다중 애플리케이션을 지원하는 설정 가능한 GStreamer 파이프라인 가상화

이하윤°, 신동군
성균관대학교 전자전기컴퓨터공학과
lhy920806@skku.edu, dongkun@skku.edu

Configurable GStreamer Pipeline Virtualization Supporting Multiple Applications

Hayun Lee°, Dongkun Shin
Department of Electrical and Computer Engineering, Sungkyunkwan University

요 약

최근 딥 러닝 등 멀티미디어 처리를 필요로 하는 애플리케이션과 임베디드 장치가 증가하고 있다. 그러나, 기존의 GStreamer에서는 다중 애플리케이션이 파이프라인을 공유할 수 없기 때문에, 이와 같은 요구사항을 만족할 수 없다. 본 논문에서는 다중 애플리케이션을 지원하는 설정 가능한 GStreamer 파이프라인 가상화를 제안하고, 가상화 모듈을 구현하였다. 가상화 모듈에서는 카메라 별로 파이프라인을 관리하고, 모든 애플리케이션의 요청을 처리한다. 본 논문에서 제안하는 가상화 모듈은 파이프라인을 설정할 수 있기 때문에 다양한 시스템에 포팅하기 쉽다. 또한 중복 동작 제거가 가능하기 때문에 다중 애플리케이션이 적은 오버헤드로 한 카메라를 동시에 사용할 수 있게 한다.

1. 서 론

최근 딥 러닝 등 영상 데이터를 처리하는 애플리케이션이 증가하고 있다. 기존에는 영상 데이터를 클라우드 서버 상에서 처리하는 경우가 많았다. 그러나, 최근에는 임베디드 하드웨어의 성능이 향상되어, 스마트폰이나 드론 등에서도 영상 처리 애플리케이션을 구동할 수 있게 되었다.

영상 처리를 위해 지금까지 많은 멀티미디어 프레임워크가 개발되었다. 멀티미디어 프레임워크의 종류로는 GStreamer[1], Phonon, DirectShow 등이 있다. 이와 같은 멀티미디어 프레임워크를 통하여 다양한 형식의 영상, 음성 데이터를 가공하고 활용하는 것이 수월해진다.

그 중에서도 GStreamer는 플러그인으로 구성된 파이프라인 구조와 쉬운 API로 인해 사용이 쉬워 널리 쓰이고 있다. 그러나, GStreamer는 카메라가 공유되지 않는 문제가 있어 서로 다른 애플리케이션에서 동시에 같은 카메라를 사용할 수 없다. 예를 들어, 영상을 지속하여 녹화하는 애플리케이션과 움직임을 인식하여 이벤트를 처리하는 애플리케이션이 동시에 동작하는 경우, 한 애플리케이션만 카메라를 선점할 수 있다.

본 논문에서는 다중 애플리케이션이 다양한 카메라 서비스를 사용할 수 있도록 하는 GStreamer 파이프라인 가상화 기법을 제안한다. 이를 위해 가상화 모듈은 인터페이스를 제공하여 애플리케이션의 요구사항을 대신 수행하게 된다. 또한, 가상화의 효율성 및 이식성을 위해 사용자가 가상화 모듈을 설정할 수 있도록 한다. 실험 결과, GStreamer 파이프라인 가상화는 다중 애플리케이션의 요청을 큰 성능 저하 없이 수행할 수 있음을 보인다.

"본 연구는 미래창조과학부 및 정보통신기술진흥센터의 SW컴퓨팅산업원천기술개발사업(SW스타랩)의 연구결과로 수행되었음" (IITP-2017-0-00914)

2. 배 경

2.1. GStreamer

GStreamer는 멀티미디어 처리를 위해 널리 쓰이는 오픈소스 멀티미디어 프레임워크다. GStreamer는 그림 1과 같이 파이프라인 구조로 되어 있어, 데이터 스트림은 파이프라인 앞에서 뒤로 흐르며 목적에 맞게 가공할 수 있다. 파이프라인은 엘리먼트(Element)와 빈(Bin)으로 구성된다. 엘리먼트는 파이프라인을 구성하는 가장 작은 단위이다. 빈은 엘리먼트 또는 빈들의 집합으로 구성된다. 빈은 자신의 원소들과 부모 자식 관계를 형성한다. 더이상 부모를 가질 수 없는 빈을 파이프라인이라고 한다.

엘리먼트를 생성하기 위해서는 플러그인이 필요하다.

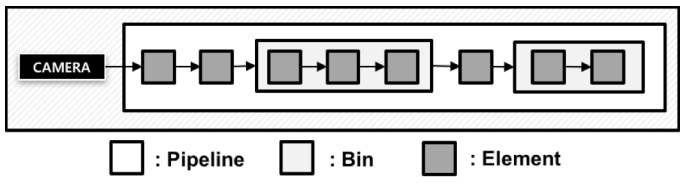


그림 1. GStreamer 파이프라인 구조

각 플러그인에는 입력으로 들어온 데이터를 어떻게 처리할 지에 대한 정의가 되어있고, 사용자는 필요한 플러그인을 엘리먼트로 생성하여 자신만의 파이프라인 구조를 쉽게 구성할 수 있다. 이미 다양한 플러그인이 제공되고, 원하는 플러그인이 존재하지 않을 때에는 API를 사용하여 직접 플러그인을 개발할 수 있다.

2.2. 카메라 공유

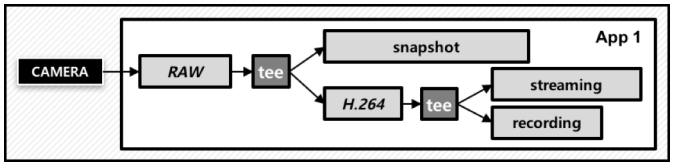
GStreamer는 tee 엘리먼트를 사용해 한 데이터를 여러 파이프라인으로 분기할 수 있다.

그림 2는 GStreamer를 사용할 때 애플리케이션이 카메라 데이터를 사용하는 시나리오를 나타낸다. 각 파이프라인을 구성하고 있는 회색 막대는 tee 엘리먼트를 제외한 엘리먼트와 빈을 추상화하여 나타낸 것이다. 논문에서는 회색 막대를 빈으로 지칭한다.

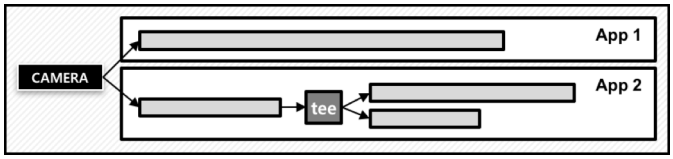
그림 2 (a)는 하나의 애플리케이션에서 카메라 데이터로 여러 기능을 구현하는 시나리오를 보여준다. App 1은 RAW 데이터를 2개의 빈으로 분기하고, 그 중 하나의 빈에서 H.264 포맷으로 인코딩한 후 다시 2개의 빈으로 분기한다. 분기된 빈들은 각자 사진, 스트리밍, 녹화 기능을 수행하다. 만약 첫번째 tee 엘리먼트에서 3개의 빈으로 분기한다면, 스트리밍과 녹화를 위해 H.264 인코딩 작업을 각 빈에서 중복으로 수행해야 한다. 즉, tee 엘리먼트를 사용하면 하나의 데이터로 여러 기능을 수행할 뿐만 아니라 중복 작업을 방지하도록 분기 지점을 선택할 수도 있다.

그러나 그림 2 (b)와 같이 다중 애플리케이션이 동시에 한 카메라를 사용하는 시나리오는 불가능하다. GStreamer는 카메라 데이터를 받기 위해 별도의 중계자 없이 카메라 장치에 바로 접근하기 때문이다.

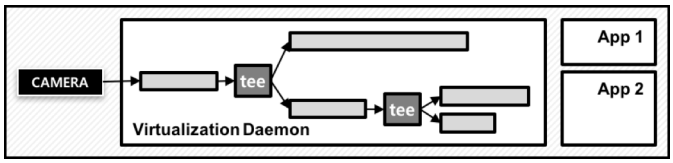
GStreamer에서 서로 다른 애플리케이션 간에 카메라 공유가 되지 않는 문제를 해결하기 위해 GStreamer 파이프라인 추상화를 제안한다. 그림 2 (c)와 같이 가상화 데몬이 카메라와 애플리케이션 사이에 중계자로서 파이프라인을 관리하여 여러 애플리케이션이 동시에 한 카메라를 사용할 수 있게 한다.



(a) Support for single application



(b) Support for multiple application (Not support)



(c) Support for multiple application using virtualization daemon

그림 2. GStreamer의 카메라 사용 시나리오

3. 설 계

GStreamer 파이프라인 가상화 기법은 그림 3과 같이 설계된다. 그림에서 파이프라인이 분기되는 지점의 tee 엘리먼트는 생략했다.

3.1. 파이프라인 가상화 API

가상화 층은 애플리케이션을 위하여 IPC(Inter-process Communication)를 사용한 API를 제공한다. IPC를 통해 전달된 요청은 가상화 데몬의 Communicator에게 전달한다.

3.2. 파이프라인 설정

가상화 층의 Configuration Parser는 사용자가 작성한

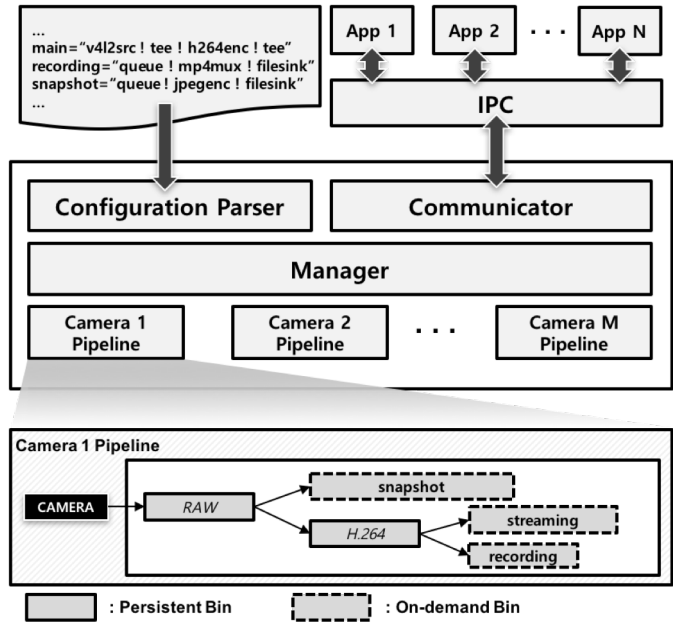


그림 3. GStreamer 파이프라인 가상화 설계

설정 파일을 읽고, 이를 통해 각 카메라마다 하나의 파이프라인을 구성한다.

파이프라인이 설정 가능하게 만든 이유는 3가지다. 첫번째는 가상화 층이 애플리케이션의 요구사항을 알 수 없기 때문이다. 애플리케이션은 자신이 직접 GStreamer 파이프라인을 구성하고 수행하는 것이 아니라, API를 통하여 가상화 데몬에게 자신의 요청을 보내게 된다. 즉, 가상화 데몬은 애플리케이션이 snapshot, streaming, recording 등의 요청을 보냈을 때, 이에 알맞는 파이프라인을 구성하기 위해 요청에 대한

빈 구조를 미리 알고 있어야 한다.

두번째는 이식성 (portability) 이다. GStreamer가 실행되는 하드웨어와 운영체제에 따라 사용할 수 있는 플러그인이 달라지거나 플러그인의 특성이 달라질 수 있기 때문에 이를 고려해야 한다. 예를 들어, Jetson TX1은 제조사인 NVIDIA에서 제공하는 하드웨어 가속 지원 플러그인을 통해 카메라를 사용할 수 있다.

세번째는 파이프라인 최적화다. 2.2.절에서 언급했듯이 파이프라인 분기 위치에 따라 중복 작업을 최소화 할 수 있기 때문이다.

3.3. 파이프라인 공유

가상화 층이 실행되면 먼저 Configuration Parser가 설정 파일을 읽어 각 요청에 따라 어떠한 빈을 이어 붙일지를 결정한다. 그리고 사용 가능한 카메라를 모두 초기화하고, 설정 파일에 있는 main을 바탕으로 main 빈 (Persistent Bin)을 생성한다. 또한 Communicator는 IPC를 초기화하여 메시지를 받을 수 있는 상태로 대기한다. 이와 같은 동작을 모두 Manager가 관리한다.

애플리케이션이 IPC를 통하여 요청을 보내면 Communicator는 해당하는 Camera로 요청을 전달한다. Camera는 Configuration Parser에게 요청에 맞는 빈 (On-demand Bin)을 전달받아 이미 설정된 main 빈의 분기점에 이어 붙인다. 예를 들어 recording이라는 요청이 올 경우 Camera는 Configuration Parser에게 “queue ! mp4mux ! filesink”라는 빈을 받아 main 빈의 두 번째 분기점에 붙이게 된다.

4. 실 험

4.1. 실험 환경

가상화를 이용한 경우와 그렇지 않았을 때, 동시에 수행하는 애플리케이션의 개수에 따른 성능 변화를 알아보기 위해 실험을 진행하였다. 실험 장비는 라즈베리 파이 2와 라즈베리 카메라를 이용하였고, 언어는 C++, IPC는 D-Bus를 사용하여 GStreamer 파이프라인 가상화 데몬을 구현하였다.

4.2. 실험 결과

그림 4는 한 카메라를 다중 애플리케이션에서 사용했을 시 평균 FPS를 측정하여 나타낸 그래프다. GStreamer는 다중 애플리케이션이 하나의 카메라를 사용할 수 없기 때문에 실험을 위해 v4l2loopback[2] 프로그램을 사용하였다. 이 프로그램은 가상의 V4L2 (Video4Linux2) 장치를 생성할 수 있다. 기존 GStreamer에서는 여러개의 가상 V4L2 장치를 생성한 후 카메라 데이터를 tee 엘리먼트를 통해 여러 가상 장치로 전달한다. 다중 애플리케이션은 각각 서로 다른 가상 V4L2 장치로부터 카메라 데이터를 받아오게 되어 마치 한 카메라로부터 동시에 데이터를 가져오는 시나리오를 만들 수 있다.

실험 결과 카메라 가상화를 이용할 때는 대략 40 FPS로 애플리케이션 개수에 따라 약간의 성능 하락이 있음을 볼 수 있다. 하지만 기존 GStreamer 수행 시에는 10 FPS까지 떨어지는 것을 볼 수 있다. 이는 기존 GStreamer 수행 시에 인코더를 각각의 애플리케이션에서 사용하기 때문에 성능저하가 심하게 나타난 것이다. 그에 비해 가상화를 이용하면 애플리케이션이 몇 개든지 한 카메라의 파이프라인에서 수행하기 때문에 인코더 중복 사용 방지와 같은 최적화가 가능하므로 성능이 유지된다.

실험에서 동시 수행하는 애플리케이션이 한 개일 때, 카메라 가상화가 기존 GStreamer보다 높은 FPS로 처리한다. 이는 카메라 가상화는 카메라 데이터를 바로 받아 요청을 처리하지만, 기존 GStreamer 실험에서는 카메라 데이터를 가상 V4L2 장치로 보내는 추가적인 오버헤드가 존재하기 때문에 성능이 낮아진다.

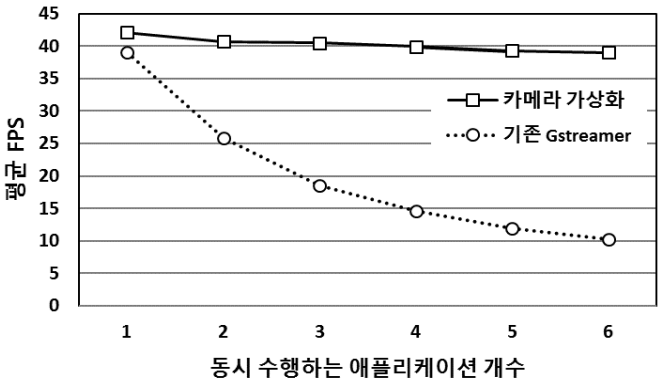


그림 4. 한 카메라를 다중 애플리케이션에서 사용 시 평균 FPS

5. 결 론

본 논문에서는 GStreamer 파이프라인 가상화를 제안한다. 이 기법은 기존의 GStreamer가 한 카메라를 다중 애플리케이션이 공유하지 못하는 문제를 해결한다. 또한, 가상화 데몬의 파이프라인이 설정 가능하기 때문에 사용자 시스템에 맞게 최적화하여 사용할 수 있다. 추가적으로 애플리케이션은 GStreamer에 대한 고려가 없어도 되므로 쉽게 프로그래밍을 할 수 있다.

향후 연구에서는 자바스크립트 API와 딥러닝을 지원하도록 구조를 발전시킬 것이다. 이 연구를 통해 다양한 장치에서 자바스크립트를 이용한 다양한 애플리케이션이 개발될 것으로 기대한다.

참고문헌

[1] GStreamer. https://gstreamer.freedesktop.org
[2] v4l2loopback.
https://github.com/umlaeute/v4l2loopback