Deliverable 4.4.1

Project Title	Next-Generation Hybrid Broadcast Broadband
Project Acronym	HBB-NEXT
Call Identifier	FP7-ICT-2011-7
Starting Date	01.10.2011
End Date	31.03.2014
Contract no.	287848
Deliverable no.	4.4.1
Deliverable Name	EVALUATION: Intermediate Middleware Software Components for Cloud Service Offloading
Work package	4
Nature	Report
Dissemination	Public
Author	Bin Cheng (NEC)
Contributors	Michael Probst (IRT)
Due Date	31.05.13
Actual Delivery Date	31.05.13

Table of Contents

Executive Summary

This document is the second deliverable for Task 4.3 in HBB-NEXT. In Task 4.3, HBB-NEXT explored the terminal functionalities to be offloaded to the cloud and designed and developed a cloud-based media processing platform (called media cloud) for handling them dynamically and efficiently. It is one of two Milestone 7 deliverables created by WP4 and thus presents Milestone 7 for the Cloud Offloading Enabler Part of WP4: "MS7: The intermediate version of the software components of WP3, 4 and 5 is in place".

Use case: the media cloud synchronizes two separate streams and delivers the synchronized stream to various terminals, for example, the video of a sign language interpreter can be played out along with the video of a TV channel both on mobile devices and HbbTV terminals.

Design: The media cloud is able to receive media content from broadcasters or other content providers via different sources, for example, from DVB signal receivers, from an IP network over HTTP, or from a cloud storage system. On the other hand, it first detects the capability of the end-user devices, in terms of device type, browser type, supported codecs, screen size, and also network bandwidth and latency, then accordingly launches appropriate processing components to generate adaptive streams suitable for the devices, allocates those tasks over different virtual machines in the cloud, and finally delivers the generated streams to the user devices over the network. Within the cloud, the entire media processing is decomposed into different media processing tasks, such as encoding separated streams with different codec or resolutions, overlaying subtitles, and multiplexing streams with different container formats. Later on, different media tasks are composed together to provide the required stream. By splitting up complex tasks, lots of media processing computation can be shared across users, devices, and applications.

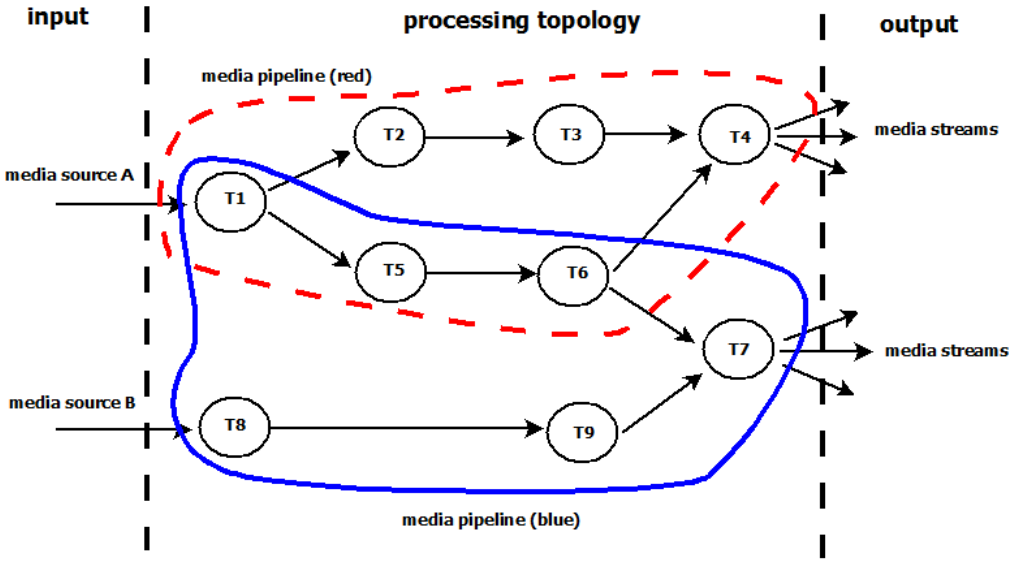


Figure 1: Concept design of media cloud

Prototype: The media cloud system is built on top of the OpenStack-based cloud infrastructure, which hosts media processing virtual machine instances. The media manager is the heart of the system. It instantiates the pipeline manager and almost every other component of the system and controls the underlying IaaS cloud service. The pipeline manager is the brain of the system, which makes important decisions regarding resource allocation and process distribution across the worker instances (virtual machine). All the media tasks are implemented based on the GStreamer framework and executed within virtual machines. Special virtual machine images with preinstalled media processing software (e.g., GStreamer library) and scripts (GStreamer-based python scripts) are created and stored in the IaaS cloud for this purpose; these images can be booted up on demand by the media manager, which enables the proper control and flexible operation of the media processing within virtual machine instances. The web front end is responsible for interacting with the web browser clients of the users. It also provides a controller console for launching, terminating and monitoring media channels in the cloud and the status of the processes deployed in the cloud.

Integration: The HBB-NEXT project is in the progress of integrating different components to enable a set of media applications on both TV sets and personal devices. The media cloud provides a UI-based interface for broadcasters to specify and manages channels on the back-end side. On the front-end side, it communicates with browsers and delivers generated streams to users.

Demonstration: An OpenStack-based cloud is set up in NEC's lab for the development and demonstration purpose. Four servers connected by a network router have been set up. Two of them are used computer nodes in the OpenStack-based media cloud, one is used as the controller node, and the remaining one is the admin node. The admin node is used as the client with the user credential to manage the media cloud. The following demonstration shows part of the functionalities that have been already implemented in media cloud.

Figure 2: Screenshots of media cloud

1. Introduction

1.1. Background and motivation

Hybrid Broadcast Broadband TV, called HbbTV in short, is closing the gap between traditional broadcasting-based TV networks and the Internet. HbbTV uses the broadcast infrastructure to deliver one-way high quality media content and also uses the two-way broadband to offer interactive and personalized secondary content and applications. Recently, mobile devices are dramatically growing in the market; we are entering a multi-device digital world where people expect to use mobile devices with TV more often in a consistent and seamless way. The use of multiple screens in peoples' everyday lives is becoming commonplace. According to a 2012 study by Google, 90% of consumers move between multiple devices – smartphones, tablets, personal/work computers, and smart televisions – to complete tasks throughout the day. Using multiple devices with TV, also known as companion screen or device, allows for new kind of services for TV consumers, for instance, they can get relevant information about the actors on their personal device while watching a movie on TV, or one can pick up his smartphone to listen to an additional audio track with a different language without affecting others in the same room. With lots of new user interaction technologies like touching, pairing, gesture and speech recognition, multi-device is redefining what can be done with multi-device in HbbTV.

Multi-device gives us a new chance to provide more personalized and more interactive media applications around TV. However, it also brings some new technical challenges to the HbbTV system. First, mobile devices usually have very limited computation, storage, and battery capability to do heavy media processing, such as media editing and high resolution content decoding. Second, there is a large heterogeneity among tablet PCs and smart phones regarding window sizes, screen resolution, and support for different audio/video codec and streaming protocols. Third, synchronization across multiple devices needs to be taken into account seriously in order to enable seamless user experience. To support inter-media or inter-device synchronization, a reliable and scalable back-end server is usually needed. All of these challenges can be tackled by offloading some terminal functionality from the end user side to the back-end side, which makes the development of interactive and personalized media applications in the HBBTV and multi-screen environment possible.

However, it is challenging for broadcasters and content providers to technically offload terminal functionality and further efficiently and economically manage the offloaded services because they usually do not have infrastructure and experience to run those services over the Internet for their users. Fortunately, cloud computing is ready to be used as an elastic and scalable infrastructure by providing scalable storage system and elastic computing capability for handling heavy and various media processing tasks. It leverages Internet networking technology and standard servers to create financial and infrastructure benefits for service providers and their customers. Financially, the cost to cloud service customers is paid only for what they use. Therefore, cloud-based service offloading can be a potential solution to sort out the issues for both broadcaster and end users.

1.2. Cloud-based media service offloading

Based on advanced virtualization technologies and the "pay-as-you-go" model, cloud computing is becoming a good choice to build up a media service platform for handling heavy and various media processing in a distributed, scalable, and economical way. In Work Task 4.3, HBB-NEXT explored the terminal functionalities to be offloaded to the cloud and designed and developed cloud-based media processing platform for handling them dynamically and efficiently.

As a service enabler, the HBB-NEXT media cloud is built on top of the open source cloud platform OpenStack. The media cloud is able to receive media content from broadcasters or other content providers via different sources, for example, from DVB signal receivers, from an IP network over HTTP, or from a cloud storage system. On the other hand, it first detects the capability of the end-user devices, in terms of device type, browser type, supported codecs, screen size, and also network bandwidth and latency, then accordingly launches appropriate processing components to generate adaptive streams suitable for the devices, allocates those tasks over different virtual machines in the cloud, and finally delivers the generated streams to the user devices over the network. Within the cloud, the entire media processing is decomposed into different media processing tasks, such as encoding separated streams with different codec or resolutions, overlaying subtitles, and multiplexing streams with different container formats.

Later on, different media tasks are composed together to provide the required stream. By splitting up complex tasks, lots of media processing computation can be shared across users, devices, and applications.

With the service portal of the HBB-NEXT media cloud, content providers and broadcasters can easily launch dynamic media pipelines for their media content to offer adaptive or synchronized media stream content to multiple devices without worrying about the heterogeneity of terminals or buying dedicated hardware. Alternatively, APIs are provided for application developers to build interactive applications associated with the delivered content. Therefore, service providers can take advantage of the two-way interactivity of the Internet to offer value-added media applications on TV or mobile devices, such as online polls, games, e-commerce, and online social sharing.

1.3. Major research activities and achievements

In the previous one and a half years of HBB-NEXT, lots of effort has been contributed to build up a cloud-based media processing platform for handling media processing. More specially, the research has been focused on the following aspects.

First, an experiment-driven analysis has been performed to understand the features of different types of media processing that can be offloaded from terminals to the cloud, and also the benefits/cost of handling those media processing tasks in the cloud.

Second, a cloud-based media processing service platform has been designed and implemented based on the GStreamer framework and the open source OpenStack cloud infrastructure. With this media cloud platform, media pipelines across virtual machines can be dynamically launched in the cloud for handling the offloaded media tasks and also maximizing the processing sharing opportunity across different applications, devices, and users.

Third, the solution to automatically detect the capability of devices has been developed and further applied into the media cloud platform for delivering adaptive media content to various end user devices.

Fourth, the interfaces of the HBB-NEXT media cloud have been designed for content providers to create, manage, and monitor the media pipelines associated with their content and their users. Some interactive media applications, such as TV-related social voting, have been further developed to demonstrate the capability of the designed interfaces.

1.4. Structure of the deliverable

The rest of this deliverable is organized as follows.

Section 2 first describes a brief architecture overview of the entire HBB-NEXT platform, including all underlying enablers and their interfaces with other enablers and applications. More details are presented to introduce the cloud service offloading enabler and its role in the entire project.

Section 3 presents the analysis results from the initial experiments that have been performed for understanding the features of different types of media processing in terms of CPU and bandwidth resource usage, requirement predictability, time-related fluctuation, and also for understanding the features of cloud infrastructure, its difference from the native hardware resource, and the benefit and penalty of handling media processing in the virtualized cloud environment.

Section 4 starts with the description of the basic approach to cloud service offloading, then presents the system architecture of the HBB-NEXT media cloud. Based on the presented architecture, this section describes how the building blocks of the media cloud are designed and implemented, including how device capability is automatically detected, how to dynamically launch and manage media pipelines across different virtual machines in the cloud, and how to monitor the entire media cloud platform to provide a media cloud analytics service.

Section 5 describes the interfaces of the media cloud for content providers to create media pipelines. These interfaces will be used by HBB-NEXT application providers to create channels so that users can fetch the generated adaptive streams from the media cloud via different terminals, such as TARA HbbTV box and mobile devices.

Section 6 presents the demonstration of the HBB-NEXT media cloud with a brief introduction to its setup, functionalities, applications, and screenshots.

Section 7 documents related work from different perspectives, such as cloud offloading and media cloud, and also explains a little bit how the HBB-NEXT media cloud differentiate from them and what goes beyond the state of the art.

Section 8 summaries the deliverable and provides an outlook to the future work plan.

2. Architecture Overview

This section first describes the architecture overview of the entire HBB-NEXT project and then points out the relationship between the cloud service offloading enabler and the other components. Later on, the major tasks of the HBB-NEXT media cloud are introduced with more details.

2.1. HBB-NEXT

HBB-NEXT ([1]-[4]) aims at creating an open web-based advanced middleware framework for seamless integration of broadcast and Internet content, further enabling advanced media applications for geographically distributed groups of users to interact with each other and consume content from both Internet and broadcast sources on heterogeneous end-devices. In order to achieve this, HBB-NEXT has been put its research and development efforts on enhancing current state of the art in the following areas: identity management and trust, multimedia content synchronization, cloud media computing, multi-user and multi-modal personalization, and context aware value added services.

One of the major objectives of HBB-NEXT is to define an open framework for the next generation of hybrid broadcast internet (HBI or HBB) services. On top of deployed standards like HbbTV, HBB-NEXT defines a set of extensions and open APIs to enable more advanced services and business cases. Figure 3 shows the system architecture of the HBB-NEXT platform as developed in WP6. The diagram consists of multiple enablers provided by WP3/4/5, which either reside on the internet (cloud) or on the terminal, and public interfaces to the HBB-NEXT applications (APPS). The interfaces shown in the diagram can be distinguished into APIs and protocols. APIs of internet based enablers are potentially exposed by one or multiple service provider and named A1, A2 etc. The API that is exposed to applications on the terminal is named T1. HBB-NEXT also defines open protocols for the advanced synchronization features. These protocol interfaces are named as P1 and P2. APIs between enablers are labelled by the name of the sub module that exposes it, for example, MMI, IDM and SM. The following is a brief description of what each component is responsible for.

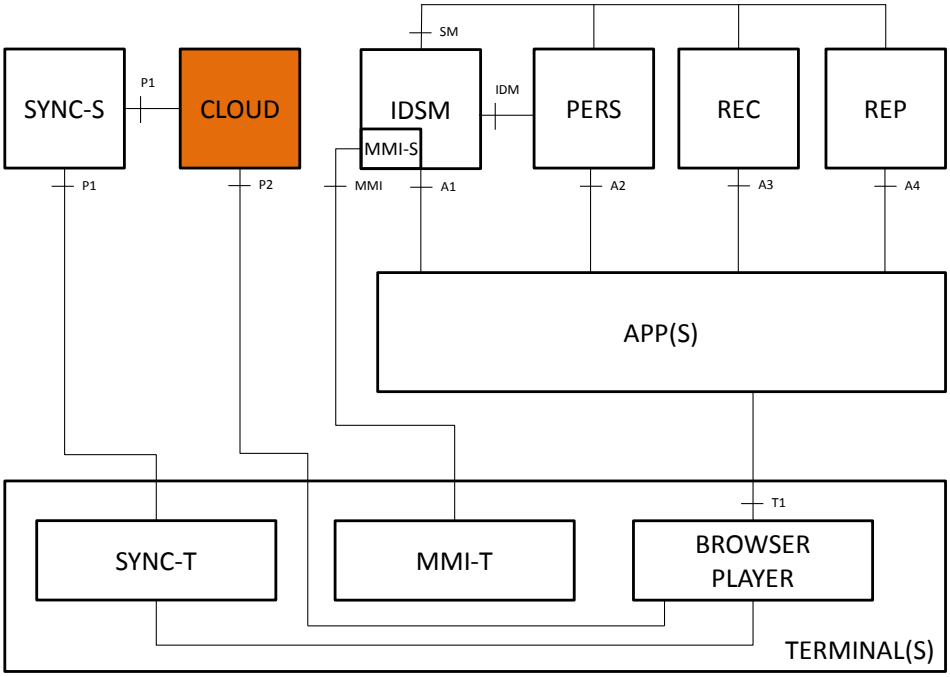


Figure 3: Architecture of the HBB-NEXT platform

REP *(reputation management enabler):* managing the reputation of applications to ensure the security of the third-party applications.

REC *(recommendation enabler):* enabling context-aware recommendation.

PERS *(personalization module):* providing personalized media content to user(s).

IDSM *(identity and security management enabler):* managing user identification and user profile and also providing interfaces to authenticate users and devices and sharing user profile information to the recommendation engine to analyse users.

CLOUD *(cloud offloading enabler):* offloading some media processing tasks from terminals to the cloud while providing streaming content to different types of devices.

SYNC-S *(server-side synchronization):* synchronizing the source content before sending out them to the cloud or the terminals.

SYNC-T *(terminal-side synchronization):* being responsible for inter-stream synchronization and inter-device synchronization on the end user side.

MMI-T *(terminal-side multi-modal interface):* enabling different methods for users to interact with the system, for example, eye tracking or gesture recognition.

APPS *(application layers):* managing all applications which are working on top of the HBB-NEXT platform and utilize the functionalities from the underlying enablers.

BROWSER-PLAYER *(HbbTV-supported browser):* providing the browser environment for all applications with an enhanced media player.

2.2. Cloud offloading

Usually advanced media services must simultaneously render and display Audio/Video content on a single end device, while the processing power in typical HBB end user devices is still limited. To overcome resource limitations of terminals, e.g., TV, STB, and mobile devices, WP4 will realise mechanisms for distribution / offloading of application functionalities, for example, video rendering services or remote transcoding services, incorporating cloud computing solutions to enable low performance terminals to serve all HBB-NEXT features.

The work is focusing on how processing limitations within the end device can be overcome by offloading application and video processing to powerful computing systems in the cloud. Based on the HBB-NEXT usage scenarios and user cases described in D2.1 [2] and D2.2 [3] and the outcome of the state of the art analysis in D4.1 [1], cloud-based software components, will be designed and implemented that allow dynamic offloading of processing and rendering to cloud systems on demand. Offloading is based upon the capabilities of the terminal and the requirements of the service. This processing offloading will happen in a way that is transparent for both service provider and user. Existing cloud products (e.g. Eucalyptus) and/or providers (e.g. Amazon EC2, Google AppEngine) will be integrated with the system and “hidden” by an adaptation layer which abstracts from the peculiarities of the underlying platform.

The adaptation layer preserves the scalability of the cloud and makes it accessible to service provisioning. Software components appropriate for being deployed on terminals as well as on the cloud will be developed in D4.4.2 [15] and D4.6.2 [16]. These components will be utilized in the prototype developed in WP6 in order to demonstrate the capabilities of dynamic transparent cloud offloading.

3. Analysis of Cloud-based Media Processing

3.1. Media processing to be offloaded

Regarding enabling cloud offloading, the first question is to investigate what can be offloaded. For content providers/broadcaster, the same media content like images and audios/videos need to be rendered on different type of terminals in their services. Usually, in order to provide proper media content adaptive to terminals, they have to do certain processing for their content. This type of processing usually requires dedicated high-end hardware with a huge amount of buying cost and maintenance cost and is better to be offloaded into the cloud. For application developers, they might require an elastic platform to handle media editing to provide some personalized or interactive media applications, for example, enabling dynamic video ad insertion. For end users, they generate lots of media content in their daily life and would like to save and edit them. More specifically, in the HbbTV environment the following media-related processing tasks have been identified to be offloaded from terminals to the cloud.

Content fetching: there are different types of sources to fetch/receive media content, for example, receiving MPEG transport streams via a DVB-T receiver or fetching adaptive streaming through different streaming protocols like RTSP/RTP, RTMP, WebM. Putting the content fetching module into the cloud can overcome the limitation of mobile devices to make broadcasting content available over the IP network.

Media transforming: this mainly means image resizing and video transcoding. Given a media source (image, audio, or video), the media cloud first decodes it into the raw data and then converts it into the required format of the end user device, with regard to resolution, size, frame-rate, codec.

Stream multiplexing: the media cloud can dynamically multiplex the streams with different codec and resolution and then pack them up using different streaming container formats which can be suitable for different devices/browsers, for example, Apple HLS, WebM, MPEG transport stream over HTTP.

Inter-media synchronization: Given a set of input media sources (main video, second video, audio, and subtitle stream), the media cloud is able to synchronize them and combine them together to deliver a combined stream. By offloading inter-media synchronization into the cloud, the picture-in-picture functionality on tablets can be achieved.

Video editing: video editing allows third-party applications to change video streams, which usually requires lots of computation and cannot be done on normal terminal devices. By offloading editing into the cloud, the media cloud can easily enable dynamic video ad insertion and provide interfaces for application developers to implement some interactive media applications with TV and second device, such as social voting.

3.2. Understanding media processing

Different from other data intensive processing, media processing has its own unique characteristics. First, media processing has a wide range of computation and network consumption. For example, content with different resolutions need completely different CPU consumption for encoding, different bit rate streams lead to largely different bandwidth usage for content delivery. Second, media processing has very strict time constraint. Quality of streams could be seriously affected if some part of content is late to be processed. Third, unlike text-based content, media content adaptation also requires lots of processing in order to adapt to different devices, networks, or browsers.

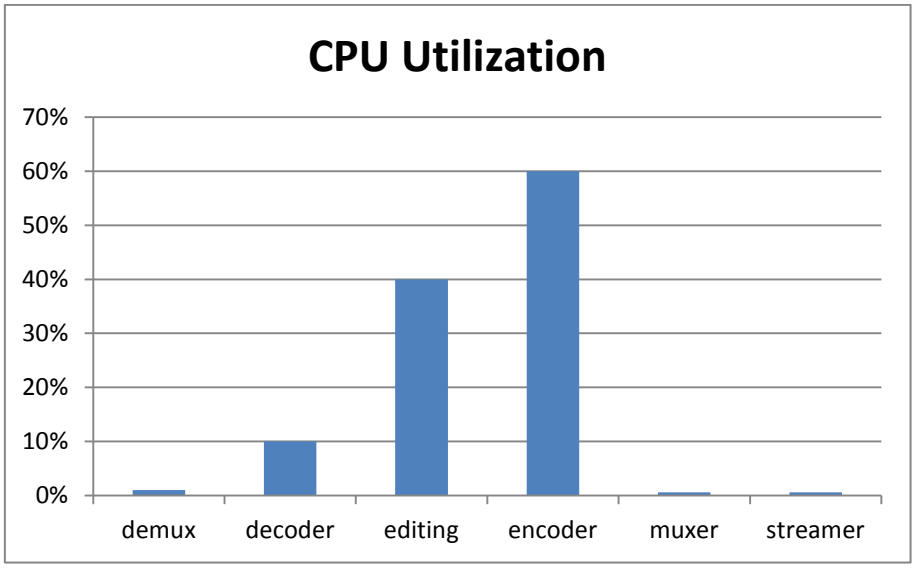


Figure 4: avg. CPU usage of different tasks within the same pipeline

Computation consumption

To understand the CPU consumption of media processing that can be offloaded, some initial experiments have been performed. A typical media pipeline usually consists of the following tasks: demuxer, decoder, editing, encoder, muxer, and streamer. Figure 4 shows how much CPU computation is used by each of these tasks when a media pipeline is launched. The result shows that tasks are largely different in terms of their CPU utilization. It also indicates that this diversity has to be taken into account for task allocation algorithm in the cloud.

An investigation is done to look at how CPU utilization is changing over time for a single task. Figure 5 shows the distribution of CPU utilization of a decoder task. In the figure, x-axis means the range of CPU utilization, from the low range to the high range; y-axis is the probability of each CPU utilization range. As seen in Figure 5, for most of the time, the CPU utilization is close to 36% and the distribution reasonably follows the normal distribution. This result indicates, although there is a certain fluctuation of CPU utilization for media processing tasks, but the fluctuation turns to be reasonably predictable.

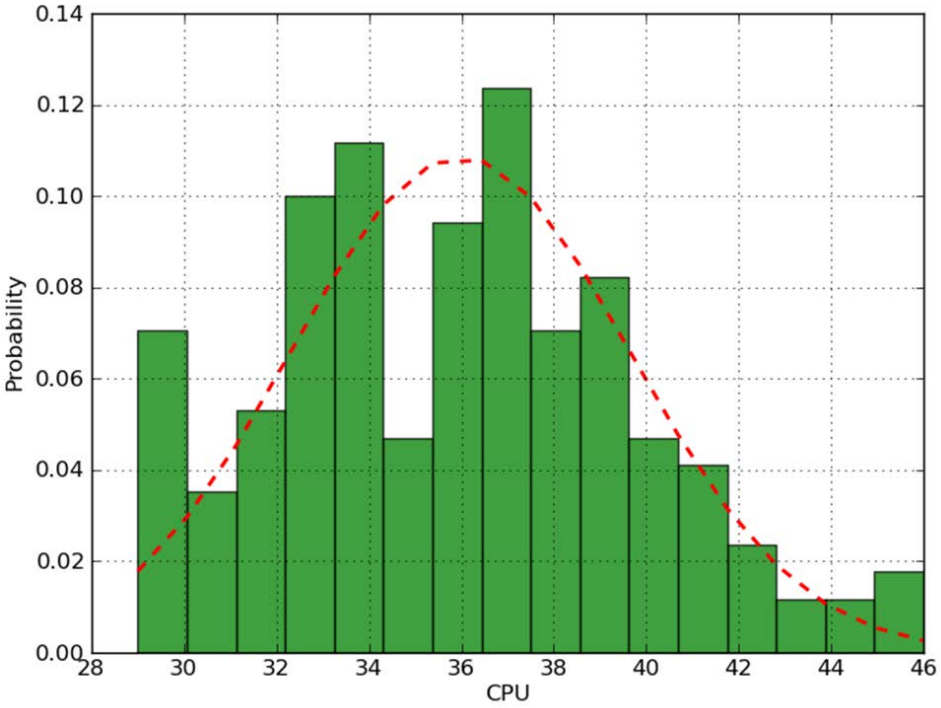


Figure 5: distribution of CPU utilization for a running decoder

Further investigation shows that codec, resolution, and stream rate also affect the required CPU utilization for media tasks, especially for those CPU-intensive tasks like encoding, editing, and decoding. For instance, high-definition (HD) content requires much more CPU utilization for encoding than standard-definition (SD) content, as shown in Figure 6. Based on our experience with HD content, a medium VM instance with two virtual CPUs is not even suitable for handling a single HD content encoding task. In addition, CPU utilization is strongly correlated to quality of streams. For example, when CPU capacity does not meet the requirement of an encoding task, it will slow down the frame rate of the generated stream.

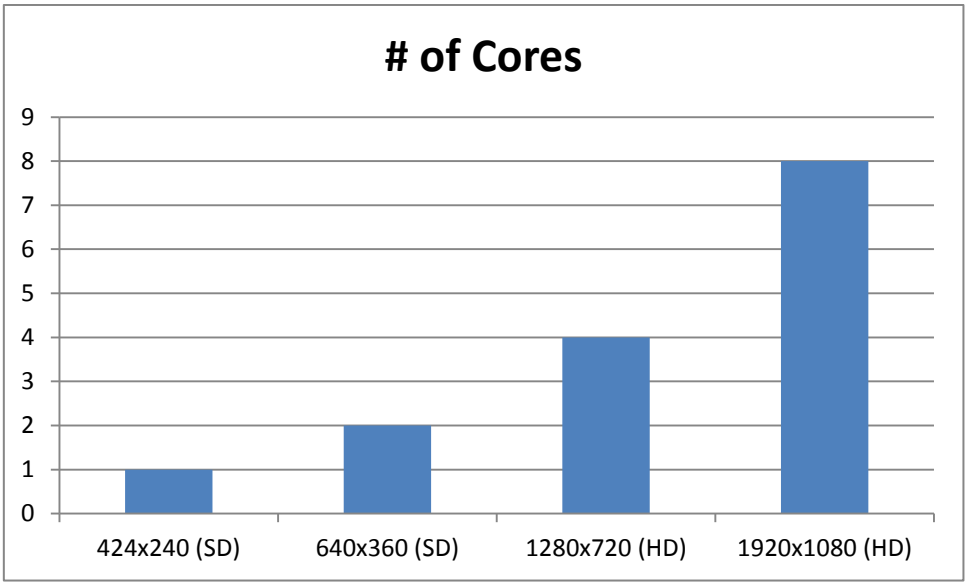


Figure 6: the number of required CPU cores for h264 encoding with different resolutions

Bandwidth consumption

Another practical concern is the bandwidth consumption when data is transferred from one task to another task in the same media pipeline. Usually, a compressed data stream rate is much lower than its raw data stream rate and the codec determines the compression ratio. For example, the compression ratio for H264 is about 7%. That means, for a compressed H264 video stream with 3Mbps, the raw data stream data will be around 44Mbps. Therefore, when the media tasks are allocated across different virtual machines, the transmission costs between tasks need to be considered seriously. Especially, for decoder and encoder, it is better to allocate them on the same virtual machine so that lots of network traffic can be reduced across virtual machines or physical nodes in the cloud.

3.3. Understanding cloud infrastructure

Based on advanced virtualization technologies and "pay-as-you-go" model, Cloud IaaS gives us a new chance to build and deploy large scale distributed systems in a fast, scalable, and economical way. As one of the most data-intensive and computation-intensive applications, media processing is a perfect example to leverage the power of cloud computing. However, as compared to dedicated hardware resource, the cloud-based infrastructure has its own features.

First, resources in the cloud are shared with multiple tenants, which can cause some interference. Tenants are isolated resource containers within the cloud, including a separate VLAN, volumes, instances, images, keys, and users. However, the underlying hardware and network are still shared. So-called hypervisors provide mechanisms to support the isolation and fairness between tenants, while to some extent the interference between different tenants still exists, especially the interference from the network layer. Moreover, regarding a heterogeneous cloud with different types of hardware settings, performance interference is more serious. According to a recent measurement study on EC2 [5], Amazon EC2 uses diversified hardware to host the same type of instance. The hardware diversity results in more performance variation. For example, by selecting fast instances within the same instance type, Amazon EC2 users can acquire up to 30% of cost saving if the fast instances have a relatively low probability. Therefore, media processing tasks cloud be affected by the co-locating other VMs running different types of workloads which belong to other users in the same cloud.

Second, cloud infrastructure provides a higher degree of transparency and elasticity. Using cloud IaaS, users get less control and knowledge on the underlying network than with a dedicated server-farm setup. For example, there is not direct way to know the network locality between VMs in the cloud so that locality-aware task scheduling might be problematic. Also, usually network-layer multicast is not supported in public cloud so that sophisticated bandwidth optimization algorithms need to be considered at the application layer.

Third, the cost in the cloud is more dynamic. Different from dedicated server-farm setup, the costs in the cloud directly correlated to the resource usage, in terms of type and number of virtual machine instances, their running time, used bandwidth. Therefore, designing efficient resource usage strategies and playing with the pricing rules of cloud infrastructure providers can help to save the total cost.

3.4. Summary

This section first discusses the tasks that can be offloaded to the cloud and then lists some particular media processing tasks suitable for cloud offloading. Later on, it presents an experimental analysis on the feature of those media processing in terms of their resource usage and discusses the features of cloud infrastructure. These understandings and observations reason about the architecture design and strategies of the HBB-NEXT media cloud for enabling cloud-based service offloading service.

4. Cloud Offloading for Media Processing

4.1. Basic approach

Based on the idea of moving more media processing to the cloud, a cloud-based media processing platform, called MediaCloud, is designed to allow media application developers to economically and efficiently utilize cloud infrastructure for developing next generation TV applications. With MediaCloud, media application developers can quickly launch their services to support a large number of users from the beginning, without having a big initial investment for buying the required hardware to start their own business.

MediaCloud is built on top of the open-source cloud infrastructure (IaaS), called OpenStack [6]. This gives MediaCloud elastic cloud computing and cloud storage capabilities. On the media production side, cloud-based media servers can provide sophisticated media processing capabilities. On the consumption side, the set-top-box functionality can be reproduced in the cloud, along with the added advantage of sharing computation and processing between identical user requests and reducing the per-user computational overhead. The provisioning of cloud resources can be closely tied to the user demand, thereby reducing the operating expenses of the MediaCloud infrastructure. More importantly, much of the common processing tasks can be shared in the cloud. For example, one user may consume live content with English subtitles and/or audio line whereas another user may request the same content with Japanese subtitles and/or audio line. In this case MediaCloud can share the video processing of the two outputs, add different audio lines and/or subtitles for each user and might even perform live translations of (e.g. English) subtitles in other languages.

In our media cloud, the following important concepts are introduced to enable media processing in the MediaCloud.

Media Task: a media task performs an operation on media content. For instance, in order to deliver adaptive video content to different browsers or end user devices, media content needs to be encoded with suitable codecs and resolutions. Here video encoding is a media task. Each media task type has a certain computing profile that specifies when and how long the task uses computing resources, how much network bandwidth and memory it requires, and how much delay it introduces.

Media Pipeline: a media pipeline consists of several media tasks which are logically connected to finish a certain media processing and can be executed in parallel or sequentially on demand. Within one cloud, there can be several media pipelines running in parallel. Therefore, media tasks as described above can be seen as partial media pipelines.

Media Source: each content input represents a single media source for the media cloud. It could be a media file from local file systems or cloud storage, or a media stream delivered over Internet.

Media Stream: each media stream represents a streaming output from the media cloud to a device or a user. Media streams can be delivered over different streaming protocols with different container formats, for example, Apple HTTP live streaming, RMTP streaming, UDP streaming, MPEG transport stream over HTTP, or MEPG DASH stream over HTTP.

Figure 7 illustrates the relationship among these concepts. Basically, media sources and media streams represent inputs and outputs of the media cloud respectively. Media streams and media sources are connected through a series of media tasks. A media pipeline takes some media sources into the cloud and then generates a certain type of media streams through a set of media tasks. Media tasks are shared across media pipelines. Media source can be also shared between media pipelines, but media synchronization has to be ensured.

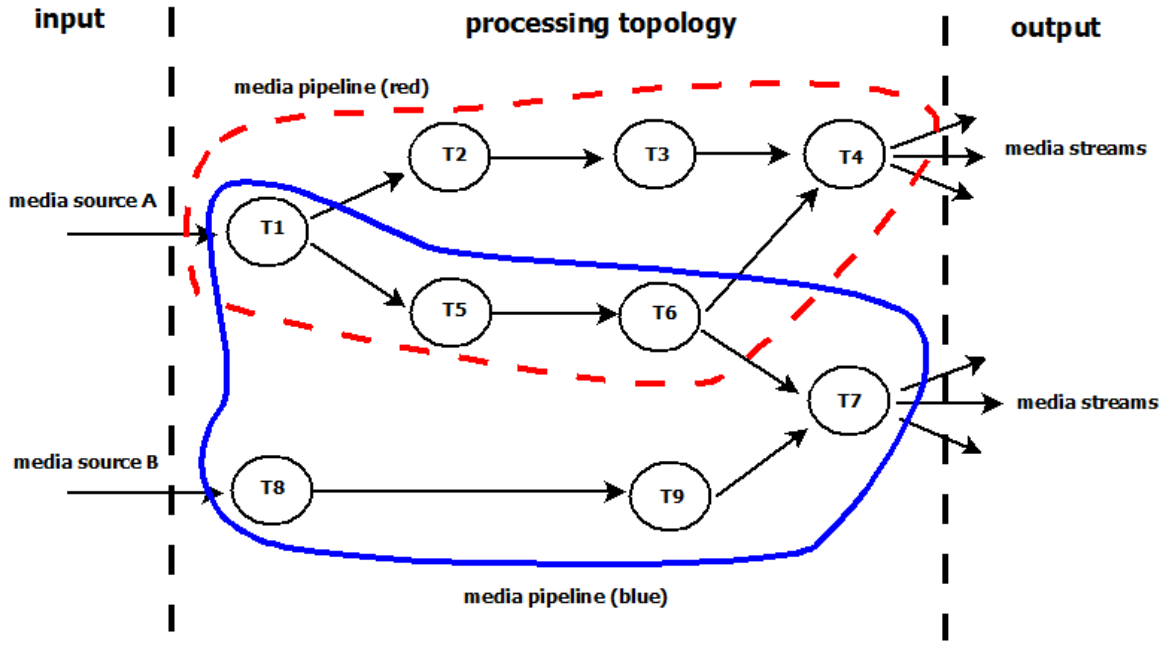


Figure 7: relationship of channels, pipelines, tasks, and streams

4.2. System architecture

As shown in Figure 8, the whole media cloud system consists of the following components: *content sources, media manager, pipeline manager, web front-end, and the underlying virtual machines.*

Content Sources: providing media content as a file or as a stream into media pipelines for further processing.

Virtual Machines: representing the underlying computing resources. For the remainder of this paper, we assume that all virtual machines are of the same type, i.e. all virtual machine have the same computing power and memory. However, the concepts applied here can also be applied to heterogonous resources. Media tasks are executed within one virtual machine. The tasks in a single media pipeline can be distributed over several virtual machines. Hence, a communication framework is established for communication of media tasks located at different virtual machines.

Media Manager: being responsible for coordinating the assignment of media tasks within media pipelines to virtual machines.

Pipeline Manager: being responsible for matching requests for content against running pipelines to foster their reuse. The pipeline manager is the key interface between the media-aware platform cloud and the infrastructure level IaaS cloud. The pipeline manager handles the entire lifecycle of media processes running in the cloud.

Web Front-End: delivering web pages with the media content to the users' browser. We assume that the user has a standard HTML5 capable browser, such as Mozilla Firefox, Google Chrome, Internet Explorer or Apple Safari. As part of these web pages, some scripts are executed within their browser and they detect capabilities of the device and they submit requests for media content to the web front end which dispatches them to the media manager.

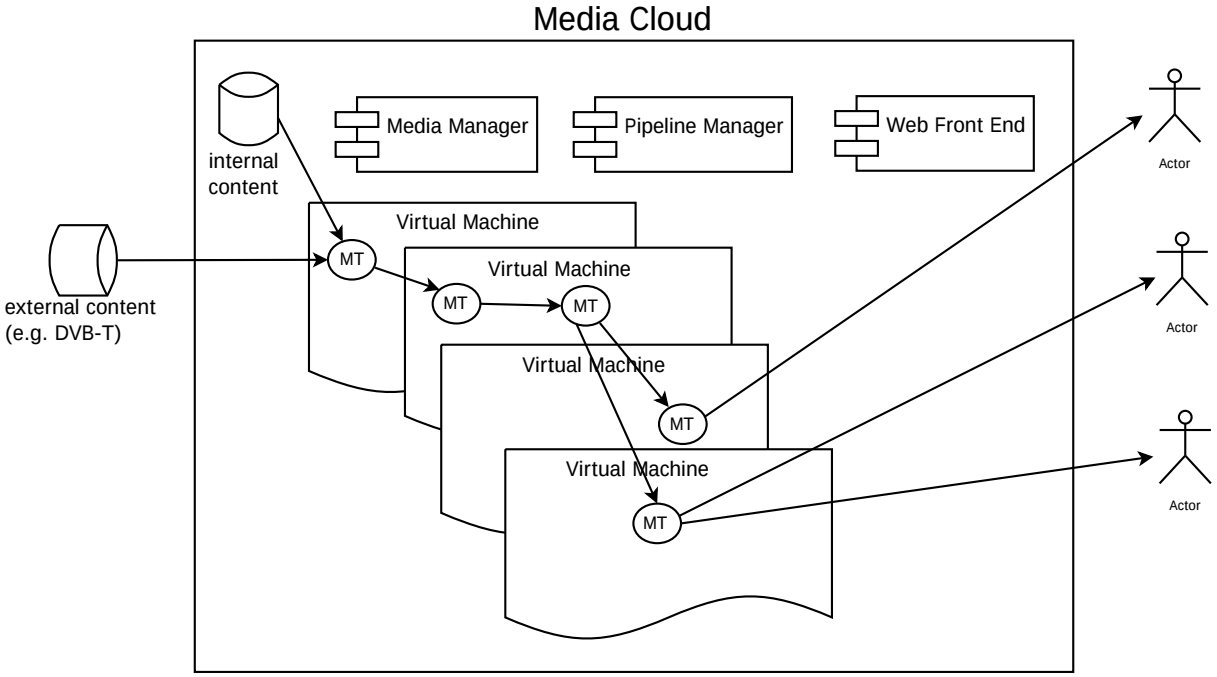


Figure 8: system architecture of media cloud

The entire MediaCloud runs over the OpenStack-based cloud infrastructure [6], which hosts media processing virtual machine instances. The media manager is the heart of the system. It instantiates the pipeline manager and almost every other component of the system and controls the underlying IaaS cloud service. The pipeline manager is the brain of the system, which makes important decisions regarding resource allocation and process distribution across the worker instances (virtual machine). All the media tasks are implemented based on the GStreamer framework [7] and executed within virtual machines. Special virtual machine images with preinstalled media processing software (e.g., GStreamer library) and

scripts (GStreamer-based python scripts) are created and stored in the IaaS cloud for this purpose; these images can be booted up on demand by the media manager, which enables the proper control and flexible operation of the media processing within virtual machine instances. The web front end is responsible for interacting with the web browser clients of the users. It also provides a controller console for launching, terminating and monitoring media channels in the cloud and the status of the processes deployed in the cloud.

4.3. Communications between different components

The MediaCloud platform is a highly distributed system on top of the OpenStack cloud. One of the big challenges to build such a distributed system is to design and implement the communication mechanisms for all different components in the system to work together efficiently and reliably. The following communication ways have been implemented in the media cloud for different purposes of usages.

The key communication framework is the AMQP (Asynchronous Messaging Queue Protocol) used by most of the components to communicate with each other. In effect, every component instance (except the VMController Daemon which is managed by media manager and communicates with all launched virtual machines via SSH) creates an AMQP client and links it to the central AMQP server. Then, when other component instances want to send messages to the component they simply send the messages to this queue. The receiving component is then free to pop messages from this queue and process them. The particular library used for message exchanging between components is a python-based RabbitMQ [13] library, called pika. The advantage of using RabittMQ is that the RabbitMQ broker server existing in the OpenStack setup can be shared.

In addition to AMQP, a few other protocols are also used at the periphery of the system. The SSH-based communication is used between the pipeline manager and the SSH servers on all launched virtual machines for transferring executable python scripts to virtual machines and then launching media processing tasks. The VMController Daemon sends UDP packets with the resource usage information from each launched virtual machine to the analytics server. RPC is used for intra-VM communication between the GStreamer processes (producers, consumers, splitters) and the VMController daemon. In addition, multicast is used between different tasks across different virtual machines for data

transmission with regard to the fact some tasks share the same data streams generated from the same previous tasks (this cloud be seen in Section 4.5). The web front-end application serving the media to the web-browser as well as serving the monitoring and system control views to the web browser make use of a combination of HTTP and webSocket protocols for dynamic real-time application execution.

4.4. Device capability detection

The assumption is that end users access the media cloud from a browser on the terminal device. In order to generate appropriate streams suitable for the user's device, a Javascript-based module is designed to detect device capability in terms of the following factors.

The type of device: desktop, tablet, smartphone, HbbTV,

The screen size of device: this information can be retrieved from the width and height properties of "screen" or "window" objects.

The type of browsers: Safari, Chrome, Firefox, Internet Explorer,Opera

HTML5 Features: whether HTML5 video is supported

This device detector module is implemented based on a third-party library, called *Modernizr*, which is a small JavaScript library that detects the availability of native implementations for next-generation web technologies, i.e. features that stem from the HTML5 and CSS3 specifications. Many of these features are already implemented in at least one major browser (most of them in two or more), and what Modernizr does is, very simply, tell you whether the current browser has this feature natively implemented or not. Different from the traditional "user-agent" sniffing method, Modernizr aims to have a more reliable mechanic to determine the capabilities of a web browser. It tests for over 40 next-generation features.

On the back-end server side, the web front-end server receives this information and determines what are the proper resolution, codec, and multiplexer for this client by searching for a pre-defined device profile in a database, and then starts to launch the appropriate media tasks in the cloud, finally fits the generated content to the corresponding client when receiving its request. If the request comes before the generated content is ready, the request will be hung up for a while and the user will encounter a waiting time.

Therefore, to reduce this waiting time, the media cloud also launches some processing tasks for well-known devices in advance.

4.5. Media pipeline composition

The chosen approach in processing media streams is to deal with multimedia files in a single media manner. As content consists of (multiple) audio and (multiple) video streams, accompanied by (multiple) subtitle tracks splitting them in independent streams offers the possibility to present various combinations serving many requests. Distributed processing allows duplicating some of the operations with slightly differing configurations, e.g. transcoding HD video to smartphone resolution and tablet resolution. As shown in Figure 9, by reusing existing audio outputs with these different video outputs, many devices can be served with tailor-made content. Likewise similar devices requesting identical content in different languages can be served by reusing/sharing the same video output. In this way, lots of media processing can be shared across different users and devices.

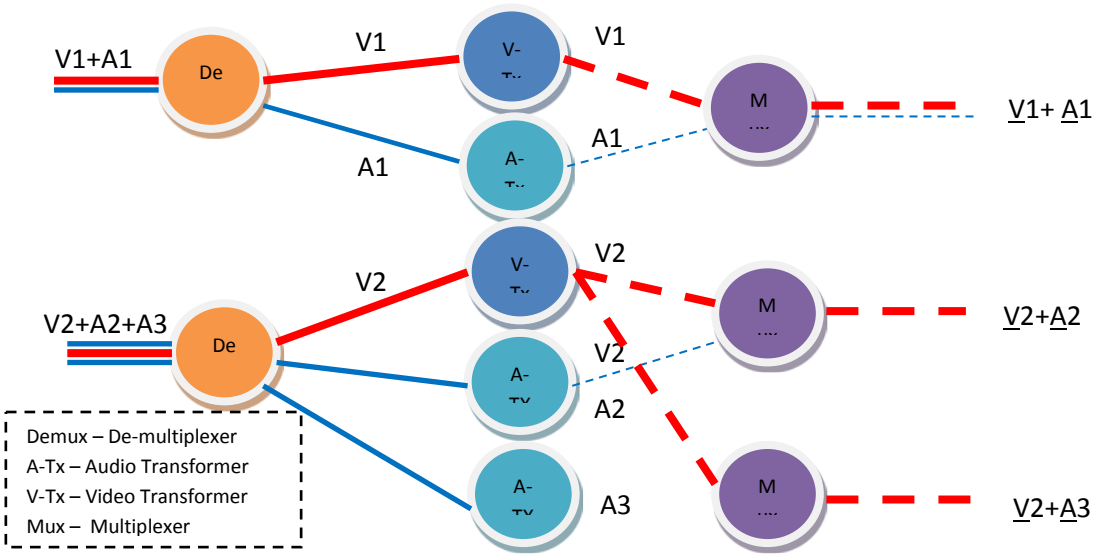


Figure 9: decomposing and recomposing of media processing in the cloud

In the media cloud, there are three types of media tasks commonly used to do media pipeline composition.

Splitter: The demultiplexer task is introduced to maintain synchronicity and dependencies of video and related audio. Each element represents one channel and is spawned with a given file or multimedia source (such as Webcam streams or DVB streams) as parameter.

The input is de-multiplexed, announced to the system and offered as independent elementary streams.

Producer: The transcoder task is decoding, processing (editing) and re-encoding given audio or video content in a new format/codec. The new format is chosen according to the destination browser and its HTML5 video tag capabilities. This type of tasks is the most CPU-intensive one because lots of media processing could be involved in a transcoder task. The system keeps track of all offered channels / contents and all client requests. If a client's choice is already covered by an existing producer, its output will be re-used. If no client requests the output of a certain producer it should be shut down.

Consumer: The multiplexer task multiplexes two incoming streams (one audio and one video) to a valid multimedia stream. This represents the choice of one client browser connected to the cloud. Switching a channel or selecting one of the offered video or audio lines inside a channel is reflected in the cloud as reconfiguring this task to listen to messages from another transcoder task. Multiplexer tasks are spawned in advance and assigned to one specific client upon connection to the Media Cloud system.

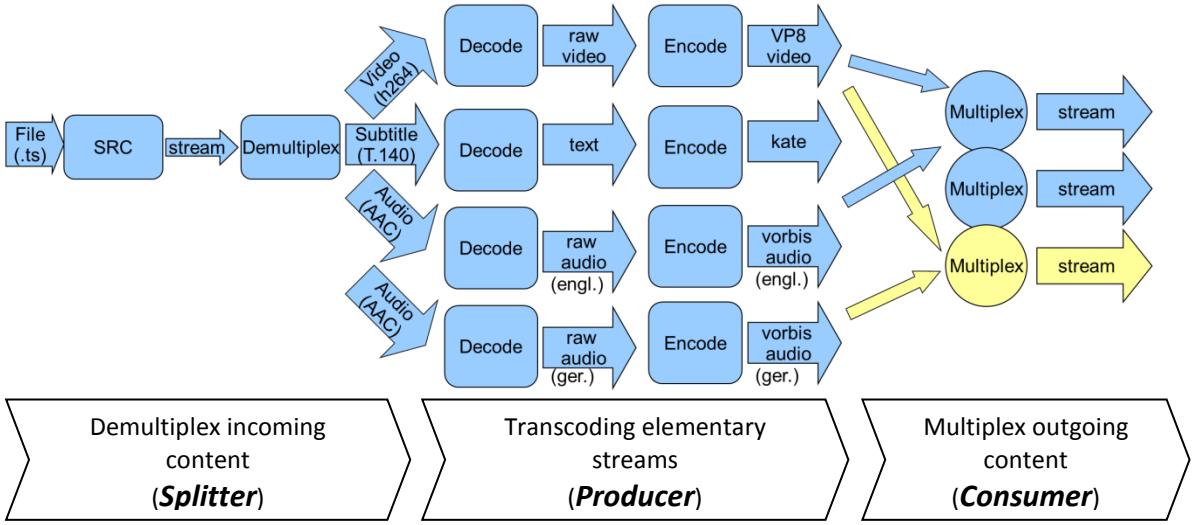


Figure 10: media processing pipelines in the media cloud

The information about client browser capabilities (accepted streaming formats depending on vendor and version) are stored within the scripts configuration and cannot be changed afterwards. A client disconnecting from the cloud triggers the shutdown of its associated multiplexer task. Thus, this element can be seen as the representation of one client entity inside the cloud.

As Figure 10 shows, splitter, producer, and consumer represent different stages of media processing, from getting source content into the cloud to sending out the generated stream out from the cloud. In the following the basic workflows for the splitter, producer, and consumer components are presented in detail.

The workflow of splitter

Given the URL of the media source, the splitter will dynamically construct a task chain to split up into different streams (like audio, video, subtitle) step by step, where a subsequent task is determined by the output of its previous task. As shown in Figure 11, if the given media source is a http stream, a “souphttpsrc” GStreamer element will be first used to fetch media content from this http stream and connected by a “typefind” element, which has a “have-type” callback function. When the type of the input media source has been identified by the “typefind” element, the “have-type” callback function will be triggered to accordingly create the demux element (for example, qtdemux for Apple QuickTime, hlsdemux for Apple HLS, or tsdemux for MPEG transport streams) and link it to the “typefind” element. After that, the created demux element starts to work and extract the embedded streams in the media source.

Whenever a new stream is extracted, its “pad-add” callback function will be called to generate corresponding stream. To determine the stream type, another “typefind” is created and linked to the demux. Its “have-type” callback function will be triggered to report the generated pad information to the media manager and the pad information will be later used by a producer. Finally, an “udpsink” element is created to send out the generated stream through a multicast address so it is easy to be received by multiple producer tasks.

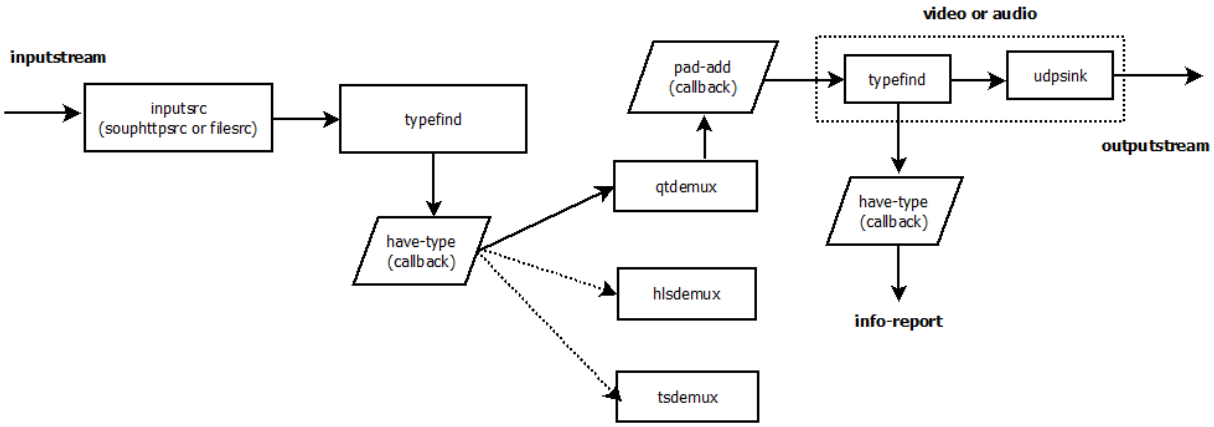


Figure 11: workflow diagram of a splitter

The workflow of producer

The producer is a heavy processing job, including decoding, editing, and encoding. This way the transmission of raw data can be localized within a single virtual machine in the MediaCloud and therefore lots of cross-VMs/hosts network traffic can be largely reduced. As Figure 12 shows, the workflow of a producer has the following steps. First, using the pad information extracted from the splitter, an “udpsrc” element is first created for handling the incoming stream and forwards the stream to the “decodebin” element. Then, the “decodebin” element will automatically decode the stream choosing appropriate decoder and generate the raw data. After that, some optional editing can be performed over the raw data and the proper codec to encode the raw data stream is chosen. The scripts of the editing and encoding tasks have been predefined in Table 1. Finally, the encoded data stream will be sent out to the next task via an “udpsink” element.

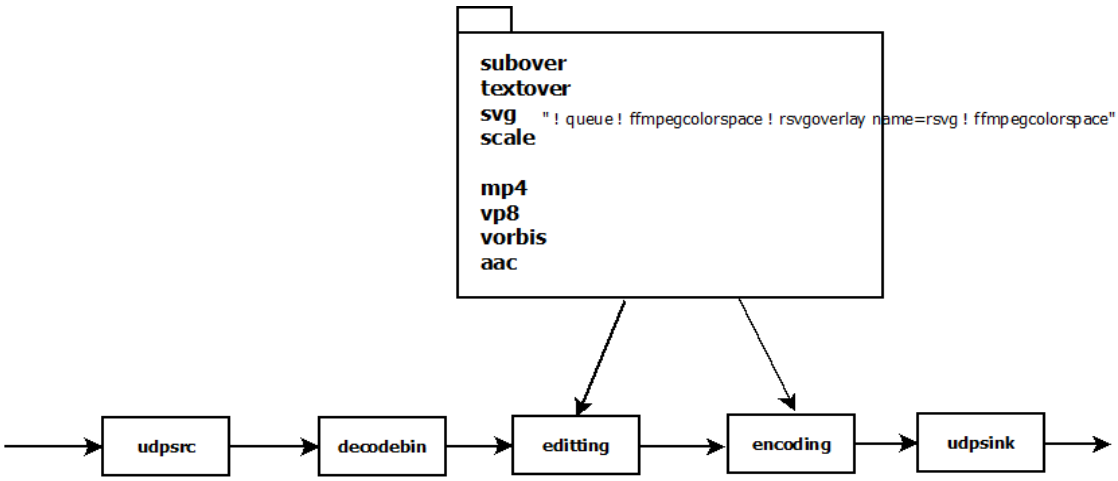


Figure 12: workflow diagram of a producer

Type	scripts
subsover	" ! queue ! subtitleoverlay name=subsover"
textover	" ! queue ! textoverlay name=textover"
svg	" ! queue ! ffmpegcolorspace ! rsvgoverlay name=rsvg ! ffmpegcolorspace"
scale	" ! videoscale add-borders=true method=1 ! video/x-raw-yuv, "
mp4	" ! queue ! x264enc bframes=0 cabac=false profile=1 byte-stream=false name=enc ! queue ! typefind name=typefinder ! udpsink name=sink"
vp8	" ! queue ! vp8enc threads=2 max-keyframe-distance=1 name=enc ! queue ! typefind name=typefinder ! udpsink name=sink"
vobis	" ! audioconvert ! audio/x-raw-float, channels=2 ! audiorate tolerance=200000 ! queue leaky=downstream ! vorbisenc ! queue ! rtpgstpay name=enc ! typefind name=typefinder ! udpsink name=sink"
mp3	" ! audioconvert ! audio/x-raw-int, channels=2 ! queue ! lame ! typefind name=typefinder ! udpsink name=sink"
aac	" ! audioconvert ! audio/x-raw-int, channels=2 ! queue ! ffenc_aac ! typefind name=typefinder ! udpsink name=sink"

Table 1: gstreamer scripts for different tasks

Workflow of consumer

The workflow of consumers is relatively simple. As Figure 13 shows, a consumer usually takes several input streams generated by different producers, multiplexes them together with a certain container format, and then delivers the combined stream to lots of clients as MPEG transport stream or WebM over HTTP. The consumer is a proper place for inter-stream synchronization in the cloud because the time difference between streams can be accordingly adjusted before sending the combined stream out. Different muxers are used to generate different type of streams for different browsers/devices. For example, webmmux can be used for both Chrome and Firefox and mpegtsmux can be used for HbbTV terminals.

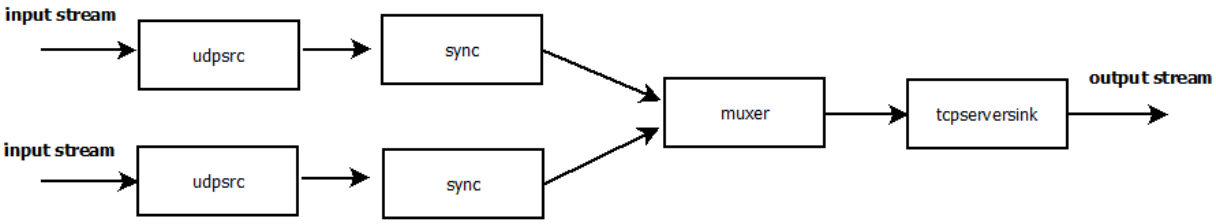


Figure 13: workflow diagram of a consumer

In terms of data transmission from splitter to producer and from producer to consumer, "udpsink" and "udpsrc" are connected by UDP multicast. This is because, as compared to unicast, multicast can avoid some redundant network traffic when multiple consumers/producers access the same output. The side effect of this design choice is under investigation, because some packet lost could be caused by heavy network traffic or network congestion. Therefore, more sophisticated task allocation algorithms are needed with regard to the bandwidth consumption across virtual machines and hosts.

Using these tasks provided by MediaCloud, new media pipelines can be composed dynamically, according to application requirements. The MediaCloud platform provides application developers with tools to compose and personalize media content using easy-to-deploy software applications. In addition, the MediaCloud platform automatically extracts common processing functionality across users and applications to reduce redundant processing in the cloud. The finalized content is streamed to different users on different devices by a dedicated multiplexer task through unicast or multicast networks (depending on the underlying transport technology of the network provider and the extent of personalization in the content application).

The pipeline manager takes care of the latest statue information of all running pipelines and tasks in the media cloud; it also manages the mapping from all pre-defined pipelines to their associated tasks. On receiving a new client request for a certain pipeline, the pipeline manager will first check whether the requested pipeline has been spawn. If there is a running pipeline that matches the requirements, the same pipeline will be reused to serve the request through a new media stream.

If none of the active pipelines matches the client request, the pipeline manager will look up the pre-defined pipeline list to find out the one suitable for this new client and then try to launch the media tasks associated with this new pipeline. If some task has been created in the cloud and is needed by this new pipeline, the pipeline manager will try to reuse it. This way, the opportunity of sharing media pipelines and media tasks across different clients/devices can be maximized and the resource usage of the media cloud can be minimized.

4.6. Inter-media synchronization

The previous subsection presents the way of creating a media pipeline based on a set of correlated components: splitter, producer, and consumer. The concept of splitter-producer-consumer represents an abstract and flexible model for distributed media processing. Based on this model, inter-media synchronization can be also achieved in the cloud. Using the media cloud to synchronize two streams (DVB stream and IP stream), it is possible to show the picture-in-picture functionality on mobile devices. A particular use case for inter-media synchronization is the video of a sign language interpreter along with the video of a TV channel.

Figure 14 shows how the splitter-producer-consumer model can be utilized to do inter-media synchronization. The producer is extended to do various media processing: text overlaying, SVG overlaying, video mixing, and sync. A producer component can be configured to any of these operations. Inside the producer component, the detailed GStreamer-based script to do video mixing and inter-media synchronization will be referred to the sync-module developed by TNO [14]. As illustrated by Figure 14, two splitters created for two separate streams. One is the main TS stream which contains the main video, audio, and subtitles. The other is the sign language video. A producer takes care of the video streams, synchronizing them first and then combining them together with the subtitle. Another producer takes of the audio stream. Finally, the video stream and audio stream will be multiplexed using a proper container.

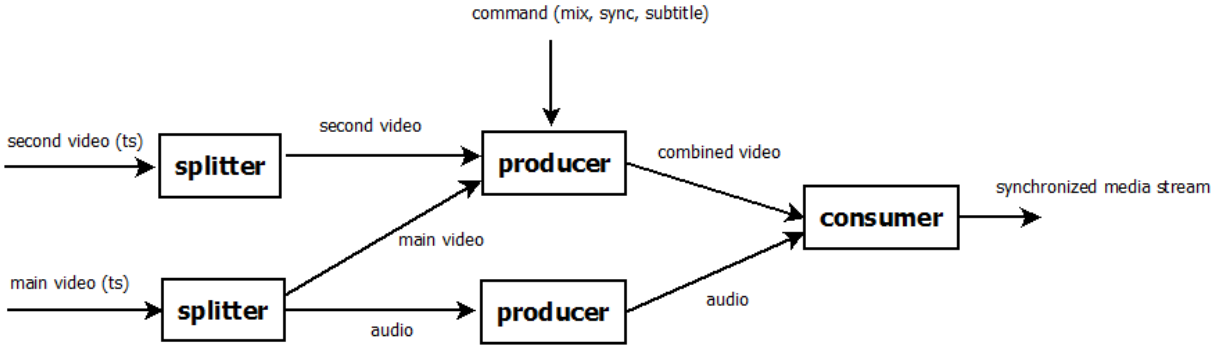


Figure 14: workflow of inter-media synchronization in the cloud

4.7. Cloud Resource Analytics

The media cloud platform must provide suitable monitoring tools to ascertain the cloud resource usage and collect data that can be used to predict future media processing resource requirements. To monitor and analyse the cloud resource usage, a pull-based mechanism has been designed to collect the detailed CPU, network, and memory usage information of all virtual machines, pipelines, and tasks. As illustrated in Figure 15, an agent processer is initially launched on every active virtual machine. It receives the command from the media manager to know when it should start to capture which type of information for which task and where is the destination to push back the captured information. On the other hand, it controls the monitor module to capture the required data from the local operation system. When the user accesses the analytics web portal from a browser, the analytics server will tell the media manager what are the metrics triggered by the user. After that, the media manager sends the command to all active agents for triggering them to capture the requested information. After a while, the analytics server will receive information from different agents, cache the recent results in a document-based database, and further present them to the user as pictures plotted over time.

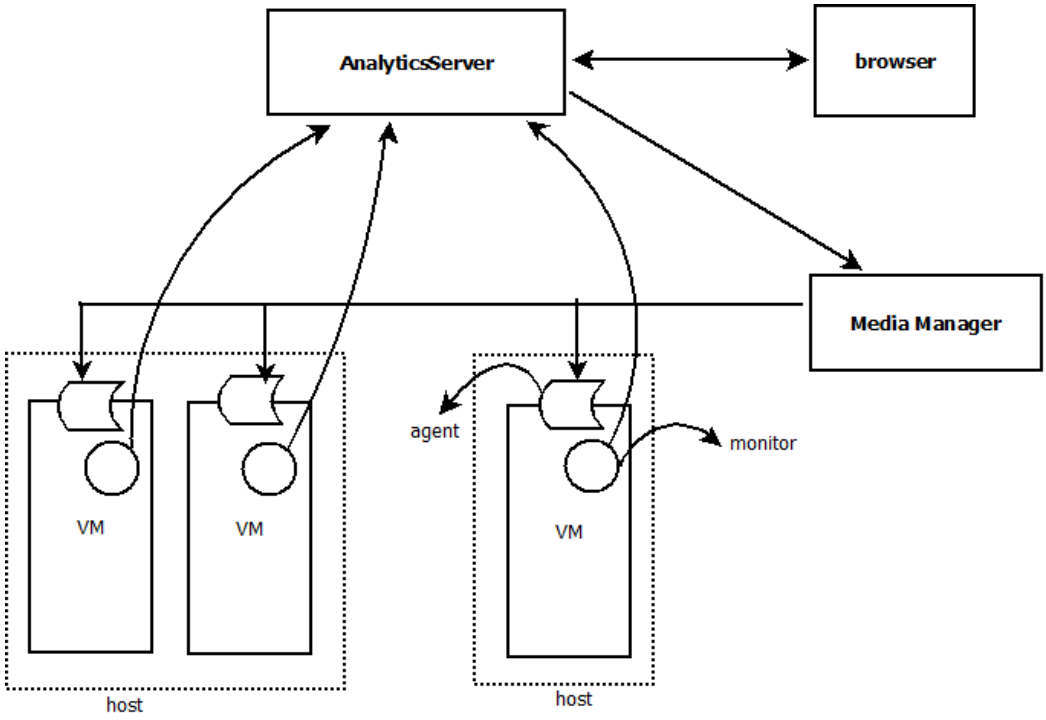


Figure 15: pull-based analytics service for media cloud

4.8. Summary

Starting with the basic approach to media processing cloud offloading, this section presents the system architecture design of the media cloud and further describes its underlying communication mechanisms. With more details, the following core issues are describe with their workflow diagrams: 1) device capability detection which allows us to identify the properties of a terminal device and then to determine the media pipeline and tasks suitable for the device; 2) dynamic pipeline composition which can construct the required media pipeline dynamically and share/reuse pipelines and tasks to reduce cloud usage; 3) cloud resource analytics that provides the real-time analytics results on what is happening in the media cloud in terms of resource usage of each virtual machine, media pipeline, and media task.

5. Interfaces of Media Cloud

5.1. Interface analysis

As a platform, the media cloud provides media processing services for both content providers/broadcasters and third-party applications. On the one hand, it requires content providers to bring their media content into the cloud and provides some processing for their content to enable multi-screen media consumption experience. On the other hand, it opens up some opportunities for third-party application developers to affect the running media pipeline or specify some advanced processing, such as media synchronization, content editing, and dynamic ad insertion. In some cases, the content providers can also play an application developer role in developing media applications for their own content, or the applications rely on users to provide user-generated content to put into the cloud.

Regarding those requirements, the following two levels of interfaces are designed and implemented for managing media pipelines in the cloud. As shown in Figure 16, all media tasks are managed by the media manager. When a media source is first provided by content providers, the media manager will spawn some splitter to fetch the media source content and then split it into separate streams. By doing so, the media manager can know the meta-data information of the media source, for example, the number of streams and the type of each stream. After this first step of pre-processing, the media manager allows the following managements.

Basic management (static, more likely for content providers)

With basic management producers and consumers will be spawned to generate some media streams in advance, according to predefined scripts before users/clients request them. Therefore, those components are prepared to provide adaptive streams for the potential users/devices. This type of pipeline creation is triggered by content providers when they provide a media source.

Advanced management (dynamic, more likely for applications)

With advanced management producers and consumers are spawned on the fly, according to the requests from users within applications. For example, when the user chooses to have the sign language video inside the main video, the media synchronization functionality will be triggered and then the media manager creates the corresponding producer to combine related input streams and synchronize them. Here the interface to do advanced pipeline management is more likely used by applications and triggered by users.

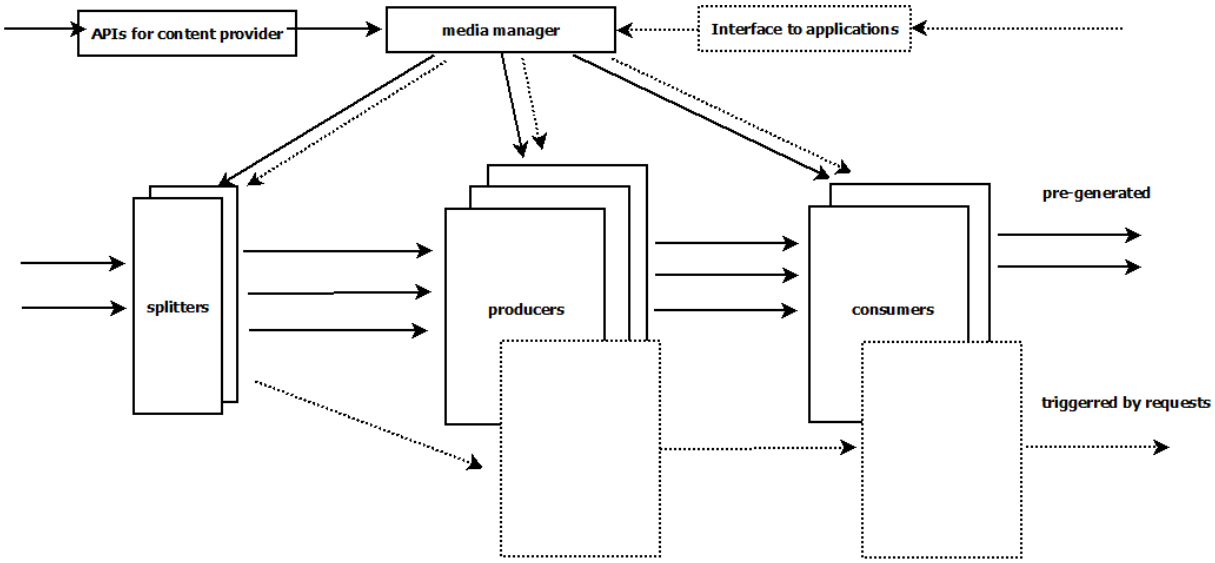


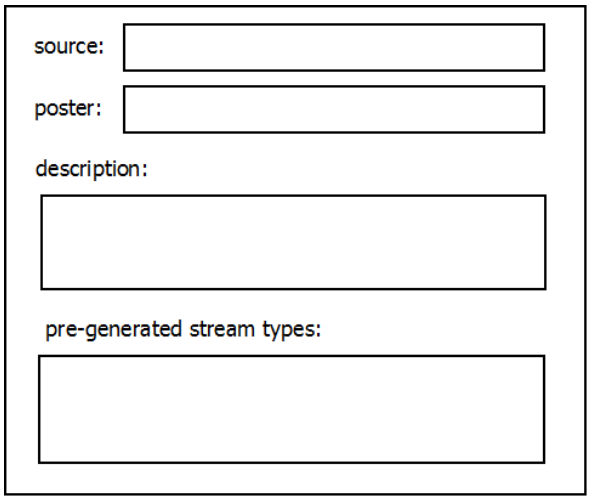
Figure 16: Interfaces of media cloud

5.2. Interface specification

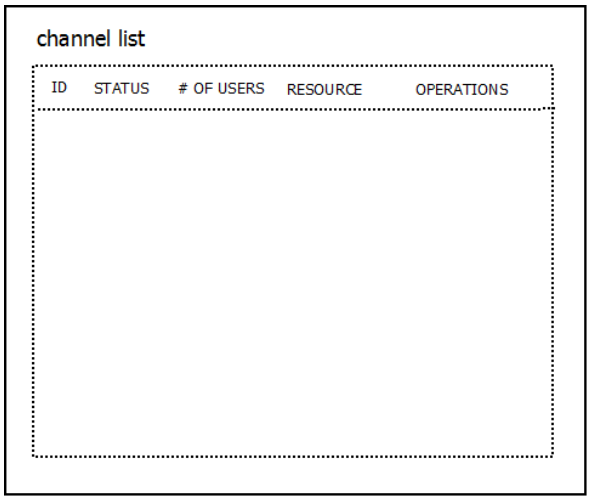
The HBB-NEXT project is in the progress of integrating different components to enable a set of media applications on both TV sets and personal devices. According to the diagram in Figure 3, the media cloud needs to provide a UI-based interface for broadcasters to specify and manage channels on the back-end side. On the front-end side, it communicates with browsers and delivers generated streams to users. Here the final interfaces are described with some diagrams. The full implementation of these interfaces will be finished and be integrated with other partners in the next stage of the project.

These interfaces will be implemented as web-based services provided by the media manager. Figure 17 illustrates them with some designed diagrams. Figure 17(a) shows the interface to create a normal channel. After specifying the media source location, its thumbnail, and a brief description, the content provider can add some pre-generated stream types. For these types, the media manager will schedule media tasks to launch the corresponding media pipelines in advance. Figure 17(b) shows the channel list with their current status information. It also provides some operations for the content provider to control each channel, such as start or stop a channel. Figure 17(c) shows the interface to launch a special channel that allows inter-media synchronization functionality.

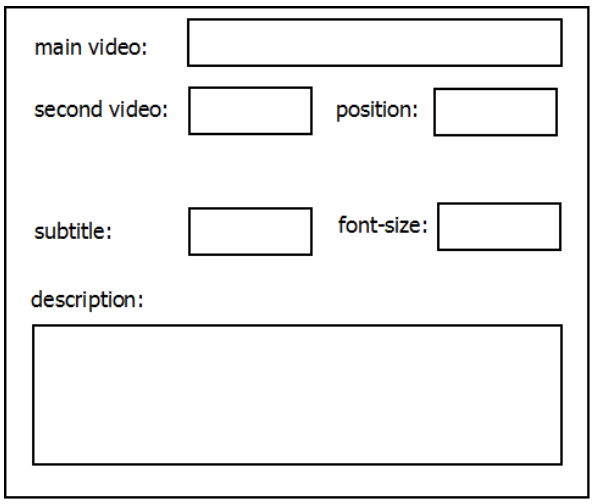
To do that, the content provider has to specify a main video with both video and audio streams, plus a second video (for example, sign language) that will be overlayed into the main video at some corner and a subtitle stream that will display the subtitle synchronized with the main video. Figure 17(d) shows the channel list with poster pictures for users to navigate channels.

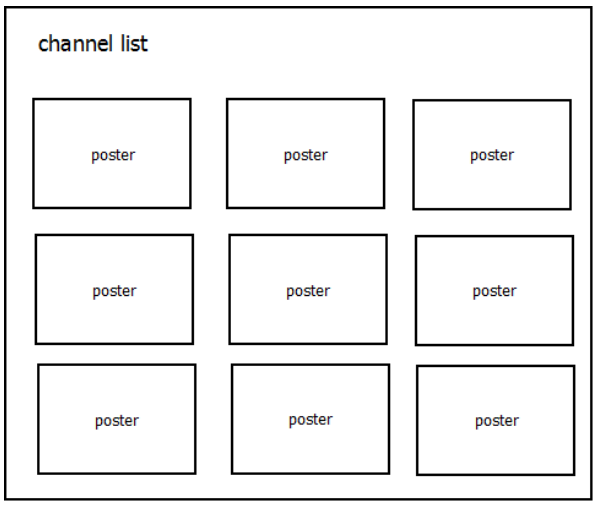


17(a) Create a channel



17(b) Manage channels





17(c) Inter-media synchronization *17(d) Channel list on the client side*

Figure 17: illustration of media cloud interfaces

6. Demonstration

In this section the setup of the media cloud is described shortly and its current functionalities is introduced by some screenshots.

6.1. Setup of system deployment

An OpenStack-based cloud is set up in NEC's lab for the development and demonstration purpose. As Figure 18 shows, there is a server room with a network router and four servers. Two of them are used computer nodes in the OpenStack-based media cloud, one is used as the controller node, and the remaining one is the admin node. The admin node is used as the client with the user credential to manage the media cloud. The following tools and libraries are used in the setup.

- OpenStack: the Essex version is running on Ubuntu 12.04 server hosts to provide cloud infrastructure
- VM image: Ubuntu 11.10 based virtual machine images with the installed GStreamer framework and python-based executable component scripts, including splitter, producer, consumer
- Python: the script interpreter for executing scripts inside virtual machines, version 2.7
- GStreamer: the media processing framework, version 0.10.36
- NodeJS: the platform to support Javascript-based server, version 0.82
- MongoDB: store utilization data acquired from vmcontroller scripts, version 2.2.2
- RabbitMQ: for messaging between different components and its Pika interface module
- Pika: the module providing bindings for communicating via RabbitMQ from python, version 0.9.8
- Socket.IO: the module responsible for all communication with client browsers, version 0.9.13

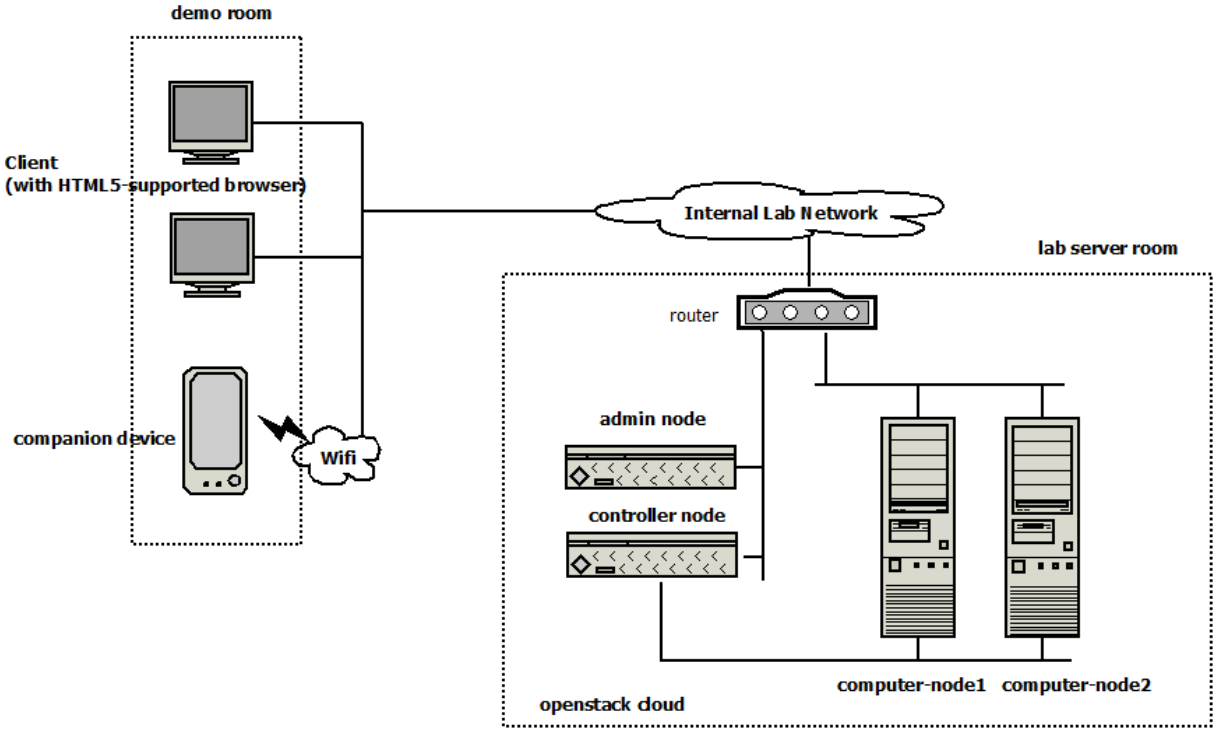


Figure 18: cloud-based service offloading platform

6.2. Current Functionalities

Here is the demonstration of the functionalities that have been already implemented in media cloud with a few system screenshots.

Figure 19 shows the channel list on the front-end client side. The current channel list is generated when the system starts. As the content provider manages the channels, the list is accordingly updated.

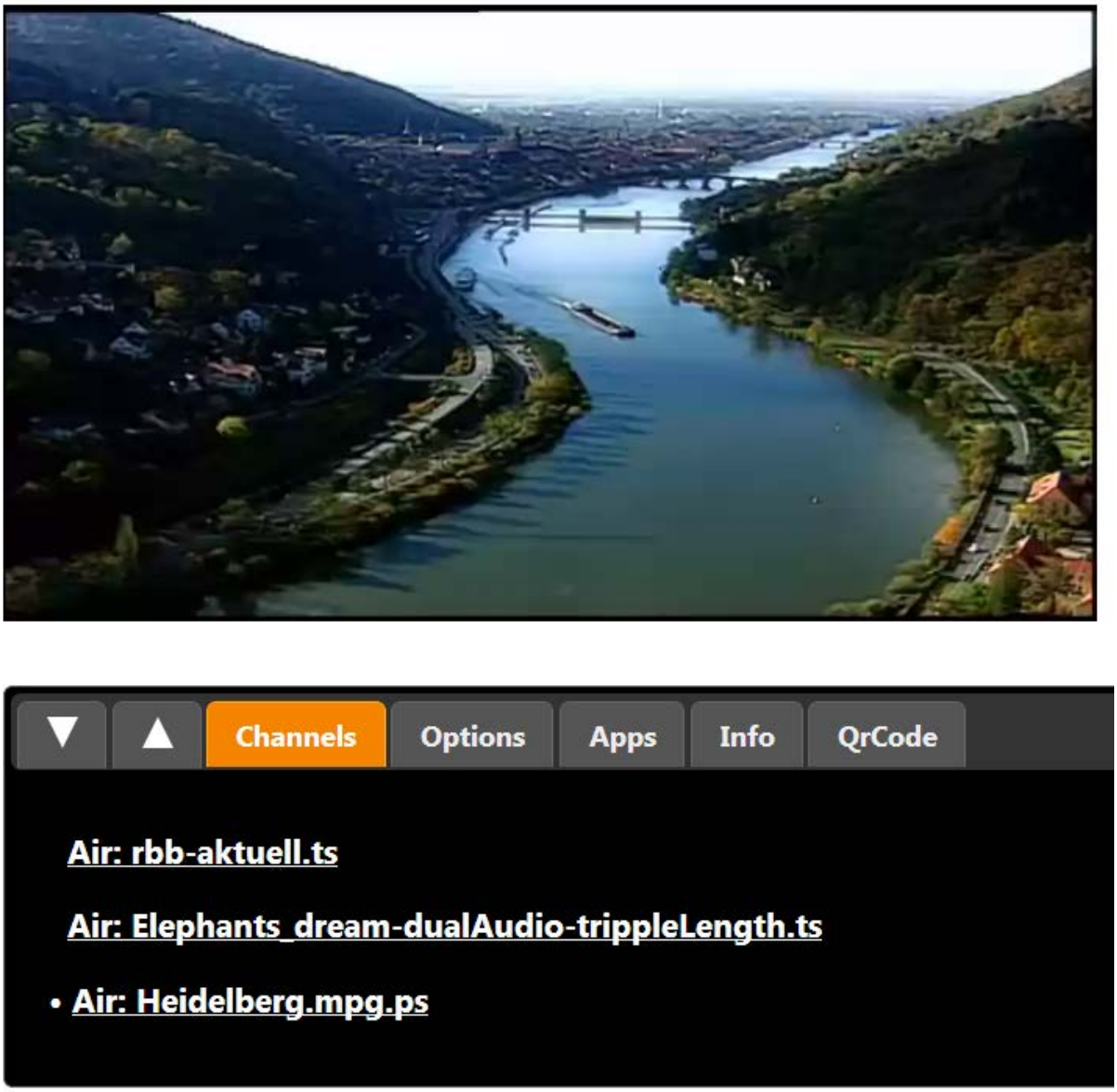
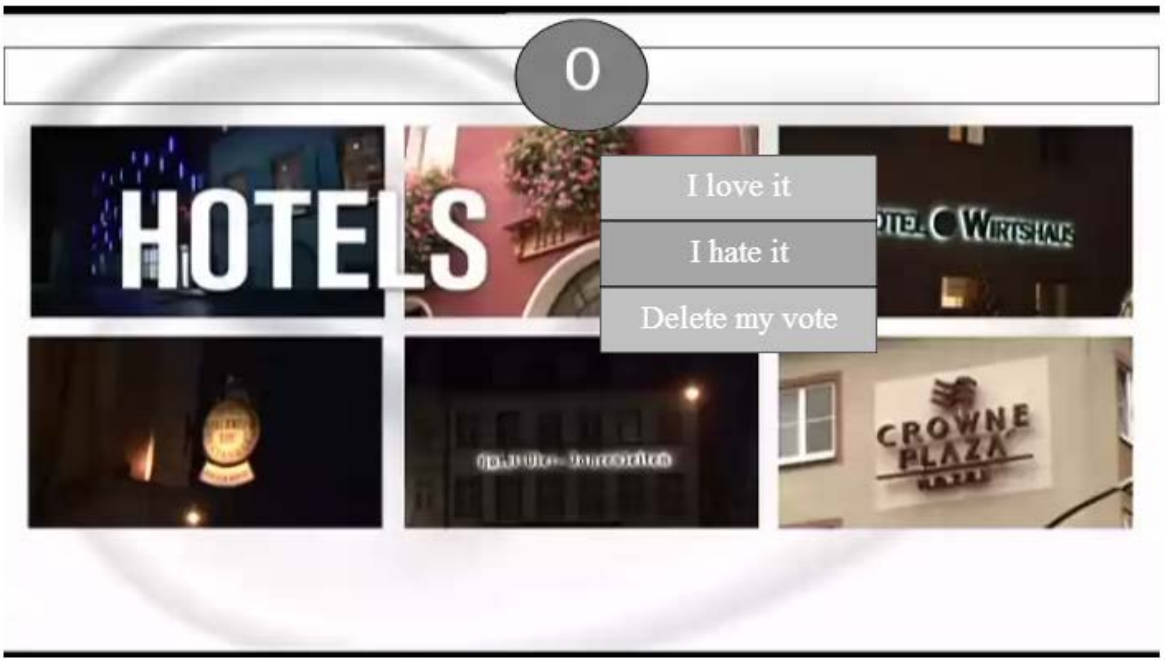


Figure 19: channel list

Figure 20 shows the application list of the front-end portal on the client side. The user can now enable multiple applications from the application list. For each channel, all the applications will be running in the cloud and communicate with the underlying Media Cloud platform through two types of APIs: one to give the output to Media Cloud as a SVG overlay and the other one to give output to the client. The Media Cloud platform has a SVG combiner, which basically takes multiple SVG inputs from applications, combines them, and puts the combined SVG into the associated producer as a new overlay.



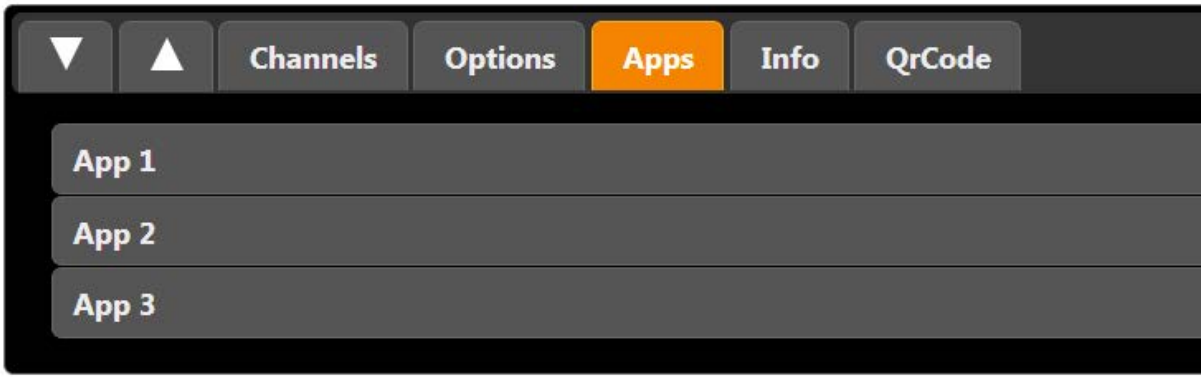


Figure 20: voting application over the video overlay

Figure 21 shows two screenshots of the cloud resource analytics results. The first one shows how many virtual machines have been launched in the cloud and plots the total CPU utilization of each virtual machine over time in different colours. The analytics result indicates whether each virtual machine is properly utilized and how balanced the workload has been spread across them. The second one shows the inbound bandwidth usage of all launched virtual machines and how each of them changes over time. The result shows the inbound bandwidth usage is largely different across different virtual machines. The producer can dramatically compress the received data stream when transforming video content from high resolution to low resolution.

If the producer and the consumer are located on two different virtual machines, the introduced inbound bandwidth usage for these two virtual machines will be largely different. This result also indicates a sophisticated task allocation algorithm needs to be considered in order to alleviate network traffic imbalance among virtual machines in the media cloud.

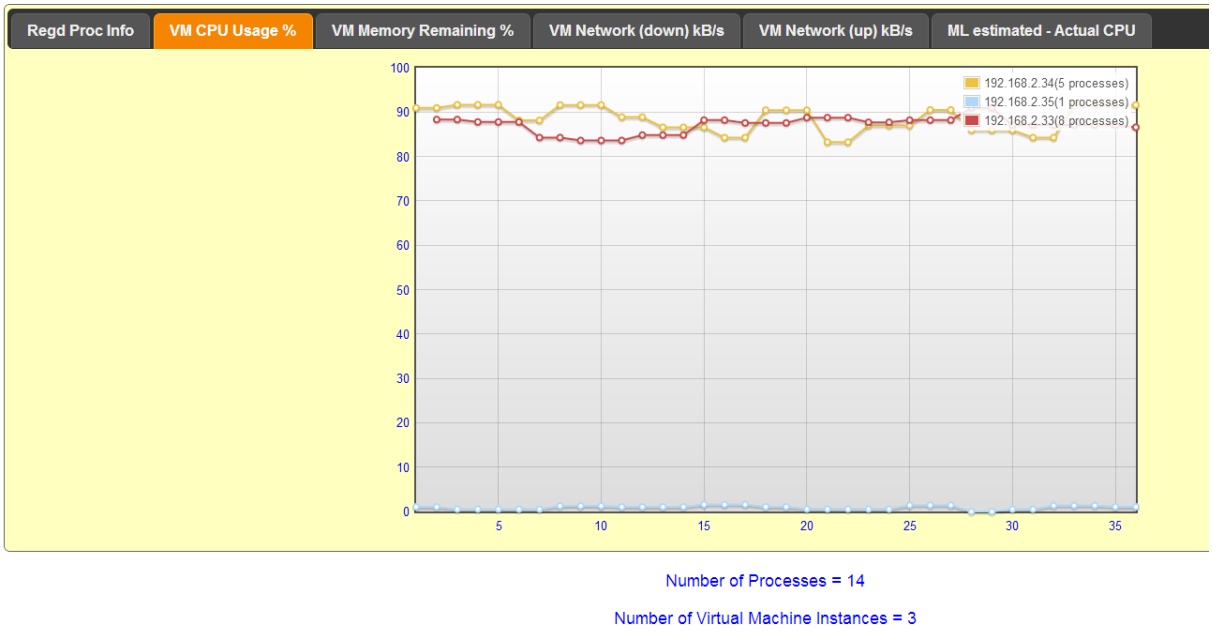


Figure 21: cloud resource analytics service

Figure 22 shows a very simple web-based interface for the content provider to manage and control their channels. From the channel list, the content provider can choose to start or stop a channel. Currently, the channel list is pre-defined by a file server and the system does not allow the content provider to create a new channel on the fly without stopping the file server. A more convenient and advanced management portal will be implemented to allow dynamic channel creation.

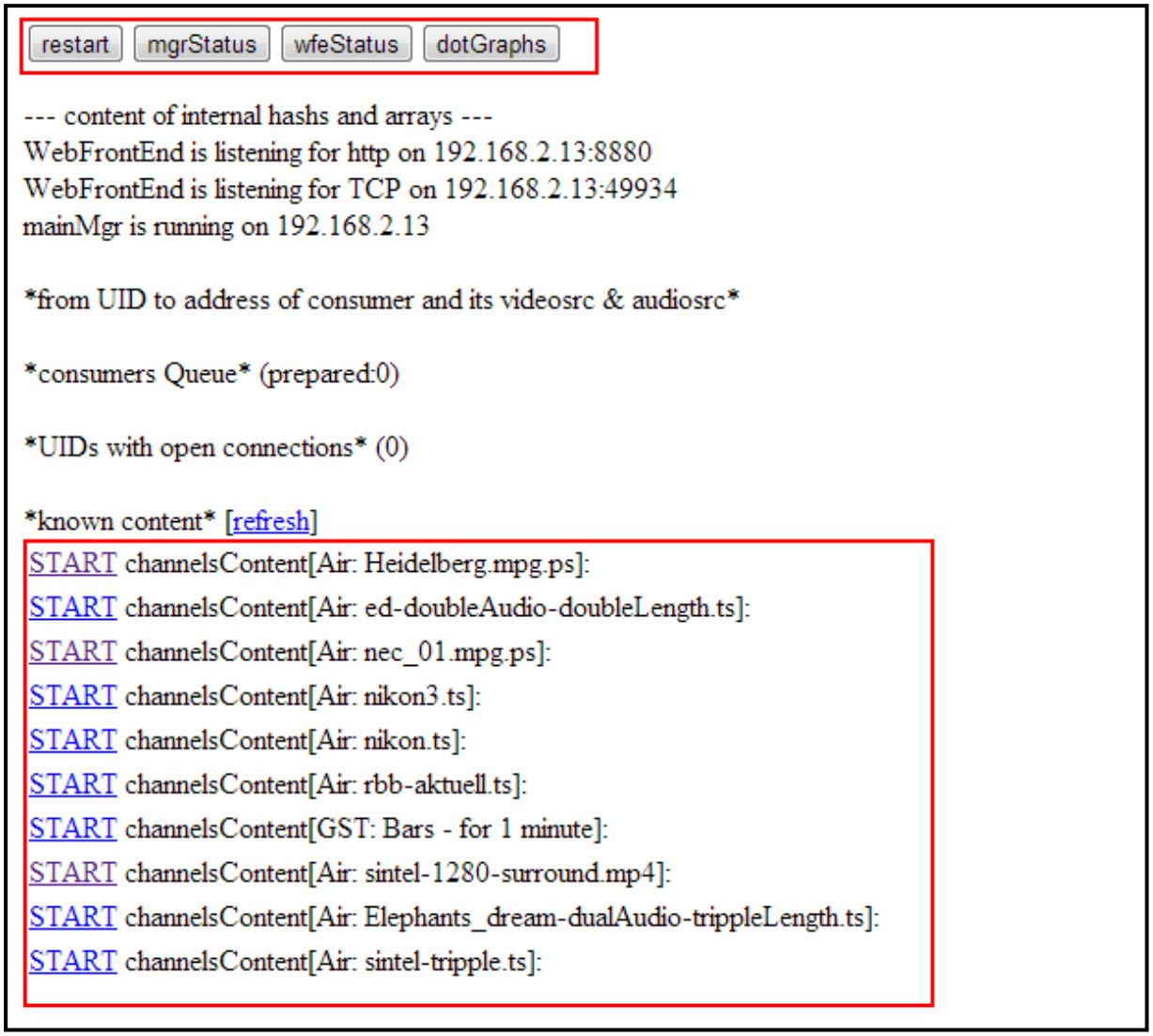


Figure 22: channel management portal

7. Related Research in the State-of-the-art

There are some existing studies related to cloud offloading. According to our investigation, offloading has been considered at different levels in different use cases. Microsoft Research is working on a cloud-based GPS offloading technology at the device level [8]. The cloud-based GPS offloading reduces high power requirements of GPS sensing by splitting the GPS location sensing into a device part and a cloud part and offloading the heavy number crunching to the cloud. CloneCloud [9] is a computing offloading from devices to the cloud, working at the level of application-layer virtual machines such as JavaVM. It proposes cloud-augmented execution using a cloned VM image as a powerful virtual device in the cloud. Using this approach, a parallelizable application can invoke multiple VMs to execute in the cloud in a seamless and on-demand manner such as to achieve greater reduction on execution time and energy consumption. MAUI [10] describes a system that enables fine-grained energy-aware offloading of mobile code to infrastructure. Its main aim is to optimize energy consumption of a mobile device, by estimating and trading off the energy consumed by local processing vs. transmission of code and data for remote execution. ThinkAir [11] is close to MAUI and they all present a cloud offloading approach at the application code level.

Our work has been also pretty much focused on cloud-based offloading, but more for media processing. As compared to the related work mentioned above, our approach focus more on the offloading of media processing at the application level, which is a light-weight solution more suitable for media applications. The most similar work is CloudMov [12], which is a cloud-based social TV for mobile devices. But differently, CloudMov sticks to cloud-based transcoding while our media cloud is a more scalable and flexible media processing platform that allows more media processing tasks in a distributed manner. For example, media tasks in the same pipeline can be deployed across multiple virtual machines when a single virtual machine does not fit the resource requirement of the media pipeline. Also, the HBB-NEXT media cloud platform enables inter-media synchronization, overlaying, and audio/video watermarking.

8. Conclusion and Outlook

8.1. Conclusion

In this deliverable we present the detailed architecture design and core technologies of our media cloud which enables cloud-based service offloading for a variety of media applications, especially in the environment of HBB TV with second devices. More importantly, we performed a measurement analysis for media processing which provides lots of inputs and indications for the architecture design of our media cloud and points out some future research issues. One of the big achievements is that we designed and implemented the initial media cloud platform which already enables some media applications. Based on the proposed distributed media pipeline model, our cloud-based service offloading platform is scalable and flexible to support different types of media applications, such as adaptive streaming, inter-media synchronization in the cloud, and social voting on second devices. Some of these functionalities have been internally demonstrated by our cloud setup in the lab. Regarding the requirements from other partners, we further specified the interfaces of our media cloud which can provide live streams suitable for different devices and also synchronized streams with multiple video inputs for second devices.

8.2. Future work and plan

In the rest of the project we will focus more on the integration of media cloud with other applications. For this purpose, we will refine and implement the designed interfaces to make sure the applications developed by the other partners can access the live stream services from our media cloud. More importantly, we will enhance the current splitter, producer, and consumer tasks to support the following features: 1) bring real TV channels into the cloud and use the extended splitter to deal with them; 2) extend the current producer to enable inter-media synchronization for multiple video inputs and one audio input; 3) extend the current consumer to support more containers like TS over HTTP and HLS. Also, to improve the cloud resource usage of media pipelines, we will explore resource allocation and task scheduling strategies for dealing with the dynamicity and heterogeneity of cloud infrastructure.

9. References

[1] HBB-NEXT Deliverable D4.1 - ANALYSIS: Cloud-Based Services and Service/Content Synchronisation, http://www.hbb-next.eu/index.php/documents

[2] HBB-NEXT Deliverable D2.1 - Usage Scenarios and Use Cases, http://www.hbb-next.eu/index.php/documents

[3] HBB-NEXT Deliverable D2.2 - System-, Service-, and User Requirements, http://www.hbb-next.eu/index.php/documents

[4] HBB-NEXT Deliverable D6.1.1 - Initial Version of the HBB-NEXT System Architecture, http://www.hbb-next.eu/index.php/documents

[5] Ou, Zhonghong, et al. "Exploiting hardware heterogeneity within the same instance type of amazon EC2." In *The 4th USENIX Workshop on Hot Topics in Cloud Computing (HotCloud)*, 2012

[6] Openstack Cloud Software. [Online] http://www.openstack.org

[7] Gstreamer: Open Source Multimedia Framework. Gstreamer. [Online] http://gstreamer.freedesktop.org

[8] Jie Liu, Bodhi Priyantha, Ted Hart, Heitor S. Ramos, Antonio A. F. Loureiro, and Qiang Wang, "Energy efficient GPS sensing with cloud offloading", In *Proc. of the 10th ACM Conference on Embedded Network Sensor Systems (SenSys '12)*, 2012, ACM, New York, NY, USA, 85-98

[9] Byung-Gon Chun, Sunghwan Ihm, Petros Maniatis, Mayur Naik, and Ashwin Patti. Clonecloud: Elastic execution between mobile device and cloud. In *Proc. of EuroSys*, 2011

[10] Eduardo Cuervo, Aruna Balasubramanian, Dae-ki Cho, Alec Wolman, Stefan Saroiu, Ranveer Chandra, and Paramvir Bahl. MAUI: making smartphones last longer with code offload. In *MobiSys*, 2010

[11] Kosta, Sokol, Andrius Aucinas, Pan Hui, Richard Mortier, and Xinwen Zhang. "Thinkair: Dynamic resource allocation and parallel execution in the cloud for mobile code offloading." In *INFOCOM, 2012 Proceedings IEEE*, pp. 945-953. IEEE, 2012

[12] Wu, Y., Zhang, Z., Wu, C., Li, Z., & Lau, F. C., "CloudMoV: Cloud-based Mobile Social TV", in *IEEE Transactions on Multimedia*, Jan. 2013

[13] RabbitMQ, [Online] http://www.rabbitmq.com/

[14] HBB-NEXT Deliverable D4.2 - DESIGN AND PROTOCOL: Middleware Components Content Synchronisation/Cloud Service Offloading, http://www.hbb-next.eu/index.php/documents

[15] HBB-NEXT Deliverable D4.4.2 - SOFTWARE: Intermediate Middleware Software Components for Cloud Service Offloading

[16] HBB-NEXT Deliverable D4.6.2 - SOFTWARE: Final Middleware Software Components for Cloud Service Offloading