REPORT

[4조 프로젝트 중간 진행보고서]

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	2019.10.23

목 차

1. 도메인 분석

1.1 웹환경에서 사용할 수 있는 비디오 포맷 선정

웹 브라우저 스트리밍 시 사용될 수 있는 비디오 영상파일의 형식은 상당히 제한된다. 따라서 어떤 파일형식을 사용할 지 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

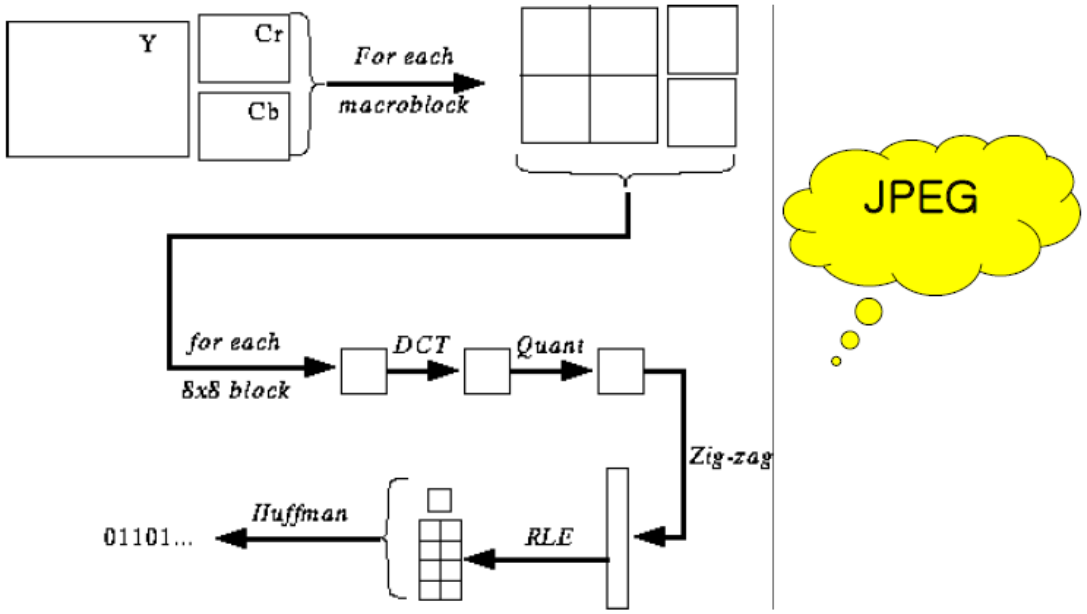
Browser	Operating System	Theora (Ogg)	H.264 (MP4)	HEVC (MP4)	VP8 (WebM)	VP9 (WebM)	AV1 (WebM)
Android browser	Android	Since 2.3[46]	Since 3.0[46]	Since 5.0[46]	Since 2.3[46]	Since 4.4[46]	Since 10
Chromium	Unix-like and Windows	Since r18297[47]	Via FFmpeg[48][49]	No[50]	Since r47759[51]	Since r172738[52]	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]

1.2 MP4 비디오 형식 손실 압축 과정

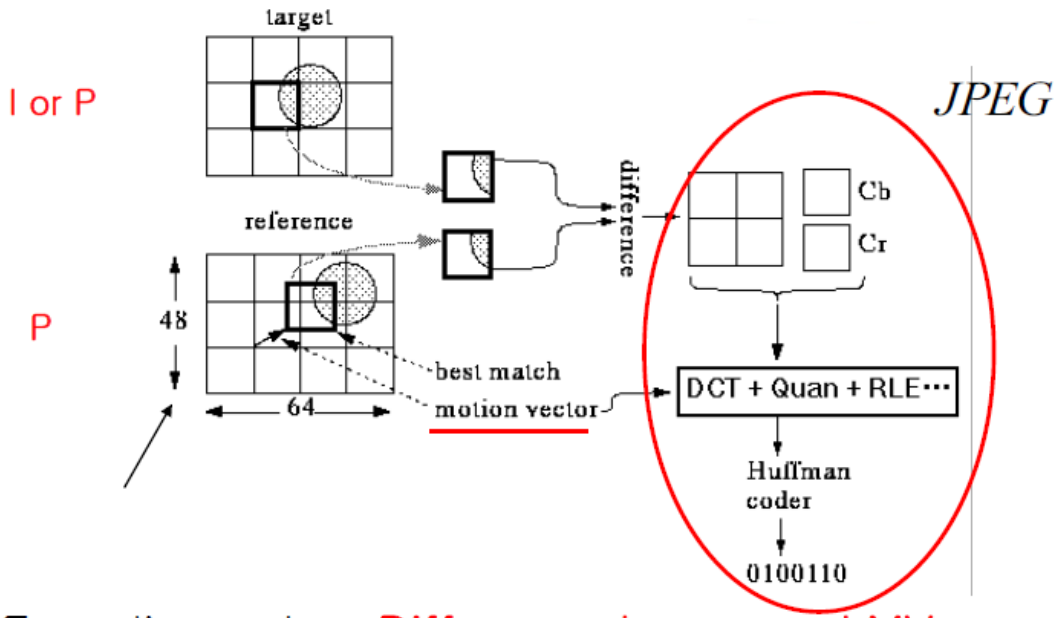
MP4는 크게 두가지의 작업을 통해 손실 압축된다. 먼저 영상에서 특정 주기로 화면을 JPEG형식으로 압축하여 키프레임으로 뽑아내고 그 다음 키프레임 간의 차이(모션 벡터)만을 인코딩하여 영상을 구성한다. 따라서 키프레임을 뽑는 주기, 모션벡터 간의 차이 등 여러가지 영상의 품질과 용량에 영향을 미치는 요소가 생길 수 있다.

❖ Intra-frame (I-picture) Coding



[그림 MP4 압축과정 1]

❖ P-picture Coding



❖ Encoding only a Difference Image and MV

[그림 MP4 압축과정 2]

1.3 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.

1.3.1 연구 중 사용된 옵션태그

옵션 태그 명	설명	사용한 값
-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

1.4 CRF (Constant Rate Factor)

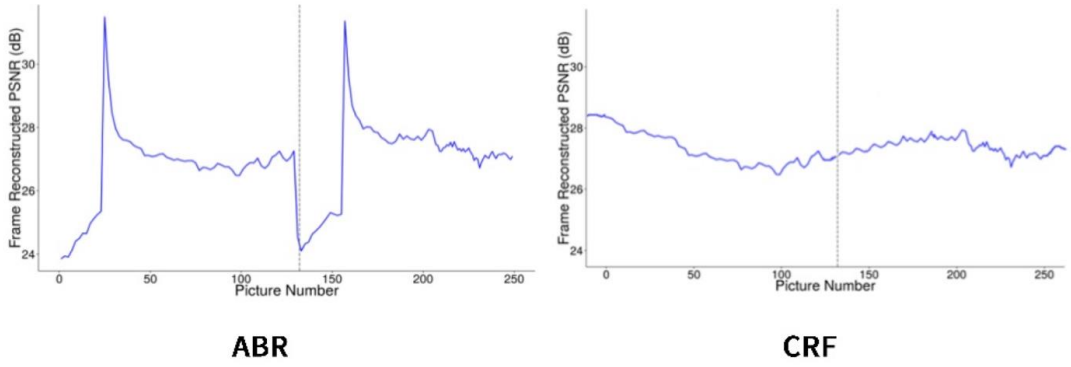
1.4.1 CRF 중요성

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.

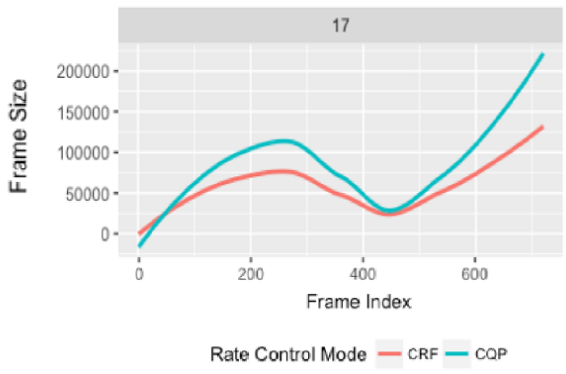
1.4.2 ABR vs CQP vs CRF

CRF가 가진 방식을 이해하기 위해서 저희는 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]



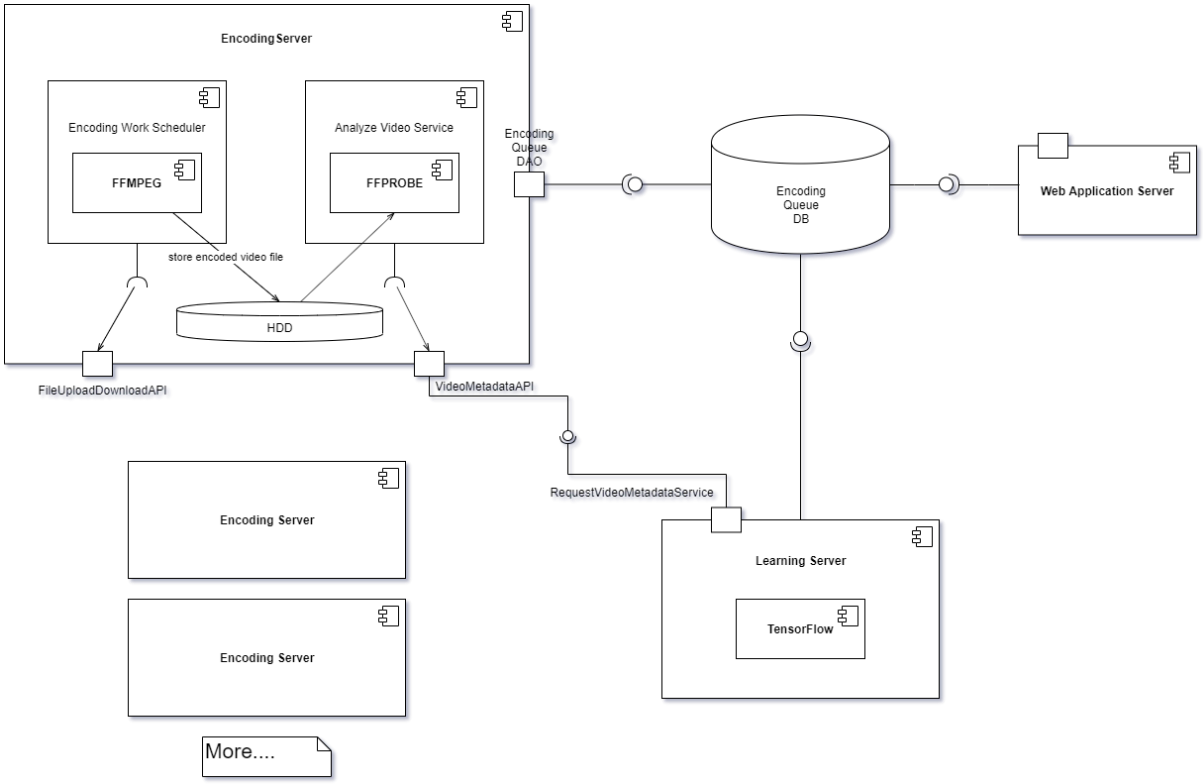
[그림 CQP vs CRF]

2. 구현 과정

2.1 구조 설계

2.1.1 분산 서버 설계

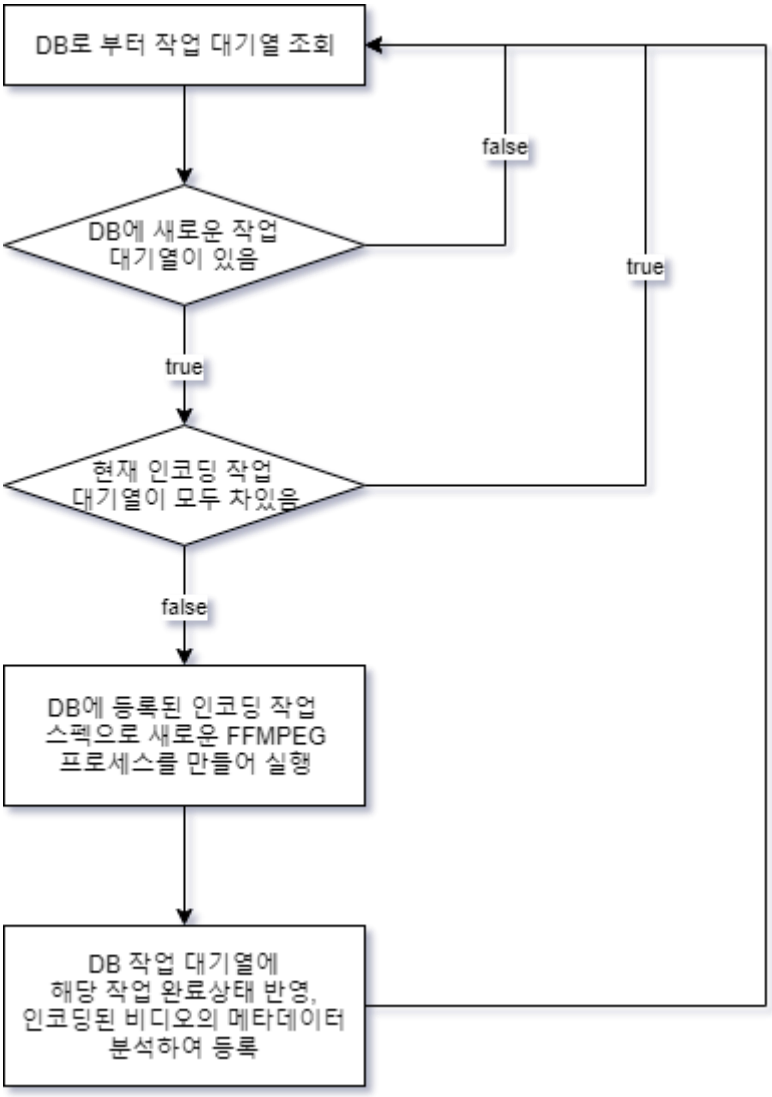
인코딩과 머신러닝 과정은 CPU 집약적인 작업으로 하나의 PC에서 테스트하기에는 무리가 있을 것으로 판단하여 각각의 기능들을 분산시켜 각각 맡은 작업을 진행하도록 구현하였다. 크게 인코딩 작업 연산을 담당하는 'EncodingServer'(이하 인코딩서버), 인코딩 연산 결과물을 분류 및 학습하는 서버인 'LearningServer'(이하 러닝서버)로 나누고 이 둘을 관리할 수 있는 간단한 웹 어플리케이션 서버를 구현하기로 하였다. 이들을 간단하게 그림으로 표현하면 다음과 같다.



[그림 Project SCV Component Diagram]

위 그림과 같이 서버간 통신은 DB를 통하여 구현되었고, 작업 부하가 큰 인코딩 서버는 여러 서버를 사용하여 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. 그리고 현재 본 중간보고서 작성 시점까지는 인코딩서버와 DB의 구현까지 진행되었다.

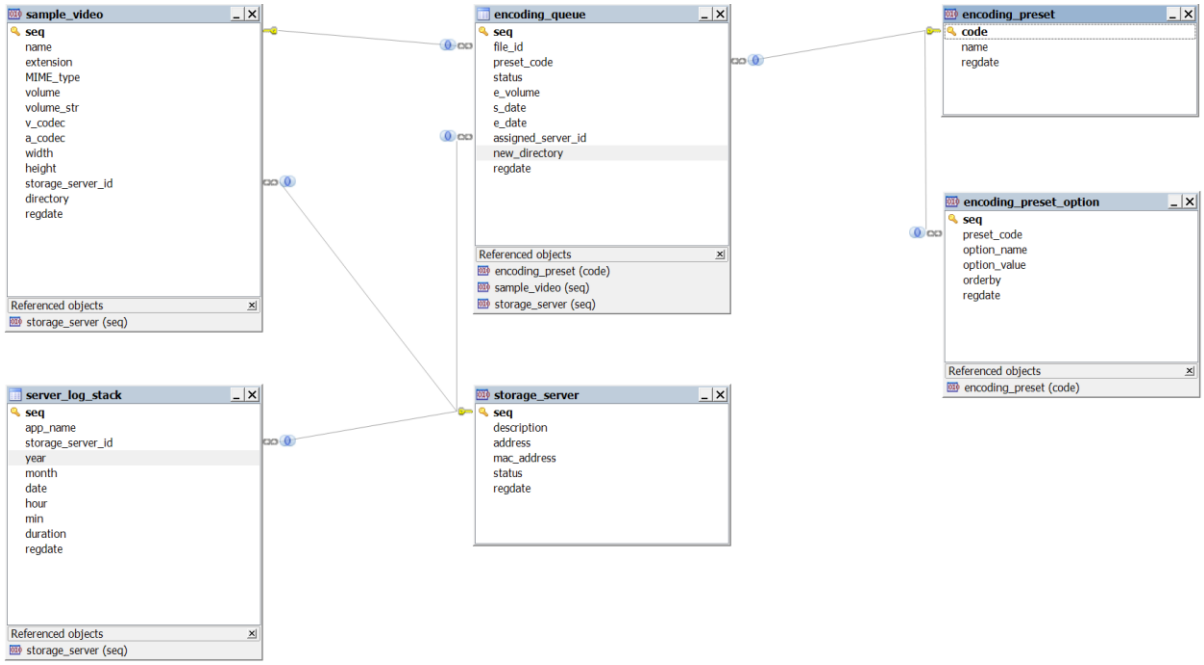
2.1.2 인코딩 작업 진행 시퀀스 설계



[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.

2.2 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.

2.3 EncodingServer 구현

먼저 인코딩 서버 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.

2.3.1 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.

2.3.2 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.

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

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

3. 연구 진행과정

3.1 샘플 영상의 선정과 선정 기준

샘플 영상의 선정기준은 최적의 인코딩 방법을 찾는 목적의 부합하는 기준을 선정하였다.

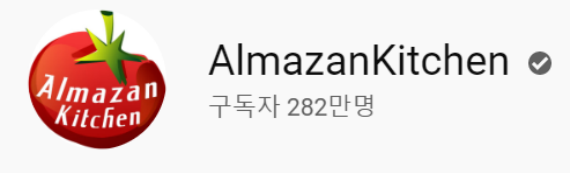
1) 화질이 충분히 좋은가?

2) 해상도가 4K (3840×2160)인가?

3) 색감이 화려한가?

4) 정적인 화면과 동적인 화면이 고루 나타나는가?

- AlmazanKitchen



4K 영상을 기반으로 하는 요리 채널

[그림 선정된 영상 중 일부 사진 캡쳐]

3.2 CRF 옵션에 대한 연구

3.2.1 연구 과정

FFMPEG에서 사용되는 'crf'와 'preset' 옵션을 변경하며 인코딩 작업을 진행하였다. 'crf'옵션은 각각 18, 21, 24, 27 3종류 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.

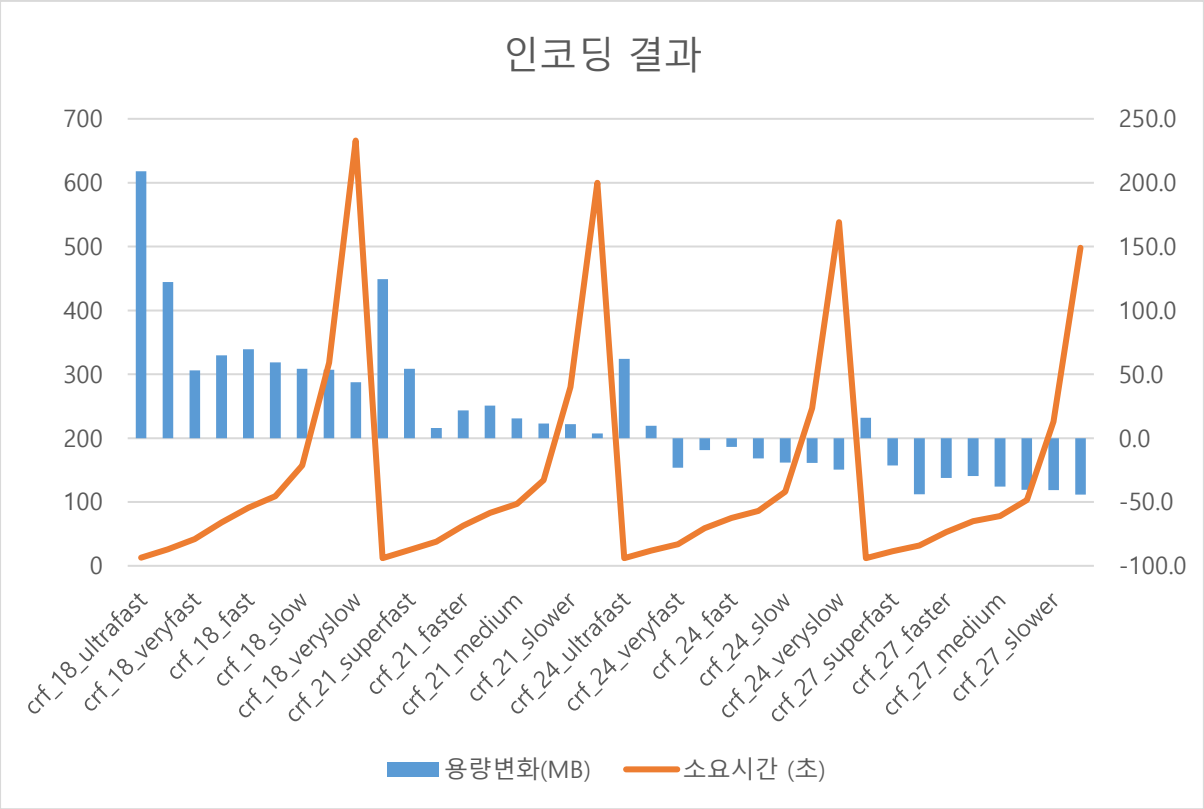
3.2.2 연구 결과 분석

영상의 화질은 4K 환경이 재생가능한 4K 모니터에서 비교 분석하였다. 또한, 연구결과를 인코딩에 걸린 시간, 인코딩 전과 후의 용량변화를 결과의 지표로 뽑았다. 그 결과 2가지의 옵션 태그가 끼치는 영향이 확연히 갈렸다. Preset의 값은 용량과 시간에 큰 영향을 끼쳤고, CRF의 값은 화질의 큰 영향을 끼쳤다. 즉, Preset의 값은 시간과 용량에 직접적으로 영향을 끼치므로 이 tradeoff를 조절하면 비용에 영향을 끼칠 것이다. CRF의 값은 화질의 직접적으로 영향을 끼치므로 이 값을 잘 조절해야 사용자의 니즈를 만족시킬 수 있을 것이다.

결과 영상은 다음 그림과 같으며 crf 값에 따른 화질의 차이가 있음을 알 수 있다.

[그림 crf 27 - preset ultrafast]

[그림 crf 18 - preset ultrafast]



4. 향후 구현계획

먼저 연구에 적합한 또 하나의 Factor가 필요하다고 생각했다. 용량과 시간은 정확한 값을 추출 가능하지만, 화질은 그런 것이 아닌 눈으로 비교해야 하는 부분이기 때문에 부정확할 수도 있다고 생각했다. 향후 머신 러닝을 도입할 것을 생각한다면, 화질을 명확히 구분해낼 Factor를 찾아내는 것이 필요해 보인다. 그렇게 한다면 머신 러닝으로 화질, 시간, 용량의 정확한 Tradeoff를 찾을 수 있을 것이다.

참고문헌

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

- 비디오 특성 분석 및 딥러닝을 이용한 실시간 인코딩 효율 최적화 SlideShare, "if kakao 비디오 분석", 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.)