REPORT

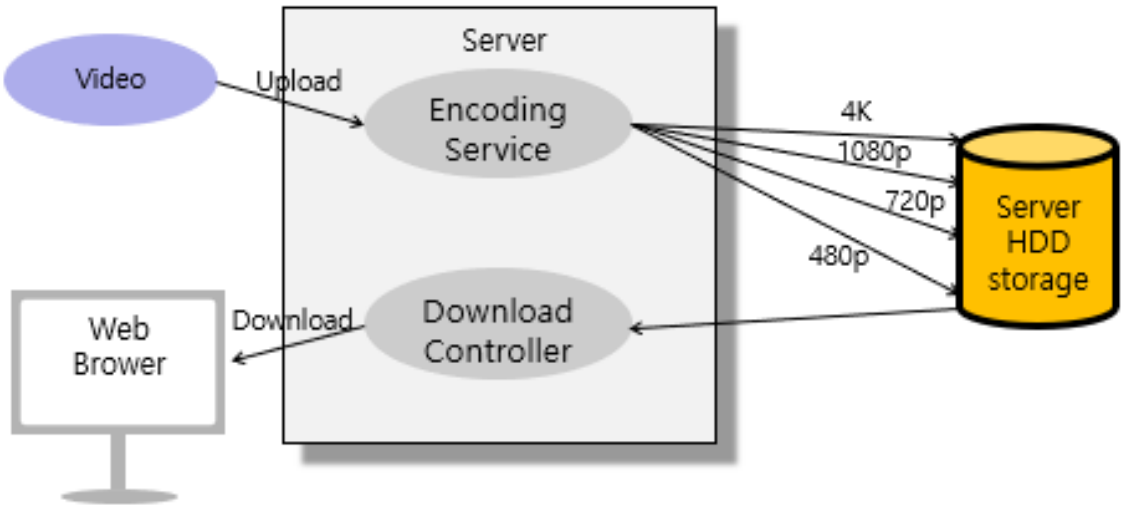
[4조 프로젝트 최종보고서]

교 과 목	실무중심산학 협력프로젝트
분 반	7
담 당 교 수	박창섭 교수님
학 과	소프트웨어학과
학 번	32141868 박유현
이 름	32144697 최광진
제 출 일	2019.12.04

목 차

1. 서론

웹상에서의 미디어는 텍스트에서 이미지를 거쳐 비디오까지 점점 발전해왔다. 현재는 영상 컨텐츠의 발전으로 인하여 다양한 화질의 영상을 제공하는데, Youtube를 기준으로 생각해 본다면, 144p 부터 4K, 최근 들어서는 8K 영상까지도 제공된다. 즉, Server storage에는 그 만큼의 인코딩된 영상만큼 추가적인 용량이 필요하게 된다. 또한 인코딩 작업이 세분화되고 양이 많아질 수록 그에 따른 시간과 용량에 대한 비용이 크게 들게 되므로 이를 줄이며 최대한 좋은 영상 품질을 얻는 방법을 찾기 위해 이 연구를 시작하게 되었다.



[그림 웹 스트리밍 서비스 모식도]

2. 도메인 분석

2.1 웹환경에서 사용할 수 있는 비디오 포맷 선정

웹 브라우저 스트리밍 시 사용될 수 있는 비디오 영상파일의 형식은 상당히 제한된다. 따라서 어떤 파일형식을 사용할지 선정해야 하는데, 먼저 비디오코덱은 다음 표와 같이 H.264 (mp4) 코덱은 대부분의 웹 브라우저에서 지원한다는 것을 알 수 있다. 그리고 조사 결과 오디오코덱은 AAC를 사용하는 것이 호환성에 알맞았기 때문에 이번 연구에서는 H.264, AAC를 사용하여 인코딩을 진행하기로 하였다.

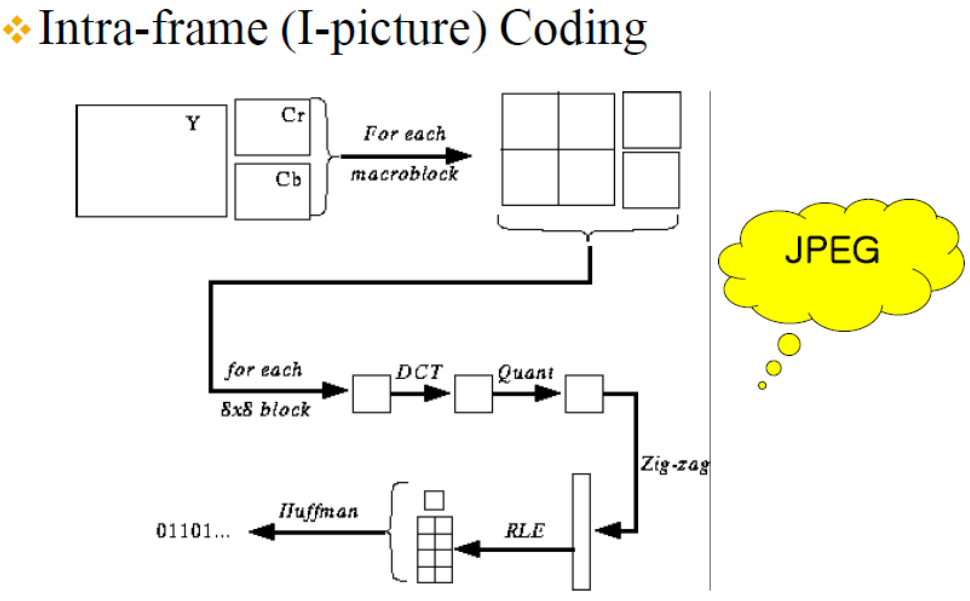
Status of video format support in each web browser

Browser	Operating System	Theora (Ogg)	H.264 (MP4)	HEVC (MP4)	VP8 (WebM)	VP9 (WebM)	AV1 (WebM)
Android browser	Android	Since 2.3[46]	Since 3.0[46]	Since 5.0[46]	Since 2.3[46]	Since 4.4[46]	Since 10
Chromium	Unix-like and Windows	Since r18297[47]	Via FFmpeg[48][49]	No[50]	Since r47759[51]	Since r172738[52]	Yes
Google Chrome	Unix-like, Android, macOS, iOS, and Windows	Since 3.0[53][54]	Since 3.0[54][a]	No[56]	Since 6.0[57][58]	Since 29.0[b]	Since 70[61]
Internet Explorer	Windows	Via OpenCodecs	Since 9.0[62]	No[56]	Via OpenCodecs	No	No
	Windows Phone	No	Since 9.0[63]		No		
	Windows RT		Since 10.0[63]				
Microsoft Edge	Windows 10	Since 17.0 (with Web Media Extensions)[64][65][66]	Since 12.0[67]	Needs hardware decoder[c]	Since 17.0 (supports <video> tag with Web Media Extensions and VP9 Video Extensions)[65]	Only enabled by default if hardware decoder present[70] Since 17.0 (supports <video> tag with Web Media Extensions and VP9 Video Extensions)[64][65][66]	Since 18.0 (with AV1 Video Extension)[71]
	Windows 10 Mobile	No	Since 13.0[72]		Since 15.0 (only via MSE)[73]	Since 14.0 (only via MSE)[74]	No
Konqueror	Unix-like and Windows	Needs OS-level codecs[d]					
Mozilla Firefox	Windows 7+	Since 3.5[75]	Since 21.0[e]	No[56]	Since 4.0[78][79]	Since 28.0[80][81]	Since 65.0[82]
	Windows Vista		Since 22.0[83]				
	Windows XP and N editions		Since 46.0[84]				
	Linux		26.0 (via GStreamer)[f] 43.0 (via FFmpeg)[87]				Since 67
	Android		Since 17.0[88]				in Nightly
	macOS		Since 34.0[89]				Since 66.0
	Firefox OS		Since 1.1[90]				No
Opera Mobile	Android, iOS, Symbian, and Windows Mobile	Since 13.0	Since 11.50	No[91]	Since 15.0	Since 16.0	since 57.0[61]
Opera	macOS, Windows, Linux	Since 10.50[92]	Since 24.0[93]		Since 10.60[94][95]	Yes	since 57.0[61]
Safari	iOS	No	Since 3.1[96]	Since 11[97]	Since 12.1 (only supports WebRTC)[98]	No	No
	macOS	Via Xiph QuickTime Components (macOS 10.11 and earlier)					
GNOME Web	Linux and BSD	Needs OS-level codecs[g]					

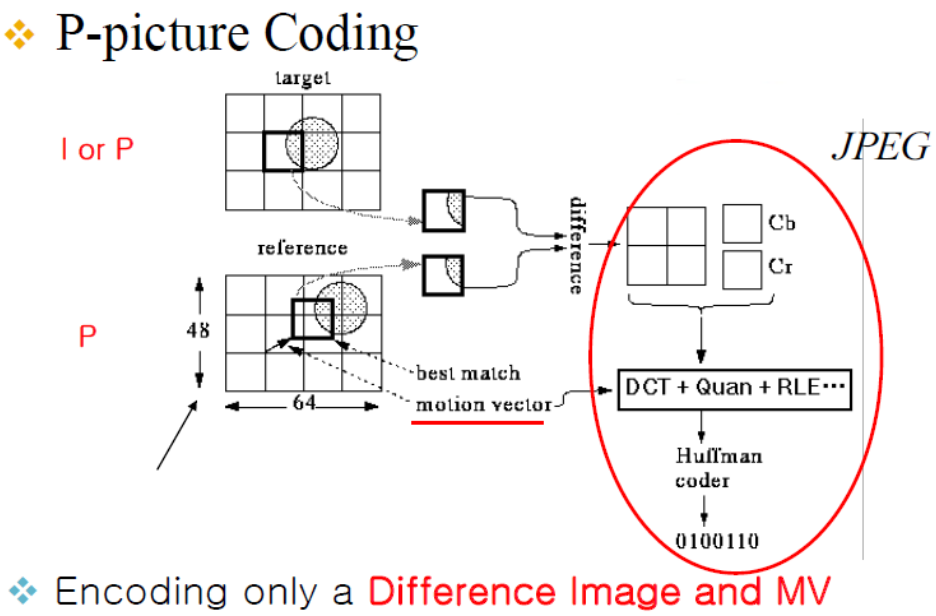
[그림 HTML5 브라우저 별 video 태그 지원 비디오 코덱]

2.2 MP4 비디오 형식 손실 압축 과정

MP4는 크게 두가지의 작업을 통해 손실 압축된다. 먼저 영상에서 특정 주기로 화면을 JPEG형식으로 압축하여 키프레임으로 뽑아내고 그 다음 키프레임 간의 차이(모션 벡터)만을 인코딩하여 영상을 구성한다. 따라서 키프레임을 뽑는 주기, 모션벡터 간의 차이 등 여러가지 영상의 품질과 용량에 영향을 미치는 요소가 생길 수 있다.



[그림 MP4 압축과정 1]



[그림 MP4 압축과정 2]

2.3 FFMPEG 사용방법

FFMPEG는 디지털 음성 스트림과 영상 스트림에 대해서 다양한 종류의 형태로 기록하고 변환하는 컴퓨터 프로그램이다. FFMPEG는 명령어를 직접 입력하는 방식으로 동작하며 여러 가지 자유 소프트웨어와 오픈 소스 라이브러리로 구성되어 있다.

2.3.1 연구 중 사용된 옵션태그

옵션 태그 명	설명	사용한 값
-movflags	웹상에서 스트리밍이 용이하도록 ATOM파일을 파일의 앞부분에 배치	faststart
-vcodec	비디오 코덱을 지정	libx264
-preset	특정 인코딩 속도 대 압축 비율을 제공	ultrafast, superfast veryfast, faster, fast medium, slow slower, veryslow
-crf	CRF 방식에서 Quality Level을 설정 값이 낮을수록 원본과 가까움	18,21,24,27
-y	파일이 이미 있더라도 덮어쓰게 하는 옵션	
-f	포맷을 지정	Mp4
-max_muxing_queue_size	Muxing 큐 크기가 너무 커져 메모리가 초과되지 않도록 하는 옵션	9999

2.4 CRF (Constant Rate Factor)

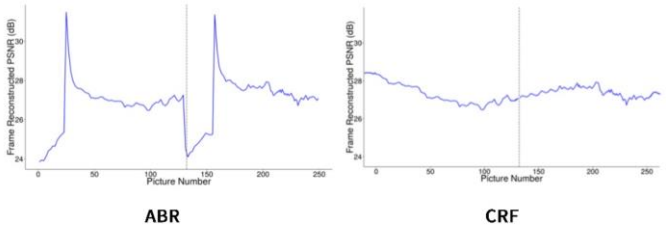
2.4.1 CRF 중요성

보다 효율적인 Bitrate 배분을 위해서 조사하던 중 if 카카오의 논문을 참고하게 되었다. 해당 논문에서는 실시간 인코딩에서의 영상 Bitrate 배분을 좀 더 효율적으로 하기위해 CRF를 사용하였다. 본 연구 또한 그 내용을 참조하여 CRF의 특성을 분석하고 이해하여 영상 인코딩에 사용해 보려고 하였다. CRF의 가장 큰 특징은 각 프레임의 압축하는 정도가 다르다는 것이다. 좀 더 자세히 표현한다면 CRF는 영상의 일부 구간까지 복잡도나 참조율(이는 CRF 가 qcomp 방식을 사용할지 Macro Block Tree 방식을 사용할지에 따라 다름)을 계산한다는 것이다. 그리고 그 참조율을 바탕으로 영상이 역동적인, 움직임이 많은 부분은 영상을 더욱 압축하여 퀄리티를 낮추고, 영상이 정적이고 움직임이 적은 부분은 영상을 덜 압축하여 퀄리티를 올린다. 그 이유는 사람은 영상이 역동적인 경우 그 화질에 상대적으로 덜 신경을 쓰게 되지만, 영상이 정적인 경우에는 화질을 자세히 보기 때문이다. 이러한 특징으로 영상의 용량을 더욱 압축하면서도 퀄리티가 좋게 느껴지게 인코딩 할 수 있다.

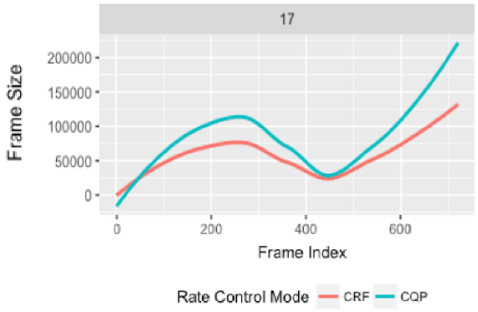
2.4.2 ABR vs CQP vs CRF

CRF가 가진 방식을 이해하기 위해서 다른 압축방식 또한 이해할 필요가 있었다. 그에 따른 이해를 토대로 표를 작성하였다.

ABP	CQP	CRF
용량을 조절하는데 초점	고정된 양자화 수치 (Quantization Parameter)로 압축을 수행함	정해진 Quality Level을 기반으로 비트를 배분함
정해진 구간 안에서 목표 Bitrate를 평균적으로 맞춤	비트 배분이라는 조절 개념이 없음	전체 구간에 대한 복잡도를 신경 쓸 필요가 없음, 일부 구간에 대하여 복잡도 계산
사용할 수 있는 비트 배분량이 정해져 있음	Quantization Parameter 값이 고정되어 있기 때문에 화면이 복잡하면 Bitrate가 높아짐, 반대로 화면이 단순하면 Bitrate가 낮아짐	복잡도에 따라서 비트를 좀더 효과적으로 배분
구간의 전체 복잡도를 미리 알 수 없기 때문에 효과적인 비트배분이 어려움		



[그림 ABR vs CRF]



[그림 CQP vs CRF]

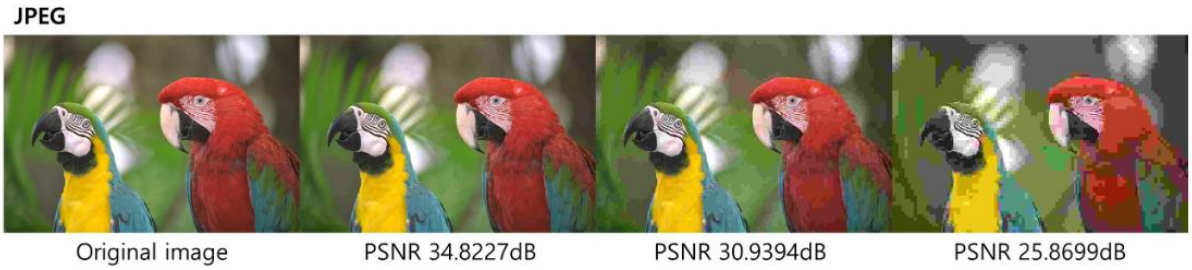
2.5 SSIM (Structural similarity)

연구를 진행함에 있어서 우리가 정한 Tradeoff의 3요소는 인코딩 시간, 용량 변화, 화질이다. 그 중 화질을 객관적으로 판단하기 위해서 화질측정 알고리즘 중 하나인 SSIM을 선택하여 사용하였다. 화질을 판단하기 위해 고려했던 알고리즘은 다음과 같이 PSNR, SSIM 두 가지가 있었다.

2.5.1 PSNR

SSIM을 이해하기에 앞서 PSNR의 이해가 필요하였다. PSNR은 최대신호 대 잡음 비(peak signal-to-noise ratio) 로써 신호가 가질 수 있는 최대 전력에 대한 잡음의 전력을 나타낸 것이다. 여기서의 잡음은 손실을 뜻한다. 즉, 원본 신호대비 인코딩 후의 손실이 얼마나 큰 지를 계산해주는 알고리즘이다.

$$
\begin{aligned}
PSNR &= 10 \cdot \log_{10}\left(\frac{MAX_I^2}{MSE}\right) \\
&= 20 \cdot \log_{10}\left(\frac{MAX_I}{\sqrt{MSE}}\right) \\
&= 20 \cdot \log_{10}(MAX_I) - 10 \cdot \log_{10}(MSE)
\end{aligned}
$$



하지만 PSNR은 얼마나 손실되었는지 만을 보기 때문에 실제로 우리가 보는 것과 상당한 차이를 가질 수도 있다.


PSNR 25.8699dB


PSNR 25.7261dB

다음은 PSNR이 같은 품질을 가지고 있다고 판단한 2개의 사진이다. 왼쪽은 JPEG의 손실압축을 진행한 사진이고, 오른쪽은 blur된 사진이다. 하지만 사진의 품질은 오른쪽 사진이 더욱 좋게 보인다. 이처럼 PSNR은 실제 사람의 지각품질을 반영해주지 못하기 때문에, 지각품질을 반영하여 화질을 평가하는 SSIM을 사용하게 되었다.

2.5.2 SSIM

SSIM은 "사람 시각 시스템은 이미지에서 구조 정보를 도출하는데 특화되어 있기 때문에 구조 정보의 왜곡 정도가 지각 품질에 가장 큰 영향을 미친다" 라는 핵심 가설아래 만들어진 알고리즘이다. SSIM은 이미지를 휘도, 대비, 구조 세가지의 방향성에서 본 뒤 이를 규합하여 하나의 맵을 작성한다.

$$l(x,y) = \frac{2\mu_x\mu_y + c_1}{\mu_x^2 + \mu_y^2 + c_1}$$

$$c(x,y) = \frac{2\sigma_x\sigma_y + c_2}{\sigma_x^2 + \sigma_y^2 + c_2} \quad \Rightarrow \quad \mathrm{SSIM}(x,y) = \left[l(x,y)^\alpha \cdot c(x,y)^\beta \cdot s(x,y)^\gamma\right]$$

$$s(x,y) = \frac{\sigma_{xy} + c_3}{\sigma_x\sigma_y + c_3}$$


SSIM 0.7380


SSIM 0.8646

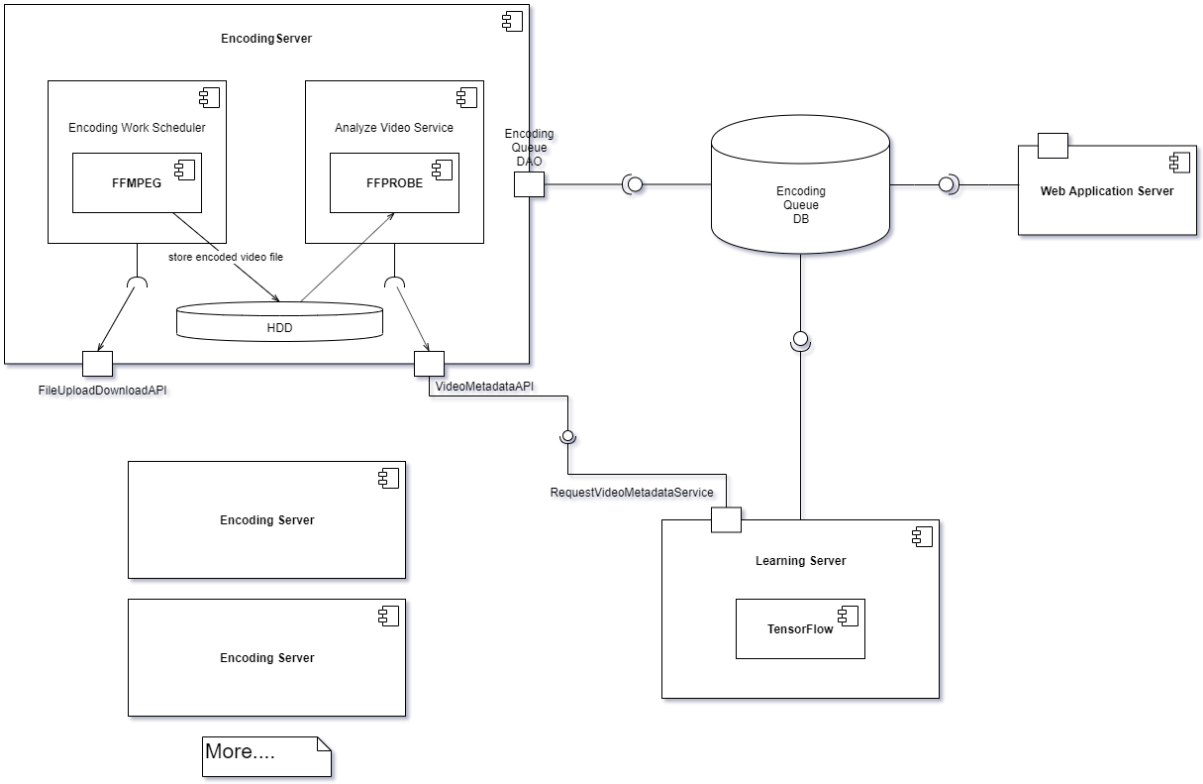
위의 2개의 사진을 SSIM으로 다시 판단해보면 오른쪽 사진의 점수가 더 높게 나온다. 즉 오른쪽의 사진이 원본과 더욱 구조적으로 유사하다고 평가내린것이다. 우리는 이 SSIM을 사용하여 영상의 썸네일을 주기적으로 뽑아 SSIM으로 평가함으로써 객관적으로 영상의 품질에 점수을 매겼다.

3. 구현 과정

3.1 구조 설계

3.1.1 분산 서버 설계

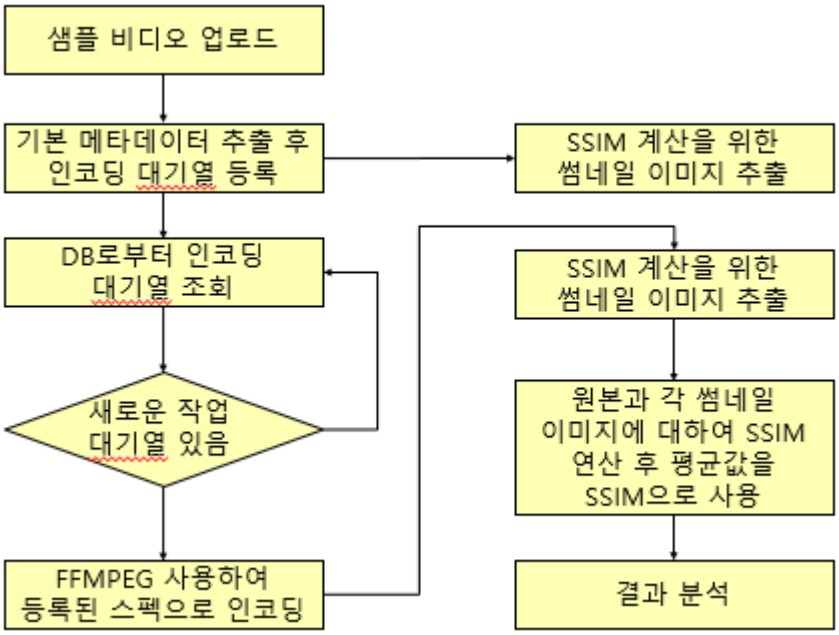
인코딩과 머신러닝 과정은 CPU 집약적인 작업으로 하나의 PC에서 테스트하기에는 무리가 있을 것으로 판단하여 각각의 기능들을 분산시켜 각각 맡은 작업을 진행하도록 구현하였다. 크게 인코딩 작업 연산을 담당하는 'EncodingServer'(이하 인코딩서버), 인코딩 연산 결과물을 분류 및 학습하는 서버인 'LearningServer'(이하 러닝서버)로 나누고 이 둘을 관리할 수 있는 간단한 웹 어플리케이션 서버를 구현하기로 하였다. 이들을 간단하게 그림으로 표현하면 다음과 같다.



[그림 Project SCV Component Diagram]

위 그림과 같이 서버간 통신은 DB를 통하여 구현되었고, 작업 부하가 큰 인코딩 서버는 물리적으로 여러 개인 컴퓨터에 분산하여 병렬처리 할 수 있도록 구현하였다. 실제로 비디오 파일은 인코딩 서버에 저장되고 비디오 파일을 제외한 모든 데이터는 DB에서 관리되게 구현하여 인코딩서버와 러닝서버에서 사용되는 데이터들을 웹 어플리케이션 서버에서 조회 및 관리할 수 있도록 하였다.

3.1.2 인코딩 작업 진행 시퀀스 설계

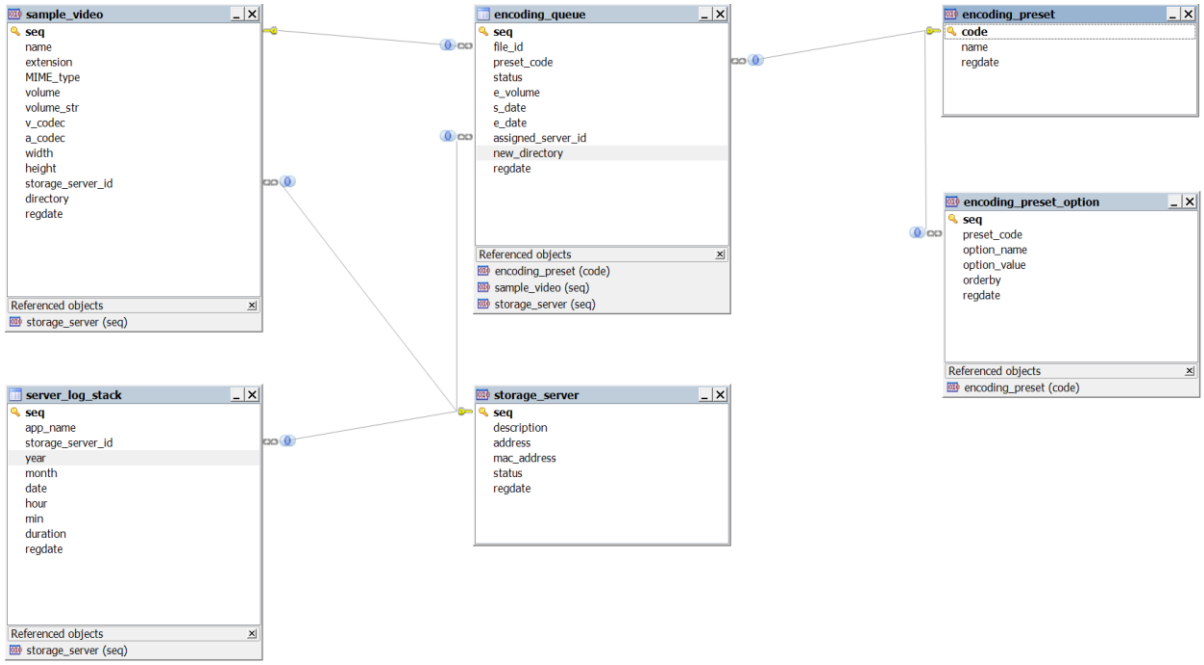


[그림 인코딩 작업 진행 순서도]

인코딩 작업은 위 순서도와 같이 구현하였다. 작업에 관련된 모든 데이터는 DB를 통해 들어오고 나가기 때문에 구현에 큰 어려움은 없었다. 해당 작업은 진행중인 작업이 없을 경우 주기적으로 DB를 조회하도록 구현하여 DB에 추가되는 새로운 작업 대기열을 받아올 수 있도록 구현하였다.

인코딩 작업 진행에 따라 영상이 인코딩되면 우리가 필요한 정보인 용량변화, 인코딩 작업 진행 시간, SSIM을 계산하여 DB에 등록한다. 따라서 다른 Application에서는 영상을 직접 다운로드 받아 판단할 필요 없이 결과 데이터만으로 영상에 대한 판단을 내릴 수 있게 된다.

3.2 DB 설계



[그림 DB E-R Diagram]

각 테이블이 저장하는 데이터는 다음과 같다.

- storage_server: 분산된 서버들의 기본정보와 현재 상태(온라인/오프라인)를 반영하여 저장

- server_log_stack: 생성되는 로그파일들의 정보를 저장

- sample_video: 업로드된 샘플 비디오영상의 기본적인 정보를 저장

- encoding_queue: 요청한 작업 대기열에 대한 정보를 저장, 러닝서버에서 추가하고 인코딩서버에서 조회하여 작업을 진행한다.

- encoding_preset: FFMPEG에서 사용될 argument 값들의 집합이름을 저장

- encoding_preset_option: encoding_preset에 저장된 집합의 상세 옵션을 저장

여러 서버에서 동시에 접근하여 사용하기 때문에 각 테이블에 어떤 서버에서 요청했는지, 어떤 서버에게 작업이 할당되었는지를 같이 저장하여 좀 더 매끄럽게 시스템이 흘러갈 수 있도록 구현하였다.

3.3 EncodingServer 구현

먼저 구현에서 사용된 기술스택은 다음과 같다.

- OS : Microsoft Windows 10

- Language : Java Servlet - JRE 1.8

- Web server : Tomcat 8.0

- Web framework : 해당없음

- IDE : Eclipse Jee

- External APIs : FFMPEG, COS.jar

서버 구현 시 인코딩 작업의 효율을 위해 별도의 프레임워크를 사용하지 않았고 기본기능만 탑재한 자체적인 프레임워크를 탑재하였다. 기본적으로 구현한 내용은 2.1.1, 2.1.2에서 설명했던 기능이다. 대표적인 코드는 다음과 같다.

3.3.1 EncodingService.java (일부)

```
…       @Override
27      public void contextInitialized(ServletContextEvent arg0) {
28              String ffmpegPath = ServerConfig.getFFMPEGPath();
…               try {
                        File file = new File(ffmpegPath);
                        if (!file.exists())
                                throw new Exception();
                } catch (Exception e) {
                        throw new RuntimeException(e);
                }

                int schedulerThreadsAmount = 1; //주기적으로 동작하는 스케줄러 초기화
                scheduler = Executors.newScheduledThreadPool(schedulerThreadsAmount);
                LogUtil.printLog("Encoding service started. It holds " +
schedulerThreadsAmount + " amounts of Threads.");

                queues = new ArrayList<>();
                for (int i = 0; i < schedulerThreadsAmount; i++) {
                        EncodingQueue tempQueue = new EncodingQueue();
                        tempQueue.observeProcess();
                        queues.add(tempQueue);
                }
        }
```

위 코드는 2.1.2에서 설명했던 서버가 실행되는 순간에 주기적으로 DB로부터 작업 대기열을 조회하며 내부적으로 작업을 할당하는 EncodingQueue를 생성하는 코드이다.

3.3.2 WindowsAppProcessBuilder.java (일부)

```
…
29
…
        /**
         * 윈도우 응용프로그램(exe파일 등..) 실행시키는 유틸
         * 개발모드(서버 전역설정 devmode)일때는 실행결과가 콘솔창에 출력됨
         * @param cmdLine
         * @return
         */
        public boolean process(String[] cmdLine) {
                // 프로세스 속성을 관리하는 ProcessBuilder 생성.
                ProcessBuilder pb = new ProcessBuilder(cmdLine);
                pb.redirectErrorStream(true);
                Process p = null;
                try {
                        p = pb.start();
                } catch (Exception e) {
                        e.printStackTrace();
                        p.destroy();
                        LogUtil.printLog("프로세스 진행 중 에러 발생");
                        return false;
                }
                exhaustInputStream(p.getInputStream());   // 자식 프로세스에서 발생되는
inputstrem를 소비시켜야합니다.

                try {
                        // p의 자식 프로세스의 작업이 완료될 동안 p를 대기시킴
                        p.waitFor();
                } catch (InterruptedException e) {
                        p.destroy();
                }

                // 정상 종료가 되지 않았을 경우
                if (p.exitValue() != 0) {
                        LogUtil.printLog("프로세스가 정상종료되지 않음.");
                        return false;
                 }
                p.destroy();
                return true;
        }
```

위 코드는 직접적으로 윈도우 운영체제에서 프로세스를 만들어 실행시키는 코드이다. 프로세스에 대한 상세 커맨드라인은 DB에서 조회하여 생성된다.

3.4 웹 어플리케이션의 구현

웹 어플리케이션에서는 인코딩 작업 상황의 확인, 추가를 위한 기능들이 구현되었다.

접속 가능한 주소 : http://welcome.blog-yh.kr/SCV/

인코딩 서버 확인 페이지

Searching for Compress optimization of Video - Management version : 1.2.1 u191128

인코딩 서버 목록 | 샘플 비디오 목록 | 인코딩 옵션 관리 | 인코딩 큐 목록 | 결과 차트 | 결과 비디오 재생

번호	서버별칭	주소	MAC	상태	최근 접속일자
6	initConfig-dev-yh	112.186.29.25:8080	7085C28FC389	offline	2019.11.27(수) 오후 10:31:41
8	initConfig-dev-yh2	172.31.36.243:8080	74E5F95D69CA	offline	2019.11.28(목) 오전 09:33:57
9	KJ_labtop	192.168.0.8:8080	E09D31F1D1E1	offline	2019.10.16(수) 오후 12:54:14
11	initConfig-server	112.186.29.44:80	74D43516744F	online	2019.11.28(목) 오전 09:34:41

prev refresh next

샘플비디오 확인 페이지

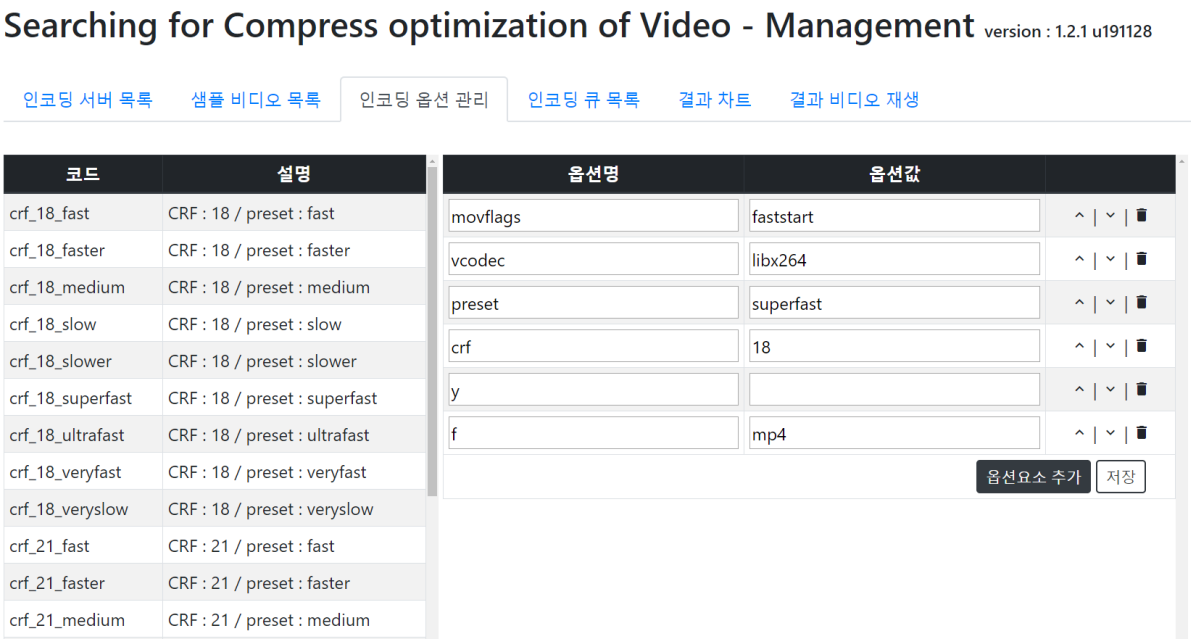
Searching for Compress optimization of Video - Management version : 1.2.1 u191128

인코딩 서버 목록 | 샘플 비디오 목록 | 인코딩 옵션 관리 | 인코딩 큐 목록 | 결과 차트 | 결과 비디오 재생

번호	파일명	확장자	MIME	용량	비디오 코덱	오디오 코덱	너비	높이	저장 서버	다운	인코딩 대기열 추가
18	Transient - 4K, UHD, 1000FPS_0015~0046	mkv	video/x-matroska	21.51 MB	vp9	aac	3840	2160	11	link	crf_18_fast 추가
17	(ASMR) Chipmunk Takes A Corn Cob Bigger Than Her Body_0120~0151	mp4	video/mp4	6.12 MB	h264	aac	854	480	11	link	crf_27_superfast 추가

prev refresh next

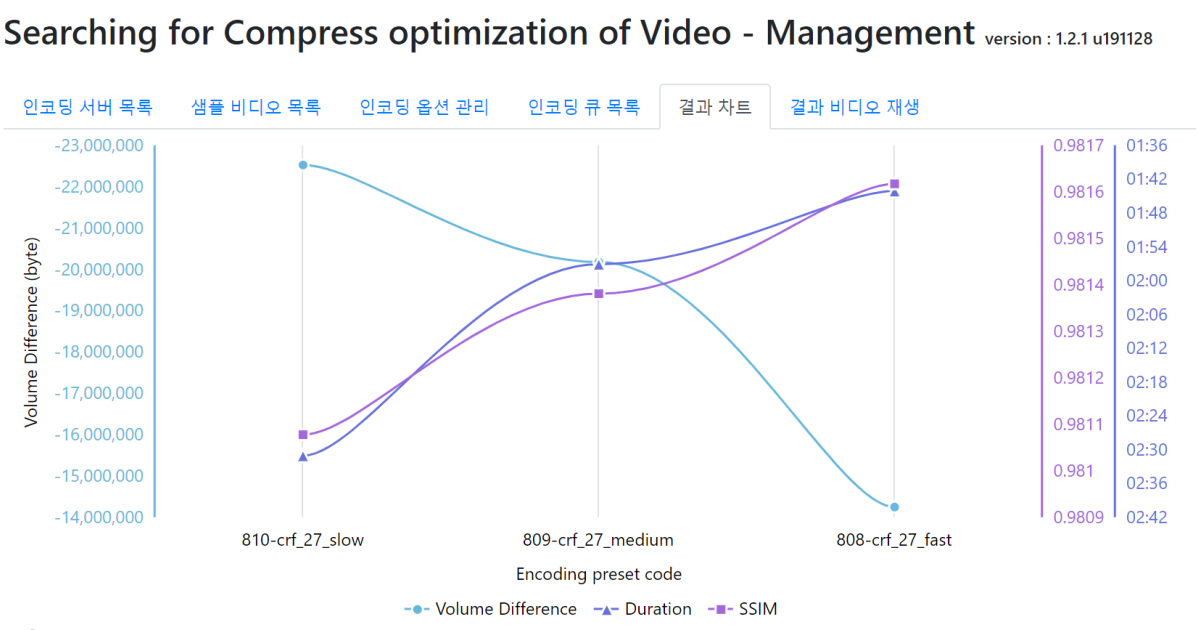
인코딩 프리셋 관리 페이지



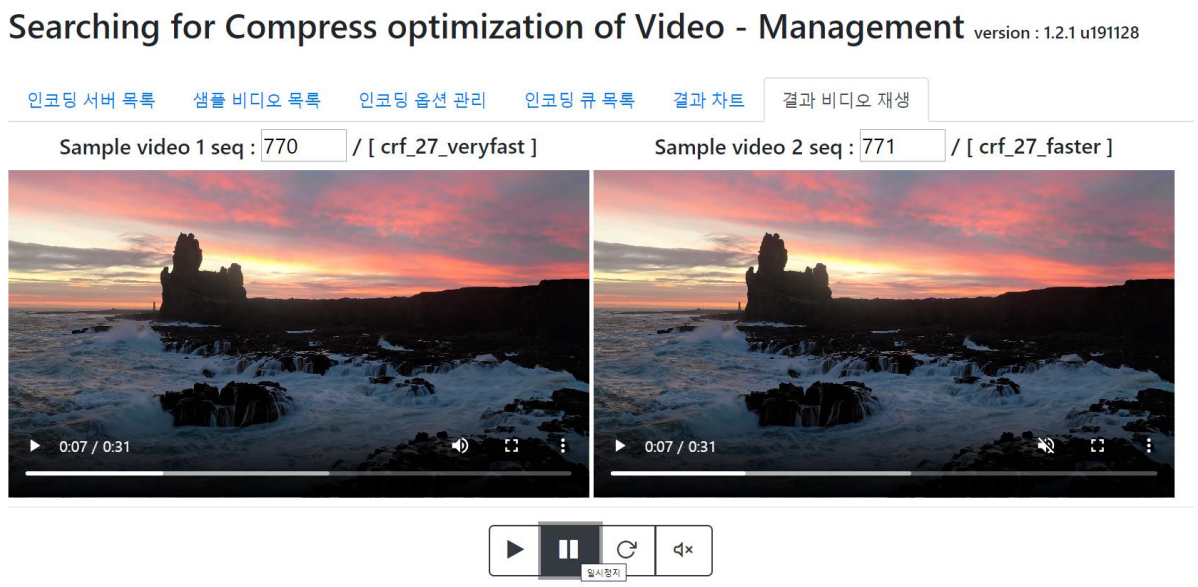
인코딩 대기열 확인 페이지

번호	파일 ID	프리셋 코드	현재상태	용량변화	소요시간	등록일자	SSIM	다운
810	26	crf_27_slow	finished	-21.49 MB	151초	2019.11.28(목) 오전 09:43:35	0.9810785803714918	영상 로그 썸네일
809	26	crf_27_medium	finished	-19.25 MB	117초	2019.11.28(목) 오전 09:43:35	0.9813819184733703	영상 로그 썸네일
808	26	crf_27_fast	finished	-13.6 MB	104초	2019.11.28(목) 오전 09:43:35	0.9816182794441135	영상 로그 썸네일
807	26	crf_27_faster	finished	-14.44 MB	89초	2019.11.28(목) 오전 09:43:35	0.9804653902670097	영상 로그 썸네일
806	26	crf_27_veryfast	finished	-26.18 MB	61초	2019.11.28(목) 오전 09:43:35	0.9756087457830619	영상 로그 썸네일
805	26	crf_27_superfast	finished	-12.44 MB	51초	2019.11.28(목) 오전 09:43:35	0.977405419602777	영상 로그 썸네일
804	26	crf_27_ultrafast	finished	8.2 MB	35초	2019.11.28(목) 오전 09:43:35	0.9720031483153543	영상 로그 썸네일
803	26	crf_24_veryslow	finished	-7.8 MB	722초	2019.11.28(목) 오전 09:43:35	0.9827054635215426	영상 로그 썸네일
802	26	crf_24_slower	finished	-4.49 MB	330초	2019.11.28(목) 오전 09:43:35	0.9832940351410832	영상 로그 썸네일
801	26	crf_24_slow	finished	-4.82 MB	169초	2019.11.28(목) 오전 09:43:35	0.98506487523483	영상 로그 썸네일
800	26	crf_24_medium	finished	-2.43 MB	129초	2019.11.28(목) 오전 09:43:35	0.9853276179911671	영상 로그 썸네일
799	26	crf_24_fast	finished	3.48 MB	112초	2019.11.28(목) 오전 09:43:35	0.9852198128768648	영상 로그 썸네일

인코딩 결과 요약 페이지



인코딩 결과 영상 스트리밍 테스트 페이지



그 외 구현 작업내역은 다음의 깃허브 주소로 접속하여 확인할 수 있다.

https://github.com/kevin0309/Searching_for_Compress_optimization_of_Video/commits/master

4. 연구 진행과정

4.1 CRF 옵션에 대한 연구

4.1.1 연구 과정

FFMPEG에서 사용되는 'crf'와 'preset' 옵션을 변경하며 인코딩 작업을 진행하였다. 'crf'옵션은 각각 18, 21, 24, 27 4종류 그리고 preset 옵션은 각각 ultrafast, superfast, veryfast, faster, fast, medium, slow, slower, veryslow의 9종류로 조합 가능한 모든 경우의 수를 테스트하였다.

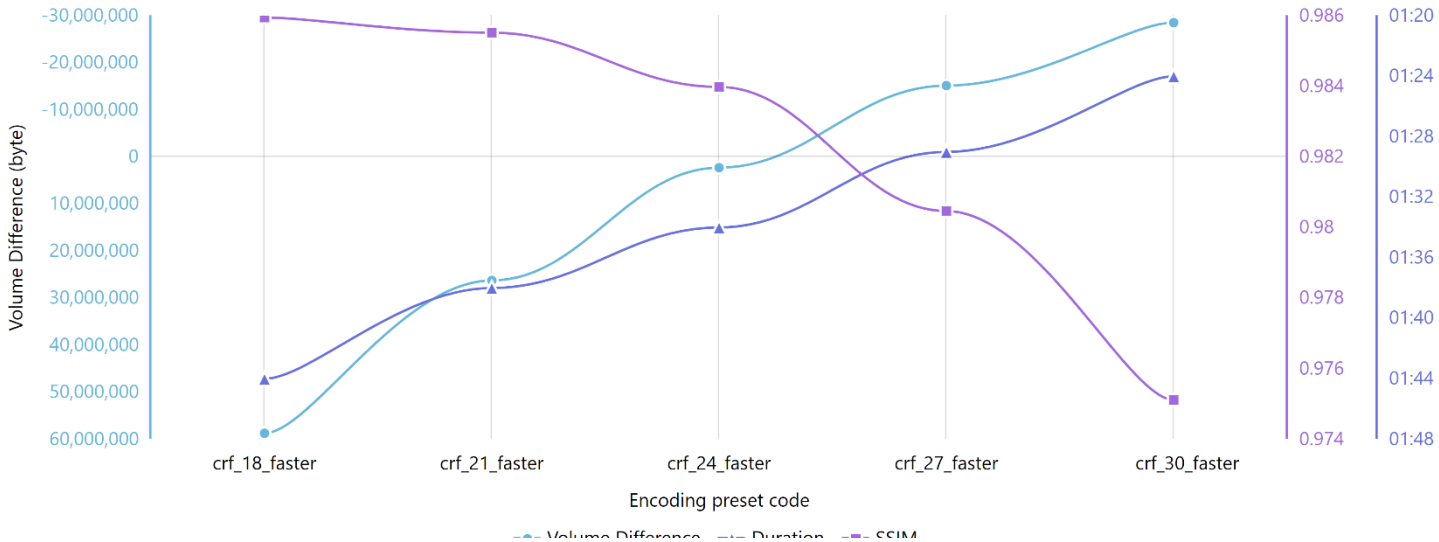
seq *	file_id *	preset_code *	status *	e_volume	s_date	e_date	assigned_server_id	new_directory	regdate *
7	5	crf_18_ultrafast	finished	79958407	2019-10-15 오후 3:43:19	2019-10-15 오후 3:43:28	9	C:/Dev/99tem...	2019-10-14 오전 12:00:00
8	5	crf_18_veryslow	finished	28550337	2019-10-15 오후 3:52:09	2019-10-15 오후 3:55:07	9	C:/Dev/99tem...	2019-10-14 오전 12:00:00
9	1	crf_18_ultrafast	finished	79917173	2019-10-15 오후 4:02:33	2019-10-15 오후 4:02:37	6	D:/DEV/99. te...	2019-10-16 오전 12:00:00
10	1	crf_18_veryslow	finished	28504904	2019-10-15 오후 4:02:57	2019-10-15 오후 4:04:16	6	D:/DEV/99. te...	2019-10-16 오전 12:00:00
11	1	crf_18_ultrafast	finished	79917173	2019-10-15 오후 4:10:47	2019-10-15 오후 4:10:50	6	D:/DEV/99. te...	2019-10-16 오전 12:00:00
12	5	crf_18_fast	finished	31073109	2019-10-16 오전 2:23:09	2019-10-16 오전 2:23:40	9	C:/Dev/99tem...	2019-10-16 오전 11:23:01
13	5	crf_21_ultrafast	finished	54310984	2019-10-16 오전 2:23:50	2019-10-16 오전 2:23:57	9	C:/Dev/99tem...	2019-10-16 오전 11:23:01
14	5	crf_21_fast	finished	19115737	2019-10-16 오전 2:24:07	2019-10-16 오전 2:24:31	9	C:/Dev/99tem...	2019-10-16 오전 11:23:01
15	5	crf_21_veryslow	finished	17636140	2019-10-16 오전 2:24:41	2019-10-16 오전 2:27:05	9	C:/Dev/99tem...	2019-10-16 오전 11:23:01
16	5	crf_24_ultrafast	finished	35796845	2019-10-16 오전 3:24:24	2019-10-16 오전 3:24:34	9	C:/Dev/99tem...	2019-10-16 오후 12:24:13
17	5	crf_24_fast	finished	11749719	2019-10-16 오전 3:24:44	2019-10-16 오전 3:25:17	9	C:/Dev/99tem...	2019-10-16 오후 12:24:13
18	5	crf_24_veryslow	finished	10948822	2019-10-16 오전 3:25:28	2019-10-16 오전 3:28:40	9	C:/Dev/99tem...	2019-10-16 오후 12:24:13
19	5	crf_27_ultrafast	finished	23067011	2019-10-16 오전 3:28:50	2019-10-16 오전 3:28:59	9	C:/Dev/99tem...	2019-10-16 오후 12:28:39
20	5	crf_27_fast	finished	7449104	2019-10-16 오전 3:29:10	2019-10-16 오전 3:29:41	9	C:/Dev/99tem...	2019-10-16 오후 12:28:39
21	5	crf_27_veryslow	finished	7079025	2019-10-16 오전 3:29:52	2019-10-16 오전 3:32:49	9	C:/Dev/99tem...	2019-10-16 오후 12:28:39
22	8	crf_27_ultrafast	finished	115217169	2019-10-16 오전 5:33:41	2019-10-16 오전 5:33:53	6	D:/DEV/99. te...	2019-10-16 오전 12:00:00
23	8	crf_27_superfast	finished	75979129	2019-10-16 오전 5:34:03	2019-10-16 오전 5:34:26	6	D:/DEV/99. te...	2019-10-16 오전 12:00:00
24	8	crf_27_veryfast	finished	52230693	2019-10-16 오전 5:34:36	2019-10-16 오전 5:35:08	6	D:/DEV/99. te...	2019-10-16 오전 12:00:00
25	8	crf_27_faster	finished	65675086	2019-10-16 오전 5:35:31	2019-10-16 오전 5:36:24	6	D:/DEV/99. te...	2019-10-16 오전 12:00:00

인코딩서버 어플리케이션을 통해 인코딩을 진행하였으며 정상적으로 제대로 진행이 되었다는 것을 알 수 있다.

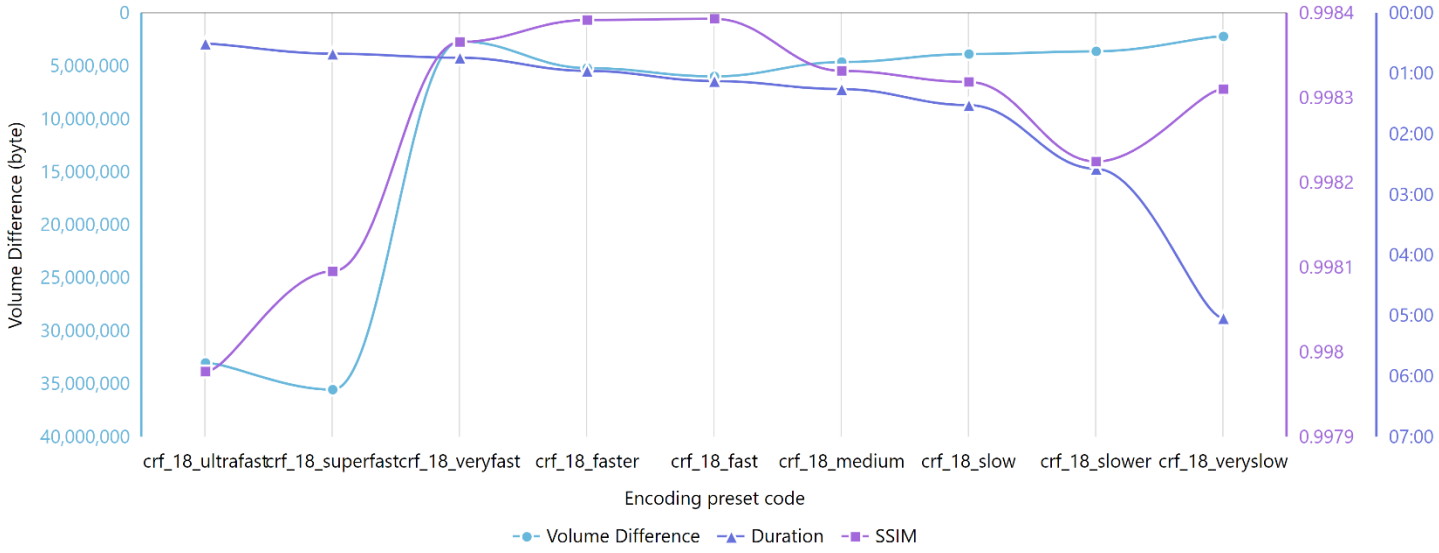
4.1.2 CRF 18,21

연구결과 그래프를 뽑아내며 몇가지 경향들을 발견할 수 있었다. CRF 18,21의 값은 Preset과 관련 없이 대부분의 영상이 인코딩 후 용량이 증가하였다. 18~21의 의 인코딩은 거의 무손실과 같은 상태의 손실압축을 진행하기 때문에 용량이 증가하는 것으로 판단된다. 그렇기 때문에 우리는 24~27의 CRF값을 위주로 판단하였다.

CRF 값에 따른 변화

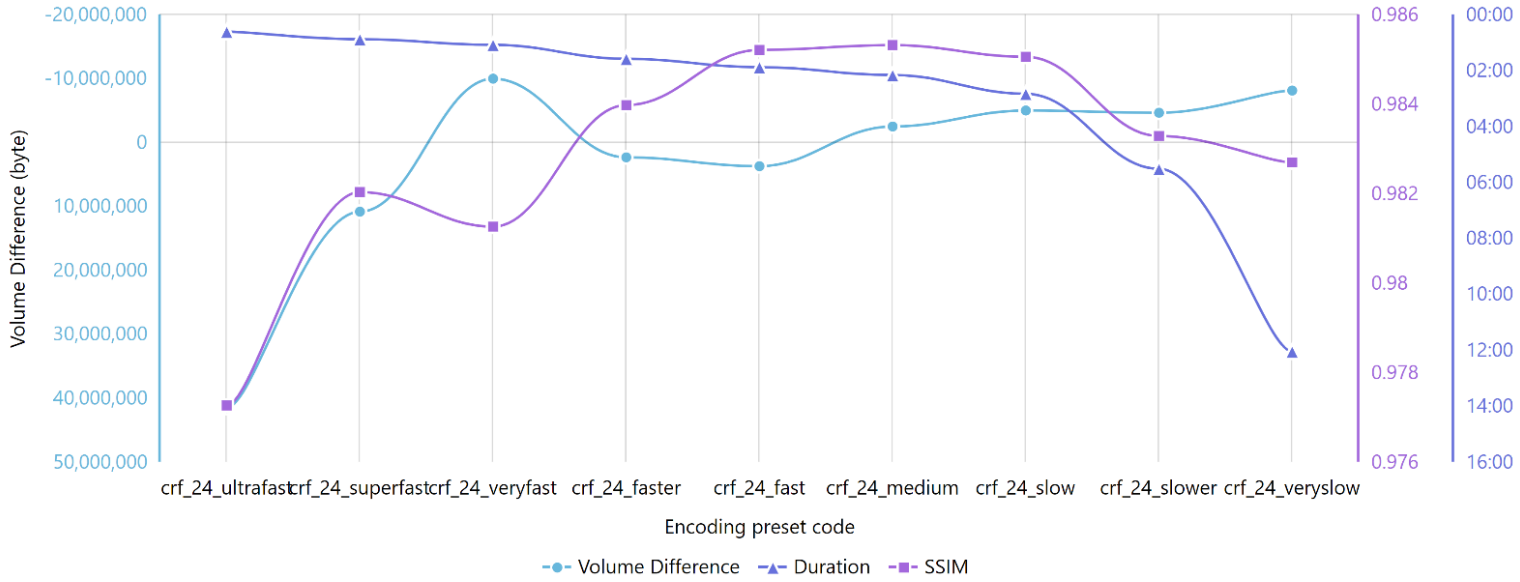


CRF 18의 용량 압축 비효율성을 나타내는 그래프

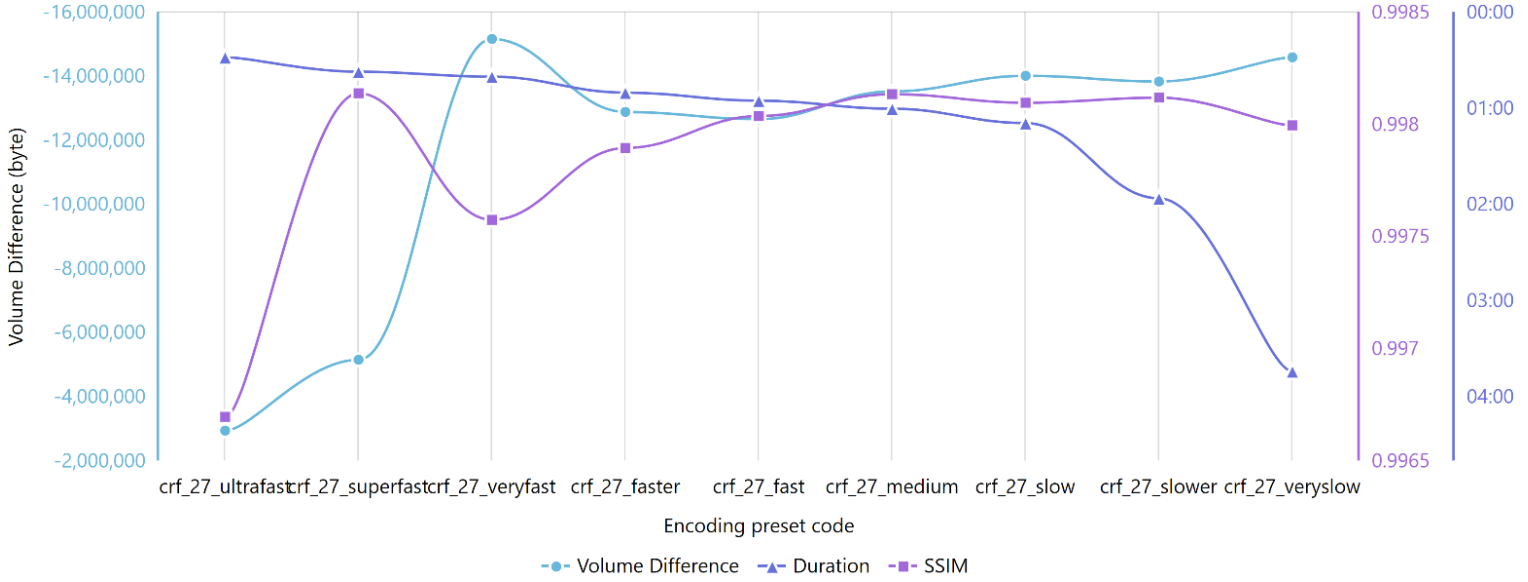


4.1.3 CRF 24,27

테스트영상 1에 대한 CRF 24 결과 그래프



테스트영상 1에 대한 CRF 27 결과 그래프



CRF값이 24, 27일 때에 의미 있는 경향이 보이기 시작했다. 좌우 양극단의 preset의 경향은 상당히 불안정하지만, faster, fast, medium, slow의 preset에서는 굉장히 안정적이면서도 좋은 성능을 내주었다. 특히 faster의 경우 다른 preset보다 빠른 속도를 가지고 (slow와 비교한다면 50초 정도가 더 빠르다.), 원본 용량의 3분의 1 수준을 압축하는 준수한 압축능력, 그리고 낮은 화질 저하를 보이고 있다. 특히, CRF값이 27일 때 그 경향이 더욱 뚜렷하게 보인다. 그렇기 때문에 우리는 CRF가 27이고 preset이 faster일 때 4K 영상을 가장 효과적으로 압축할 수 있다고 결론지었다.

5. 평가

- 화질 개선 알고리즘

앞서 얘기한 SSIM 은 완벽한 영상 화질 평가 알고리즘은 아니다. 왜냐하면 영상의 모션 벡터의 변화-시간적 특성-을 반영해주지 못하기 때문이다. 이를 개선한 MS-SSIM 이 있으나 시간의 문제로 구현하지 못하였다. 이를 구현해보고 결과의 변화를 보고싶다.

- 러닝서버의 구현

앞서 연구결과에서는 영상의 평가기준으로 용량의 변화, 인코딩 소요 시간, SSIM 세가지의 요소를 사용하여 결과를 도출했다. 하지만 하나의 결론을 도출하기 위해서 각 세가지 요소가 결과 영상에 얼만큼의 영향을 미치는지를 알아보아야 했었다. 그런데 결과를 살펴보면 SSIM 값이 기본적으로 상당히 높고, 비교 대상간 차이가 많지 않았기 때문에 러닝서버를 따로 구축할 필요가 없다고 판단했다.

참고문헌

- 위키피디아, "HTML5 video format", https://en.wikipedia.org/wiki/HTML5_video, (2019.10.23.)

ㅇ분석", https://www.slideshare.net/ifkakao/ss-113145517, (2019.10.23.)

- CRF Guide, "Constant Rate Factor", https://slhck.info/video/2017/02/24/crf-guide.html, (2019.10.23.)

- 위키피디아, "FFmpeg Wiki", https://ko.wikipedia.org/wiki/FFmpeg, (2019.10.23.)

- FFmpeg Document, "FFmpeg Document", https://ffmpeg.org/ffmpeg-all.html, (2019.10.23.)

- FFmpeg H.264 Video Encoding Guide, "FFmpeg CRF", https://trac.ffmpeg.org/wiki/Encode/H.264, (2019.10.23.)

- 최대신호 대 잡음비와 이미지 품질, "영상품질 측정", https://bskyvision.com/392, (2019.12.01.)

- Peak signal-to-noise ratio, "PSNR wiki", https://en.wikipedia.org/wiki/Peak_signal-to-noise_ratio, (2019.12.01)

- Structural similarity, "SSIM wiki", https://en.wikipedia.org/wiki/Structural_similarity, (2019.12.01)

- 2D 이미지 품질 평가에 구조변화를 반영하는 SSIM과 그의 변형들, "SSIM",

https://bskyvision.com/396, (2019.12.01)

- Z. Wang, A. C. Bovik, H. R. Sheikh, and E. P. Simoncelli, IEEE transactions on image processing, Image quality assessment: from error visibility to structural similarity, 2004.04.13