마트TV와 N스크린 서비스를 위한 Open Cloud 가상화 기반 사용자 서비스(STVS) 플랫폼 개발 ", 한국차세대컴퓨팅학회 논문지, 제 8권 제 2호, pp.28-39, 2014. 8.

www.kci.go.kr

〔15〕 L. Wei, J. Cai, C. H. Foh and B. He, "QoS-Aware Resource Allocation for Video Transcoding in Clouds", IEEE Transactions on Circuits and Systems for Video Technology, 27(1):49-61, Jan. 2017.

〔16〕 Y. Jin, Y. Wen and C. Westphal, "Optimal Transcoding and Caching for Adaptive Streaming in Media Cloud: an Analytical Approach", IEEE Transactions on Circuits and Systems for Video Technology, 25(12):1914-1925, Dec. 2015.

▌저자소개

◆ 이다영

- 2016년 인하대학교 컴퓨터공학부 학사
- 2018년 인하대학교 컴퓨터공학부 석사
- 2018년~현재 인하대학교 컴퓨터공학부 박사과정
- 관심분야: 모바일 플랫폼, 실시간 시스템, 임베디드 시스템, 멀티미디어 시스템

◆ 송민석

- 1996년 서울대학교 컴퓨터공학과 학사
- 1998년 서울대학교 컴퓨터공학부 석사
- 2004년 서울대학교 전기컴퓨터공학부 박사
- 2005년~현재 인하대학교 컴퓨터공학부 부교수
- 관심분야: 실시간 시스템, 임베디드 시스템, 멀티미디어 시스템

www.kci.go.kr