March 2015

Effectiveness of Cloud Services for Scientific and VoD Applications

Dilip Kumar Krishnappa
University of Massachusetts - Amherst

EFFECTIVENESS OF CLOUD SERVICES FOR SCIENTIFIC AND VOD APPLICATIONS

A Dissertation Presented

by

DILIP KUMAR KRISHNAPPA

Submitted to the Graduate School of the
University of Massachusetts Amherst in partial fulfillment
of the requirements for the degree of

DOCTOR OF PHILOSOPHY

February 2015

Electrical and Computer Engineering

EFFECTIVENESS OF CLOUD SERVICES FOR SCIENTIFIC AND VOD APPLICATIONS

A Dissertation Presented

by

DILIP KUMAR KRISHNAPPA

Approved as to style and content by:

Michael Zink, Chair

Lixin Gao, Member

David Irwin, Member

Prashant Shenoy, Member

Christopher V. Hollot, Department Chair
Electrical and Computer Engineering

To my wonderful parents, caring brother and loving sister.

ACKNOWLEDGEMENTS

First and foremost, I would like to thank my advisor Prof. Michael Zink. It has been an honor to be his first PhD student. For the past 5 years, he has been a constant support and guide in all my academic decisions. His mentoring and guidance has helped enjoy my stay at UMass and provided confidence to finish my PhD.

I would like to thank my committee members Prof. David Irwin, Prof. Lixin Gao and Prof. Prashant Shenoy for presiding over my dissertation. I would like to thank Prof. Ramesh Sitaraman from UMass Amherst, Prof. Carsten Griwodz and Prof. Pål Halvorsen from University of Oslo for giving me an opportunity to work with them during parts of my dissertation. I would like to thank all the reviewers and shepherds of all the conference and journal articles I have submitted during my PhD. I would like to thank my colleagues Cong Wang, Divyashri Bhat, Samamon Khemmarat and Eric Adams for working closely with me on projects related to my dissertation.

My time at UMass was made enjoyable in large part due to the many friends that became a part of my life during these past 5 years. A special thanks to Ravi Vantipalli, Vikram Desai, Abhinna Jain, Satish Inampudi, Chilakam Madhusudan, Siddhartha Nalluri, Srikar Damaraju and Pavan Prakash. It is their support during all the lows and highs of being a PhD student has helped me through the past 5 years and provided moments to cherish for life.

Finally, I am obliged to my grand parents, parents, my brother Amith Kumar, my sister Shruthi, my cousins Chethan and Chaitra, who have influenced me the most and were always there to support and encourage me. They have been my source of strength. I dedicate this thesis to them.

ABSTRACT

EFFECTIVENESS OF CLOUD SERVICES FOR SCIENTIFIC AND VOD APPLICATIONS

FEBRUARY 2015

DILIP KUMAR KRISHNAPPA

B.E., VISVESWARAIAH TECHNOLOGICAL UNIVERSITY, INDIA

M.S., UNIVERSITY OF MASSACHUSETTS AMHERST

Ph.D., UNIVERSITY OF MASSACHUSETTS AMHERST

Directed by: Professor Michael Zink

Cloud platforms have emerged as the primary data warehouse for a variety of applications, such as DropBox, iCloud, Google Music, etc. These applications allow users to store data in the cloud and access it from anywhere in the world. Commercial clouds are also well suited for providing high-end servers for rent to execute applications that require computation resources sporadically. Cloud users only pay for the time they actually use the hardware and the amount of data that is transmitted to and from the cloud, which has the potential to be more cost effective than purchasing, hosting, and maintaining dedicated hardware. In this dissertation, we look into the efficiency of the cloud Infrastructure-as-a-Service (IaaS) model for two real time high bandwidth applications: A scientific application of short-term weather forecasting and Video on Demand services.

Short-term Weather Forecasting

Since today's weather forecasts only cover large regions every few hours, their use in severe weather is limited. Dedicating high-end servers for executing scientific applications that run intermittently, such as severe weather detection or generalized weather forecasting, wastes resources. In this dissertation, we present CloudCast, an application that provides short-term weather forecasts depending on user's current location. Since severe weather is rare, CloudCast leverages pay-as-you-go cloud platforms to eliminate dedicated computing infrastructure. CloudCast has two components: 1) an architecture linking weather radars to cloud resources, and 2) a Nowcasting algorithm for generating accurate short-term weather forecasts.

Our hypothesis is that the connectivity and diversity of current cloud platforms mitigate the impact of staging data and computation latency for Nowcasting. In evaluating our hypothesis, in this dissertation, we study CloudCast's design space, which requires significant data staging to the cloud. Short-term weather forecasting has both computational and network requirements. While it executes rarely, whenever severe weather approaches, it benefits from an IaaS model; However, since its results are time-critical, sufficient bandwidth must be available to transmit radar data to cloud platforms before it becomes stale. In this dissertation, we analyze the networking capabilities of multiple commercial (Amazon's EC2 and Rackspace) and research (GENICloud and ExoGENI) cloud platforms for our CloudCast application. We conduct network capacity measurements between radar sites and cloud platforms throughout the country. Our results indicate that research cloud testbeds performs the best for both serial and parallel data transfer with an average throughput of 110.22 Mbps and 17.2 Mbps, respectively. We also found that the cloud services perform better in the distributed data transfer case, where a subset of nodes transmit data in parallel to a cloud instance. The results also indicate that serial transfers

achieve tolerable throughput, while parallel transfers represent a bottleneck for real-time weather forecasting.

In addition to network capacity analysis, we also analyze the computation capability of cloud services for our CloudCast application. We measure the cost and computation time to generate 15-minute forecasts for real weather data on cloud services. Also, we analyze the forecast accuracy of executing Nowcasting in the cloud using canned weather data and show high accuracy for ten minutes in the future. Finally, we execute CloudCast live using an on-campus radar, and show that it delivers a 15-minute Nowcast to a mobile client in less than 2 minutes after data sampling started, which is quick enough for users and emergency management personnel to take desired action.

VoD Services

Advancement in Internet bandwidth connectivity to homes coupled with the streaming of TV shows and other media content on the Internet has led to a huge increase in VoD services in the past decade. The advent of sites such as YouTube, Hulu, Netflix, DailyMotion and various other video content providers, make it feasible for the users to share their own videos on these VoD websites or watch their favorite movies or TV shows online. The popularity of VoD services has motivated U.S national TV channels such as ABC, NBC and CBS to offer their prime time programs on the Internet for users to catch up on their shows via streaming on their websites.

In 2012, Internet video traffic made up 57% of the total network traffic in the Internet. This shows that with the popularity of VoD services, the amount of Internet traffic due to video increases tremendously. However, the popularity of videos offered in VoD services follow a long tail distribution, where only a few videos are highly popular and most of the videos offered are requested rarely. This long tail popularity distribution in VoD services is especially significant when content is user generated. Analysis from a YouTube trace show that, only 23% of the YouTube videos are

requested more than once over a period of 24 hours. Consequently, 77% of the videos which are requested only once reduces the efficiency of caches deployed all around the world to serve the videos requested.

Video streaming in VoD services such as Netflix, Hulu, YouTube etc., have moved from traditional streaming of a fixed bit rate video file to the client to Adaptive Bit Rate streaming (ABR). ABR streaming is realized through video streaming over HTTP where the source content is segmented into small multi-second (usually between 2 and 10 seconds) segments and each segment is encoded at multiple bit rates. The process of creating multiple bit rate versions of a video is called *transcoding*. Once each video is transcoded into multiple bit rates, ABR streaming allows the client to choose an appropriate quality version for each video segment based on the available bandwidth between the source and client. However, long tail popularity distribution of VoD services shows that transcoding all video provided by VoD services to all the bit rates supported wastes transcoding resources and storage space.

In this dissertation, we provide solutions to the issues posed by the long tail popularity distribution of VoD services. We look into the feasibility of using cloud services to reduce the traffic in the Internet by reducing the amount of video requests from the long tail distribution of VoD services. In particular, this dissertation focuses on YouTube video service, the world's most popular Internet service that hosts user-generated videos. In 2011, YouTube had more than 1 trillion views or video requests, which lead to a huge amount of network traffic. In this dissertation, we present a related list reordering approach to improve the efficiency of caches hosting videos from VoD services. Also, we look at the feasibility of using cloud services to reduce the transcoding workload by performing online transcoding. We present a series of transcoding policies to reduce the transcoding workload and maintain performance of providing ABR streaming at the client.

Improving Cache Efficiency

To improve the cache efficiency of hosting YouTube videos, we take advantage of the user behavior of requesting videos from the related list provided by YouTube and the user behavior of requesting videos from the top of this related list. We recommend a related list reordering approach which modifies the order of the videos shown on the related list based on the contents in the cache. The main goal of our reordering approach is to push the contents already in the cache to the top of the related list and push non-cached contents towards the bottom, which increases the likelihood that the already cached content will be chosen by the viewer. We analyze the benefits of our approach by an investigation that is based on two traces collected from an university campus. Our analysis shows that the proposed reordering approach for related list would lead to a 2 to 5 times increase in cache hit rate compared to an approach without reordering the related list. The increase in hit rate would lead to a 5.12% to 18.19% reduction in server load or back-end bandwidth usage. This increase in hit rate and reduction in back-end bandwidth reduces the latency in streaming the video requested by the viewer and has the potential to improve the overall performance of YouTube's content distribution system.

Research has shown that, YouTube follows a 3-tier caching hierarchy to host its videos. In this dissertation, we use the same 3-tier caching strategy implemented on cloud services to host videos based on their global popularity and use the related list reordering approach to increase the number of requests or hits for the most popular cache or cloud server. Hosting of videos based on their global popularity, allows us to differentiate the long tail and reduce the energy required to serve videos which are requested rarely. Analysis of our video placement strategy on cloud instances shows that it can increase the hit rate of cloud instances hosting highly popular videos by almost 90% and helps reduce the energy required to serve medium and low popularity videos by 20%.

Online Transcoding

Dynamic adaptive streaming has grown over the last few years and is on its way to becoming the de-facto standard for video streaming. As a result, video providers are faced with the burdensome task of transcoding each of their videos to a large number of bit rates and formats to suit the great diversity of end-users' devices and connectivities. The current approach to transcoding in adaptive bit rate streaming is to transcode all videos in all possible bit rates and uploading these videos to a content delivery network (CDN) that can deliver the videos to users. We show that this approach wastes transcoding resources, since a large fraction of the transcoded video segments are never watched by users. To reduce transcoding work, we propose several *online* transcoding policies that transcode video segments in a "just-in-time" fashion such that a segment is transcoded only to those bit rates that are actually requested by the user. However, a reduction in the transcoding workload should not come at the expense of a significant reduction in the quality of experience of the users. To establish the feasibility of online transcoding, we first show that the bit rate of the next video segment requested by a user can be predicted ahead of time with an accuracy of 99.7% using a Markov prediction model. This allows our online algorithms to complete transcoding the required segment ahead of when it is needed by the user, thus reducing the possibility of freezes in the video playback. To derive our results, we collect and analyze a large amount of request traces from one of the world's largest video CDNs consisting of over 200 thousand unique users watching 5 million videos over a period of three days. The main conclusion is that online transcoding schemes can reduce transcoding resources by over 95% without a major impact on the users' quality of experience.

In conclusion, in this dissertation, we show the efficiency of IaaS model of cloud services for two applications, weather forecasting and VoD services. We show that, cloud services are efficient in both network and computation for real time scientific

application of weather forecasting. We present a related list reordering approach, which reduces the network traffic of serving videos from VoD services and improve the efficiency of caches deployed to serve them. Also, we present transcoding policies to reduce the transcoding workload and maintain performance of providing ABR streaming of VoD services at the client with online transcoding in the cloud.

TABLE OF CONTENTS

LIST OF TABLES

Table **Page**

LIST OF FIGURES

Figure **Page**

CHAPTER 1

INTRODUCTION

Cloud platforms have emerged as the primary data warehouse for a variety of applications, such as DropBox, iCloud, Google Music, etc. These applications allow users to store data in the cloud and access it from anywhere in the world. Commercial clouds are also well suited for providing high-end servers for rent to execute applications that require computation resources sporadically. Cloud users only pay for the time they actually use the hardware and the amount of data that is transmitted to and from the server, which has the potential to be more cost effective than purchasing, hosting, and maintaining dedicated hardware. Amazon's Elastic Compute Cloud (EC2) [8] and Rackspace [32] are two of many commercial cloud services where users can *rent* compute and storage resources based on their needs and are charged based on the usage time of compute resources, the amount of data that is transferred, and the amount of data that is stored.

Commercial clouds use the pay-as-you-use model where the users are charged for resource usage on an hourly basis. In contrast, research clouds such as GENI-Cloud [104, 45] and the ExoGENI cloud [15] provide free resources for the research community. Apart from the fact that the usage of these cloud platforms is free for the research community, they bear the following additional advantages. First of all, researchers can use them to develop prototypes of scientific cloud applications. Large-scale implementation of these applications will still have to happen in commercial clouds since the research clouds provide only a limited number of resources (e.g., available compute nodes or overall storage space). Second, research clouds such

as ExoGENI (with its NEuca extensions [28]) allow for dynamic configuration of the network topology within the cloud, a feature that is not provided by commercial clouds. Third, specific research clouds are connected via next-generation research networks (Internet2 ION [21]) that allow the provisioning of dedicated, isolated network resources. The latter will help researchers to better understand how distributed applications that run in the cloud can benefit from new network technologies. This will, for example, allow us to investigate how a dedicated layer 2 connection between the source and the receiving instance in the cloud will impact the overall performance of an application.

In this dissertation, we look at two real-time high bandwidth applications that can use the elasticity of cloud to improve their performance. One is a scientific weather application, called *CloudCast*, which uses Nowcasting algorithm, which produces highly accurate short-term weather forecasts 10s of minutes in the future for areas as small as $100m^2$. The other application is *Video-on-Demand*, which provides the clients with a wide list of videos to choose from and store them on dedicated servers all around the world. The problem with VoD services is the long tail distribution of their catalogue of videos, where only few videos are popular and a long tail of videos which are requested rarely. We look at two problems in VoD applications due to the long tail distribution of the popularity of videos.

- Long tail popularity distribution characteristic of VoD services reduces the hit rate of the caches used to serve the video requests for VoD services, which increases the traffic on the network. This is particularly true for most popular user generated video service in the world, *YouTube*. In this dissertation, we look at the usage of YouTube related list to improve the efficiency of YouTube caches by a related list reordering approach.

- Video streaming has moved from fixed bit rate streaming to adaptive bit rate streaming, which involves making multiple copies of each video file to adapt

to various client network and device conditions. The long tail characteristic of VoD services increases the amount of videos to transcode and the storage space needed to store the multiple copies which are requested rarely. In this dissertation, we look at the feasibility of online transcoding in the cloud. We present different online transcoding policies to reduce the transcoding workload and maintain the performance of providing adaptive bit rate streaming at the client.

1.1 Scientific Weather Application

The emergence of smart phones has led to the proliferation of a variety of innovative data-driven mobile applications or "apps". One useful application is mobile weather forecasting, which provides hour-to-hour forecasts on a coarse-grained geographical scale. While today's forecasting apps (e.g., my-cast [26], The Weather Channel App [35], AccuWeather [1]) are useful for long-term weather prediction, they do not provide the type of precise, small-scale, short-term predictions necessary for reacting to fast-changing severe weather. For instance, informing mobile users 10-15 minutes in advance of a tornado hitting their exact location would be a major step forward for emergency management systems.

In this dissertation, we present CloudCast, a new application for short-term, location-based weather prediction. CloudCast is an application that generates short-term forecasts based on a user's specific location. If severe weather is approaching the user's location, CloudCast automatically sends personalized notifications. To enable this functionality CloudCast combines two major components. The first component is an algorithm that produces fine-grained short-term weather forecasts (up to 15 minutes in the future) for areas as small as $100m^2$, called Nowcasting [90, 89]. Nowcasting has the potential to personalize severe weather alerts by programmatically transmitting highly targeted, location-specific warnings to mobile devices based on

their GPS coordinates. For example, rather than displaying an hourly forecast for a whole region, a personalized Nowcast is capable of issuing specific close-term predictions, such as whether or not it will rain in a few minutes, for a mobile device's specific location.

The second component is a new architecture that allows the execution of Nowcasting on cloud platforms, such as Amazon's Elastic Compute Cloud (EC2). Nowcasting is compute-intensive, requiring high-memory systems for execution. Hence, we use cloud services to execute Nowcasting algorithm for our CloudCast application. Cloud computing platforms lower the cost of computation by leveraging economies-of-scale. Generating Nowcasts on dedicated infrastructure is economically infeasible due to its enormous computational demands, which scale quadratically with spatial resolution. For instance, a 4x increase in resolution results in (roughly) a 16x increase in computation. As a result, a service that generates Nowcasts, similar to the National Weather Service (NWS) that generates coarse-grained forecasts for every three hours in the future up to three days, requires massive upfront capital costs for servers. Yet, since severe weather is rare, dedicating significant infrastructure to Nowcasting "wastes" server capacity most of the time. Weather services, such as MetService in New Zealand and the U.S. NWS, already invest millions of dollars in server capacity for today's coarse-grained atmospheric modeling and forecasting [12, 23, 29].

The primary reason weather services cite for not leveraging the cloud is data staging costs. We evaluate these costs for the extreme case of Nowcasting, which requires rapid real-time radar data uploads to predict conditions tens of minutes in the future. Recent work has shown the benefits of using cloud computing for data-intensive scientific applications [62, 56]. While these operate on massive data sets uploaded to the cloud *a priori*, their computation typically takes place offline with flexible deadlines, if any. Additionally, scientific applications often reuse the same data set for repeated executions, e.g., searches over a large parameter space, which mitigates the

upfront (time and monetary) cost of uploading data to the cloud. However, many scientific applications exist that require hard deadlines for data processing. Compared to most cloud applications, CloudCast's Nowcasting algorithm has stricter real-time constraints. Timely execution of the algorithm is critical since the Nowcast data has to be made available to end users before it becomes obsolete. For example, in a severe weather scenario Nowcast information can be used to warn the public, as well as guide spotters and other emergency management personnel. Since Nowcasting predicts weather only in the very near-term future (on the order of minutes), it is important that the algorithm produce results fast. For instance, if it takes 12 minutes to generate and disseminate a 15-minute Nowcast, that leaves just 3 minutes for the users to take action.

Our hypothesis is that the connectivity and diversity of current cloud platforms mitigate the impact of staging data and computation latency for Nowcasting, enabling them to perform atmospheric modeling and personalized weather forecasting that CloudCast requires. In evaluating our hypothesis, this dissertation makes the following contributions.

- We propose CloudCast, an architecture that links weather radars to cloud instances, allowing the components in the architecture to request computational and storage resources required to generate forecasts based on real-time radar data feeds as requested from mobile clients.

- We emulate a radar network using PlanetLab sites and conduct extensive bandwidth measurements between each site and cloud instances. We quantify average bandwidth and its variability to determine if the public Internet and today's clouds are sufficient for real-time weather forecasting.

- We analyze the computation time and cost of Nowcasting in the cloud for various instance types offered by the cloud services, to quantify the trade off between

faster execution of the Nowcast algorithm and higher cost that arise from renting more powerful cloud instances.

- We use statistical metrics to analyze Nowcast prediction accuracy for various aggregation levels to ensure accurate results given changing Internet conditions. We show that our system is able to achieve high accuracy using existing networks and clouds for up to 10 minute Nowcasts.

- Finally, we demonstrate CloudCast live using a deployed prototype radar as a proof-of-concept.

1.2 Video-on-Demand Application

The Internet has emerged as an important broadcast medium offering TV shows, radio programs, movies, and the exchange of videos for personal as well as commercial use. The advent of websites such as YouTube, Dailymotion, Metacafe, etc., which offer user generated video content and websites such as Hulu, Netflix, etc., which offer streaming of TV shows and movies, has made the Internet a major source for digital entertainment in the U.S. These video services, which provide content to the users on demand are known as Video on Demand (VoD) applications or services. The growing use of VoD services and popularity of content streaming among users are closely tied to the increasing popularity of broadband Internet connection in homes. The greater adoption of broadband in the U.S has motivated television channels such as NBC and ABC to offer their prime-time programming to online viewers via the media content provider Hulu. In parallel, Netflix, originally a DVD rental company, began to take advantage of the click-and-view streaming of full-length films and television episodes with a subscription service.

In 2012, Internet video traffic made up 57% of the total network traffic in the Internet [20]. This shows that with the popularity of VoD services, the amount of

Internet traffic due to video increases tremendously. The problem with the growth of VoD services is the huge catalogue of videos they offer in their library and the impact of the huge catalogue on the network traffic in the Internet due to videos. As reported in earlier work, the world's most popular video service, YouTube, follows a long tail distribution in the requests made by the users to the videos offered by YouTube. Zink et al. [108, 109] have shown that 77% of the YouTube videos are requested only once over a period of 24 hours, which is true even for Hulu video service as shown in our previous work [71]. This shows that we need effective caching mechanisms to improve the efficiency of caches to improve the hit rate of the caches or edge servers serving these requests all around the world and reduce the amount of backbone traffic due to the long tail popularity distribution of videos in the VoD services which are requested rarely.

Video streaming over the Internet has boomed over the past years, with HTTP as the de-facto streaming protocol. According to the latest Sandvine report [18], during peak hours (8 PM to 1 AM EDT), over 50% of the downstream US Internet traffic is video content. The diversity of client devices capable of playing online videos has also seen a sharp increase, including a variety of smartphones, tablets, desktops, and televisions. Not surprisingly, video streaming that once meant playing a fixed quality video on a desktop now requires adaptive bit rate (ABR) streaming techniques. In order to support ABR streaming, the video or content provider should create multiple copies of a single video file. This process of creating multiple copies of a single video file into multiple quality levels is called "transcoding". The long tail popularity distribution characteristic of VoD services presents an unique problem for video and content provider because transcoding videos which are requested rarely or only once into multiple bit rates wastes transcoding resources and also storage space.

In this dissertation, we provide solutions to the issues posed by the long tail popularity distribution of VoD services. We present an approach to use the related

list provided by VoD services to reduce the amount of backbone bandwidth due to long tail distribution of video service. We look at YouTube as a case study as YouTube is the world's largest user generated video service hosting billions of videos and serving millions of clients everyday. We also look at the feasibility of online transcoding in the cloud to reduce the amount of transcoding workload and storage space needed to transcode videos from the long tail popularity distribution of VoD services. We also present transcoding policies and prediction models to maintain the performance of ABR streaming at the client with online transcoding.

1.2.1 Improving the Efficiency of YouTube Caches

YouTube is the world's most popular Internet service that hosts user-generated videos. Each minute 72 hours of new videos are uploaded to YouTube. Viewers can choose from hundreds of millions of videos and over 4 billion hours of videos are watched each month on YouTube. According to [40], in 2011, YouTube had more than 1 trillion views or around 140 views for every person on Earth. Consequently, these numbers lead to a huge amount of network traffic, and Google (the owner of YouTube) maintains substantial infrastructure to provide reliable and well-performing video streaming service. For example, according to [19], on August 27 2012 at 8PM, the United States YouTube traffic made up more than 60% of the global Internet traffic. Google's approach to tackle these challenges is a network of caches that are globally distributed. With the aid of the caches, latency on the viewer's side and overall network utilization can be reduced.

Unfortunately, effective caching is much harder in the case of YouTube in comparison to other video streaming services like Netflix or Hulu. This is caused by several facts. First of all, services that offer professionally produced content like movies and TV shows provide an online library on the order of several 10,000s of titles, a number that is much lower compared to the hundreds of millions of videos offered on YouTube.

Second, the providers of purely professionally produced content determine when new content will be made available and, thus, can much better schedule the distribution of content into caches.

YouTube's dilemma, in terms of caching, is the fact that the popularity distribution of videos is a long-tail distribution. For the large number of videos in YouTube, this means that every cache that is limited to storing a small portion of the total number of videos will essentially perceive accesses to the tail as random choices, impossible to distinguish from a uniform distribution. This characteristic has already been shown in earlier studies. For example, in the trace-based characterization of YouTube traffic in a campus network [110], 70% or more of the overall videos are only requested once within 24 hours. In [47], similar results are obtained by a global analysis of metadata made available by YouTube. This characteristic leads to the fact that many of the requested videos will not be available in a cache. This problem has been officially recognized by YouTube [39], and the current approach is to increase the number of caches that serve content from the tail and fill these caches through prefetching.

In this dissertation, we propose a different caching approach that makes use of YouTube's recommendation system to increase the efficiency of its caches, since it is impossible to predict what a viewer is most likely to watch next. The general idea is to influence the video selection behavior of a viewer based on the content of the cache that serves the viewer. Each YouTube video web page offers a list of recommended videos, next to the video a viewer has selected (see Figure 1.1). In YouTube's own terms, this is described as the *related video list*. Earlier studies [105, 65] have already shown that users make significant use of the related video list. This means that, after watching the initially selected video, the viewer chooses to watch a video that is offered on the related video list next. Figure 1.2 shows a comparison of these result that were derived from five different data sets. It is important to mention that the

Figure 1.1. This screenshot shows a YouTube video page and the location of the related video list on that page.

results from [105] are obtained from two data sets of global meta information (taken at different points in time) that is offered by YouTube for each individual video. The results for [65] are obtained from two network traces taken at a campus gateway. From these results, we conjecture that both, on a global and regional level, $\sim 30\%$ or more of video requests are made by the viewer by selecting a video from the related video list.

To exploit the fact that viewers frequently choose videos from the related list, we present an approach in which the cache modifies the related video list that is sent from the server through the cache to the client. This related list modification is based on a stable sort [67] the cache performs on the list of related videos that uses cache hit as its primary sorting criteria. The set of cached videos are moved to the top of the related list, while the set of uncached videos forms the rest. The order inside each set remains unchanged. The cache changes the order of the related video list because, based on our analysis of the position of video requested from the related list provided, it is more likely that a viewer will select a video from the top of the related video list.

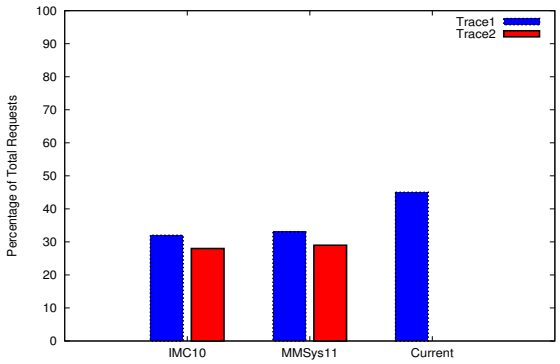


Figure 1.2. Percentage of videos selected from related video list as reported in earlier studies (IMC10 [105], MMSys11 [65]) and obtained from an analysis of the trace used in this dissertation.

To investigate the feasibility of our approach, we perform an analysis on the chains created by user behavior in selecting YouTube videos from the related list. Chains are created when a user consecutively selects videos from the related list until she changes to choose a video by other means (e.g., a search). We also perform a PlanetLab based study to analyze the global behavior of loop creation. Loops are created when a user selects videos from the related list consecutively until she selects the initial requested video again. This selection is either based on the position of the video on the related list, or randomly chosen from the related list. Our results show that YouTube creates its related list from a small pool of videos which leads to a lot of potential to related list caching and reordering based on the content in the cache.

In addition, we perform a measurement study from our campus network to determine the origin of videos that are requested from the related list. The results of this study indicate that YouTube uses a three-tier cache hierarchy to serve videos and that the origin is chosen in a way that achieves load balancing between the different levels in the caching hierarchy. Preference to the top videos of the related list for a requested video is not given. By using our approach of the related list reordering at the cache, we recommend changing the behavior of the caching technique to take advantage of user behavior in selecting a video from the top order of the related list

by serving the videos from the nearest cache to reduce latency in serving the videos to the user.

In order to verify our hypothesis that users select from the top of the related list even when provided with a reordered related list, we perform a study with 40 actual users in which we present users a randomly reordered list and track the video they choose from that list. The results from our study strongly indicate that YouTube users prefer to select the next video they want to watch from the top of the related list even when provided with a reordered list.

To evaluate the proposed related list reordering approach, we perform a study based on YouTube traces obtained from a university campus network. Based on these traces, we show that first, users indeed choose top ranked videos from the related video list and second, with our proposed approach, the cache hit rate increases from about 2 to 5 times the original hit rate. This increase in cache hit rate due to reordering of related list reduces the bandwidth consumption on the uplink from the level 1 cache to higher level caches or the server by 5.12% to 18.19% and reduces the latency in serving the videos requested to the user.

Finally, we look into the feasibility of using elasticity of cloud services to avoid the long tail popularity distribution of YouTube videos. Research has shown that YouTube uses a 3-tier caching hierarchy to serve the videos to the users all around the world [44, 43]. In this dissertation, we present CloudTube architecture, which hosts videos based on the global popularity of videos in the same 3-tier cache hierarchy used by YouTube in cloud instances. We use an abstraction of blinking [91] to switch the instances serving the less popular videos to low power inactive state when they are not serving any video requests. We measure the hit rate based on our video hosting strategy and the *uptime* of cloud instances hosting less popular videos. Uptime refers to the amount of time a cloud instance spends in high power active state to serve video requests.

We evaluate this video hosting strategy using our YouTube video requests dataset. Based on our analysis of user watching pattern of YouTube videos, we suggest that placing the first chunk (60 seconds) of all videos and highly popular videos closer to the client and the less popular long tail videos on secondary or tertiary cache cloud instances increases the hit rate of the high popular cloud instance by 90% and the uptime for medium and low popularity cloud instance decreases by almost 20%, which in turn reduces the power required to serve the contents by almost 20%.

This dissertation makes the following contributions to improve the efficiency of caches and reduce the energy in serving videos from the long tail popularity distribution of VoD services especially YouTube.

- We present a YouTube reordering approach, based on the content in the cache, which increases the hit rate of the caches.
- We perform a real user study by reordering the related list to validate our hypothesis of users requesting from the top half of the related list.
- We simulate the reordering approach using passive YouTube trace and show that the related list reordering approach has the potential to improve the cache hit rate by 2 to 5 times, which in turn reduces the network bandwidth and latency in serving videos requested.
- We present the CloudTube architecture, which uses cloud services to reduce the traffic due to long tail popularity of YouTube videos by using YouTube's 3-tier cache hierarchy to store videos based on global popularity.
- The analysis of our CloudTube architecture suggests that placing the first chunk (60 seconds) of all videos and highly popular videos closer to the client yields higher hit rate and reduces the uptime and power required to serve low popular videos, which constitute the long tail in VoD services such as YouTube.

1.2.2 Online Transcoding

Video streaming has moved from traditional fixed video quality streaming to adaptive bit rate (ABR) streaming depending on different network and client device characteristics. A key goal of ABR streaming is to avoid freezes during the play out of the video. Such freezes known as "rebuffering" are typically caused by insufficient bandwidth between the source and the client, causing the client's video buffer to drain quickly. Once the client's video buffer reaches empty a rebuffering event occurs. Rebuffering is known to have a major adverse impact on a user's video viewing experience [69]. ABR streaming requires that each video segment be encoded in different quality versions: lower quality versions use a lower bit rate encoding and higher quality versions use higher ones. The process of creating multiple bit rate versions of a video is called *transcoding*. Once each video is transcoded into multiple bit rates, ABR streaming allows the client to choose an appropriate quality version for each video segment based on the available bandwidth between the source and client. Thus, the client can switch to a lower quality video segment when the available bandwidth is low to avoid rebuffering. If more bandwidth becomes available at a future time, the client can switch back to a higher quality version to provide a richer experience.

A video provider[1] wanting to use ABR streaming must first complete the transcoding process before their videos can be delivered to their users. To support ABR streaming, a video is divided into short segments (usually of several seconds duration) and each of these segments is transcoded into different bit rates, where each bit rate represents a different quality level. According to Netflix, the vast number of today's codec and bit rate combinations can result in up to 120 transcode opera-

[1]We use the term video provider to denote any enterprise that provides video content for their users, including movies (e.g., NetFlix), news (e.g., CNN), entertainment (e.g., NBC) and sports (e.g., FIFA soccer).

tions before a video can be delivered to all client platforms [27]. Thus, transcoding is resource intensive requiring significant computing and storage resources.

In the traditional model, transcoding is first performed by the video provider (say, NBC or CNN) and the transcoded content is then uploaded to a content delivery network (say, Akamai or Limelight) that actually delivers the videos to end-users around the world. However, this model requires a major investment of IT resources on the part of the video provider to perform the transcoding. A common emerging alternative is for video providers to outsource *both* the transcoding and delivery of videos to a content delivery network (CDN). Procuring transcoding as a cloud service from the CDN eliminates the expense of procuring and operating transcoding equipment for the video provider. Thus, increasingly CDNs such as Akamai [3] perform both transcoding and delivery of the videos. The convergence of transcoding and delivery enables new possibilities for reducing the resources needed for transcoding and is the focus of our work.

CDN Transcoding Architecture. A typical CDN offering transcoding and delivery services operates a storage cloud for storing videos, a transcoding cloud for performing the transcoding, and an edge server network for caching and delivering the video segment to users. Transcoding and delivering videos entail the following steps. To publish a new video, the video provider uploads a single high quality version of that video to the storage cloud of the CDN. Then, the CDN uses its transcoding cloud to transcode the video to all the bit rates requested by the video provider and stores the transcoded output in the storage cloud[2]. The video provider then makes the new video available to users, say by publishing it on their web page. As users start watching the new video, the requested video segments in the right quality levels are downloaded by the edge servers from the storage cloud and delivered to the users.

[2]The formats and quality levels a video will be offered in is usually agreed upon in a service level agreement (SLA) between the CDN and the video provider.

The CDN often offers an SLA on how quickly a newly uploaded video is available for access by users. A typical SLA guarantees that a video of duration D is available within time D/s for users to download and is termed an `1/s` SLA, e.g., a `1/2` SLA guarantees that a 30-minute video uploaded at the time t is available for users at time $t + 15$ minutes.

Why understanding delivery helps transcoding? The convergence of video transcoding and delivery offers rich possibilities for optimization. *Understanding the interplay of video transcoding and video delivery to reduce the transcoding work is the main focus of our work.* There are two motivating reasons why understanding video delivery, i.e., understanding what parts of which videos are watched by users, can help optimize the transcoding process.

1) It is known that the popularity distribution of videos is heavily long tailed [110, 47], i.e., a substantial fraction of the published videos are requested only once or not requested at all. Transcoding video segments of unpopular videos that are never requested is a waste of computation and storage resources that can potentially be saved by using more intelligent transcoding mechanisms.

2) It is known that the video segments that correspond to the first part of a video is watched more than the later parts of the video, as users often abandon videos midway [70, 57]. This suggests that the early parts of the videos are more likely to be watched in more bit rates than the later parts. Thus, understanding the characteristics of what parts of a video are actually delivered to users can be of value to the transcoding process.

Offline versus Online Transcoding. We refer to the traditional approach where transcoding is performed *before* the delivery process begins as *offline transcoding*. Note that offline transcoding is oblivious to what video segments are delivered to (and watched by) users, as it happens before the delivery of the video begins. In contrast, we propose *online transcoding* of video segments as an alternative to offline

transcoding. In the online approach, transcoding is performed in real-time and only performed if a video segment is requested by a client in a specific quality version that has not already been created in the past. Note that online transcoding is tightly integrated with the delivery of the videos. Offline and online transcoding are two extremes and a number of *hybrid* transcoding approaches that combine aspects of both are possible. Specifically, an `x/y` transcoding policy transcodes the first `x`% of the video to *all* the desired bit rates in an offline fashion before delivery begins. Further, it transcodes the remaining `y`% of the video segments to only those bit rates that are (or, likely to be) requested by the user in an online fashion.

Contributions. This dissertation makes the following contributions to reduce transcoding workload for VoD services.

- We propose new online and hybrid transcoding policies and analyze the workload generated for these approaches using trace-driven simulations. Our extensive simulations use video request traces from one of the world's largest video CDNs. Our analysis shows that the total and peak workload[3] required for online and hybrid transcoding are an order of magnitude lower than those for the traditional approach of offline transcoding. Our 1Seg/Rest hybrid policy decreases workload by 95% as compared to the offline policy.

- We show that the peak workload induced by transcoding policies increases as the transcoding SLA becomes more stringent. In particular, a "faster-than-real-time" SLA has prohibitively higher peak workload than a more lenient SLA, e.g., a `1/4` SLA has four times the peak as the `1/1` SLA taking four times more resources.

- We present a Markov model approach to predict the quality level (i.e., bit rate) of the next video segment that is most likely to be requested by the client ahead

[3] *workload* refers to the amount of bytes to transcode.

of time. We derive a prediction model that results in an average prediction error of 0.3%. We show how to use this predictive model to perform transcoding of video segments before it is likely to be requested by the client, reducing the possibility of video rebuffering.

- We derive the impact of transcoding policies on the *rebuffer ratio* that equals ratio of the time spent in a rebuffering state and the video length. We analyze several online and hybrid approaches and show that our 1Seg/Rest hybrid policy achieves an average rebuffer ratio of 0.09% and a maximum rebuffer ratio of about 0.2% with our prediction model. Thus, our 1Seg/Rest policy achieves a workload reduction of 95% without a significant impact on the viewing experience as measured by the rebuffer ratio.

1.3 Dissertation Organization

The rest of the dissertation is organized as follows: Chapter 2 provides the related works in the area of scientific applications in the cloud, techniques to improve the hit rate of YouTube caches and online transcoding in the cloud. Chapter 3 presents our CloudCast application architecture and its components. It presents the cloud services network and computation feasibility analysis for CloudCast. We also present the analysis between the commercial and research cloud services for CloudCast.

Chapter 4 presents an overview of the related list reordering approach and the analysis for the feasibility of our reordering approach. Chapter 5 presents an in depth analysis of related list differences presented to the user by YouTube and look at the impact of such related list differences on caching. Chapter 6 presents the related list reordering approach, user study and the results from a simulation based on passive YouTube traces. Chapter 7 presents our CloudTube architecture of hosting YouTube videos on cloud instances.

Chapter 8 presents the online transcoding architecture and policies. It also presents the dataset used and the workload analysis for different online transcoding policies. Chapter 9 presents the prediction models used in our online transcoding policies and performance analysis of our online transcoding policies. Chapter 10 summarizes the work presented in the dissertation and presents few ideas for future work. Chapter 11 provides the list of publications on the work presented in this dissertation.

CHAPTER 2

RELATED WORK

This chapter provides the related research in the area of scientific applications in the cloud, techniques to improve the cache efficiency of VoD services, especially YouTube and online transcoding in the cloud.

2.1 Scientific Application in Cloud

A substantial amount of research has been carried out to investigate the feasibility of running scientific applications in commercial clouds such as Amazon's AWS. Hazelhurst examines the performance of the bioinformatics application WCD [58]. Deelman et al. provide details of performance and storage costs of running the Montage workflow on EC2 [53]. The High-Energy and Nulclear Physics (HENP) STAR experiment has examined the costs and challenges associated with running their analysis application in the EC2 cloud [63, 64]. Ramakrishnan et al. have examined the usefulness of cloud computing for e-Science applications [86, 76]. In addition, standard benchmarks have also been evaluated on Amazon EC2. Rehr et al. show that Amazon EC2 is a feasible platform for applications that do not need advanced network performance [88]. Wang et al. [98] study the impact of virtualization on network performance in the cloud. Ramakrishnan et al. perform a comprehensive comparison of the performance of EC2 with HPC platforms, using real applications representative of the workload at a typical supercomputing center [60].

The Nowcasting algorithm used in this dissertation is based on the Nowcasting system operating in the CASA Distributed Collaborative Adaptive Sensing network.

[89, 90] provide details on the Dynamic and Adaptive Radar Tracking of Storms (DARTS) Nowcasting method. The DARTS Nowcasting falls under the area-based Nowcasting methods which estimate a motion vector field over the entire radar coverage domain.

To the best of our knowledge, the work presented in this dissertation is the first to look into the feasibility of commercial and research clouds for real-time application of weather forecasting and the first work to introduce an application for short-term weather prediction in the cloud.

2.2 Long Tail Popularity Distribution

Long tail popularity distribution is a distribution of the catalogue of items offered where only a few items are highly popular followed by a long tail of items in the catalogue which are visited or requested very rarely and sometimes not at all. In this dissertation, we focus on the problems incurred by the long tail popularity distribution in VoD services. Long tail popularity distribution has been found in various other streams or categories similar to VoD services which offer large catalogue of items to the users to choose from. In this Section, we provide few related works in the area of long tail popularity distribution from the business perspective.

Tucker et al. [97] have studied the long tail vs steep tail effect from the management perspective and tried to answer whether the online popularity of items affect the choice of customers in selecting a particular item. The authors use data from a field experiment with a website that lists wedding service vendors and find empirical evidence that a steep tail exists. The most popular vendors become more popular when customers can easily observe previous customers click-through behavior. They also suggest that steep tail complements long tail by attracting customers from less popular vendors to the most popular ones. Their findings suggest that popularity

information can serve as a powerful marketing tool that facilitates product category growth.

Ratkiewicz et al. [87] provide a quantitative, large scale, temporal analysis of the dynamics of online content popularity in two massive model systems: the Wikipedia and an entire countrys Web space. They find that the dynamics of popularity are characterized by bursts, displaying characteristic features of critical systems such as fat-tailed distributions of magnitude and interevent time. Anita Elberse [54] provides a business review on whether to invest in long tail or not. She looks at the sales of books on display by different vendors based on popularity in the digitized world and conclude that the popularity leads to more sales and increase in popularity of the same books. The authors suggestion based on her analysis is to acquire and manage their customers by using the most popular products. This suggestion can be used to any e-commerce website offering customers with wide array of suggestions to choose from but very little space to show all of them.

Researchers [46, 73] have looked at the long tail popularity effect in the music industry and looked at recommendation systems to provide users with novel and more relevant recommendations from the tail of the popularity distribution. In this dissertation, we look at the long tail popularity in VoD services as the network traffic generated by these services is much higher than any other application or service with a long tail popularity distribution. We look at techniques to improve the caching efficiency, energy efficiency and reduce redundant use of resources to support the catalogue of videos offered by the VoD services. In the next section, we provide related work in the areas of caching of videos in VoD services and transcoding of videos in the cloud.

2.2.1 Caching and Migration of VoD services to Cloud

The use of proxies and caches to reduce traffic from VoD services has been intensively studied in related work. In [47] and [110], trace-driven simulations were performed to investigate the effectiveness of caching for YouTube videos. Although the traces for both studies were different, the results showed that caching can reduce server and network load significantly. Both studies did not consider reordering the related video list to increase the efficiency of the cache.

Besides caching, YouTube's recommendation system (the related video list) has also been studied in related work. In [105], Zhou et al. analyzed two data sets (one directly crawled from YouTube and the other one a trace from a campus network) to investigate if the position of a video in the related list has significant impact on it being selected by the viewers. The results of this analysis show that a large percentage of viewers select videos they watch from the related list (see Figure 1.2). In follow on work [106], Zhou et al. perform further analysis of YouTube's recommendation system based on global statistics for YouTube videos. The authors show that the click through rate is proportional to the position of the video on the related list (the higher the position of the video on the list, the higher the chance that it will be selected by the viewer). As in [105], the results confirm our assumption on related list usage and support the position centric related list reordering approach. While the goal of the work presented in [65] was to show how the prefetching of prefixes from videos on YouTube's related list can improve caching, it also shows that the related list is often used by viewers to select a video (see Figure 1.2). In contrast to the work we present in this dissertation, no modification of the related list at the cache is proposed.

Cheng et al. [50] measured the YouTube video graph created by related video links and found that the graph has a large clustering co-efficient and exhibits the small world property. A simulation-based evaluation of a P2P video sharing systems showed that if users use the related video list to browse videos, the percentage of source peers

that have the requested video in their cache is high. Cheng and Liu [49] also proposed a P2P video sharing system to reduce YouTube server load and suggested using prefetching based on YouTube's related video list at the clients of a P2P system to provide smooth transition between videos. Their evaluation was based on emulated user browsing pattern. The evaluation of their approach showed that it performs significantly better (55% hit ratio) comparing with a random prefetching approach (nearly 0% hitrate).

In [44] and [43], Adhikari et al. uncover the overall architecture of the YouTube video distribution system. They show that YouTube uses 3-tier cache hierarchy to serve the requests from around the world. While our results agree with their findings, their work does not investigate which of the videos of the related list are transmitted from which cache level. In addition, their work is not concerned with improving the efficiency of YouTube caches.

Chakareski [48] takes a more general view on recommender systems (called catalogues in his case) to develop an optimization framework that has the goal to reward providers if an item selected by a customer can be provided immediately and penalize them if the provision is delayed. Similar to our case, the author shows that optimizing for immediate access to items at the beginning (or top) of a catalogue is beneficial and optimizes the reward for the provider. The paper does not reveal how this immediate access can be provided, and we see our work as complementary part since it addresses content provision. In addition, our study is based on actual user behavior obtained from trace data.

With the rise in popularity of content cloud platforms such as Amazon CloudFront [5] and Azure CDN [10], researchers have looked at the migration of VoD services to cloud services. Li et al. [75] have looked at the advantages of partial migration of VoD services to content clouds for cost saving and design heuristics to decide the update of cloud contents. They show that hybrid cloud-assisted VoD deployment save

up to 30% bandwidth expense compared with serving from own server deployment. Qiu et al. [84, 85] presents an optimization framework for dynamic, cost-minimizing migration of VoD services into hybrid cloud infrastructure that spans geographically distributed data centers. Wu et al. [102] deploy a VoD application on an IaaS cloud containing a single data center. However, none of these works look into elasticity of cloud services for YouTube video service and do not use any cache modification mechanisms to increase the hit rate and reduce the long tail of VoD services.

To the best of our knowledge, the work we present in this dissertation is the first that investigates how reordering of information provided by a recommendation system based on a cache's content can improve its performance and usage of cloud services to host YouTube video content to reduce the long tail effect of YouTube video service.

2.2.2 Online Transcoding in the Cloud

Many researchers have investigated the problem of online transcoding. Most of the research has been focused on scheduling policies for transcoding and transcoding in the cloud environment. In the commercial sector several companies have recently started to offer cloud-based video transcoding as a service. Amazon's Elastic Transcoder [6] executes transcoding jobs using Amazon's Elastic Compute Cloud (Amazon EC2 [8]) and stores the video content in Amazon's Simple Storage Service (Amazon S3 [7]). Amazon's Elastic Transcoder enables customers to process multiple files in parallel and to organize their transcoding workflow using a feature called transcoding pipelines. It manages all aspects of the transcoding process transparently and automatically. Zencoder [11] is another video transcoder service from Brightcove, a company that offers a cloud-based online video platform. Along with typical video transcoding services, Zencoder also supports live video transcoding in the cloud. EncoderCloud [13] provides similar web-based "pay-as-you-go" service by using resources from other cloud service providers (e.g., Amazon EC2 and

RackSpaceCloud [32]). However, they offer a different pricing policy than other cloud service providers, charging by the volume of the total amount of source video transferred in and encoded video transferred out. These services provide the capability of video transcoding in the cloud, but the transcoding scheduling mechanism is non-transparent to end-users. While these services are quite popular there is only little information how much resources have to be provided by either Amazon or Brightcove. In addition, it is not possible to measure the time it takes to perform a transcoding request. This information is essential to determine if the approach of online transcoding is feasible.

With transcoder services available in the cloud, researchers have looked at different scheduling policies for scheduling video transcoding in the cloud. Ma et al. [80] have proposed a dynamic scheduling methodology for video transcoding in a cloud environment, with the goal to improve user experience by minimizing the delay of online transcoding. Li et al. [78] developed a transcoder in the cloud which utilizes an intermediate cloud platform to bridge the format/resolution gap by performing transcoding in the cloud. Their cloud transcoder takes CPU utilization into account to schedule video transcoding tasks. Ko et al. [68] looked at the amount of resources and cache space required for online real time transcoding. Kllapi et al. [66] presented an optimization framework for scheduling data flows to minimize completion time, minimize the cost and determine the trade-off between completion time and cost incurred. Li et al. [77] proposed parallel video encoding strategy based on load balance factor and [103] proposed a cost optimization framework based on parameter tuning combining bit rate and encoding speed. However, none of these works have investigated the combined space of online transcoding and video delivery. In addition, none of them have analyzed how much resources such online transcoding in a large CDN requires.

The works closest to the one presented in this dissertation were presented by Zhi et al. [100] and Shin et al. [93]. The authors of [100] have investigated the feasibility of online transcoding system in the cloud. Their approach is based on online transcoding and geo-distributed video delivery based on user preferences for CDN regions and regional preferences for certain quality versions of videos. However, their work does not provide a prediction model to transcode future segments and measure the performance of video delivery for real time systems. This dissertation provides a Markov prediction model to predict the future segments to transcode ahead and improve performance at the client.

The authors of [93] present a hybrid transcoding technique to provide users with different QoS VoD services. They present a mathematical model to pre-transcode popular videos and online transcoding of less popular QoS video requests. However, the authors perform their analysis on the assumption that they already know the popularity of the videos and they online transcode to only three different bit rates. This dissertation presents hybrid transcoding without assuming the popularity of the videos and transcode part of the video offline. This is different to Shin et al.'s approach of transcoding the whole video to one particular bit rate.

CHAPTER 3

CLOUDCAST

This chapter provides an architecture overview of our CloudCast application, a real time short-term weather application on the cloud and provides a detailed description of each block used in our architecture. We provide an in-depth analysis of the network and computation feasibility of our CloudCast application. We provide an overview of the cloud services used in our analysis, the measurement performed to analyze the network capabilities of the cloud services for our application and finally the computation analysis performed on cloud services using canned and live weather data to show the feasibility of cloud services for real time scientific application of weather forecasting.

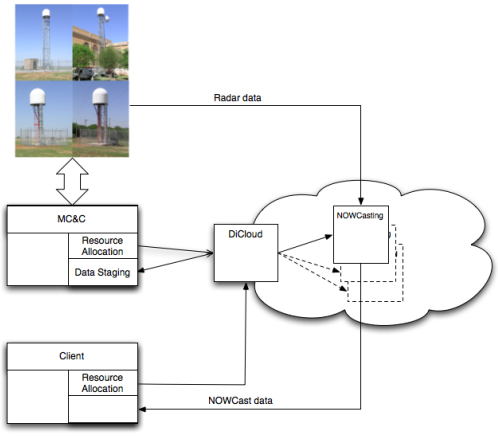


Figure 3.1. Overview of the CloudCast system architecture.

3.1 Architecture Overview

Figure 3.1 shows the components of our CloudCast architecture composed of the MC&C, which controls the scanning of the radars; DiCloud, an environment that allows users to conduct data-intensive experiments in commercial clouds (Amazon EC2 in this case); and a short term weather forecasting algorithm called Nowcasting.

We designed this architecture for a new type of radar sensor network developed by the center for Collaborative Adaptive Sensing of the Atmosphere (CASA) [81, 107]. CASA has developed this new type of radar based weather observation systems to better observe the lowest part of the atmosphere and in turn improve the detection and prediction of severe weather. We have chosen CASA since we have full access to its radars (compared to the operational radars run by the National Weather Service (NWS)) which allows us to evaluate CloudCast with real radars and weather. While we evaluate CloudCast by using the CASA system we designed its architecture in a way that will allow its use with other radar networks such as, e.g., NEXRAD.

The CloudCast architecture allows the initiation of Nowcasts in the cloud in two different ways. The first way starts a Nowcast the moment weather enters the radar networks area. To enable this functionality, the CloudCast architecture makes use of Meteorological Command and Control (MC&C). The MC&C includes several detection algorithms that run on radar data and can detect certain weather phenomena (e.g., rain). These detections can be used to initiate a Nowcast in the cloud. The alternative way to initiate a Nowcast is more end-user centric. In this case, a Nowcast process is started if the conditions in the case described above are met and if a mobile user has requested a Nowcast service for a specific area. E.g., one can imagine that a user has subscribed to a Nowcast service and would like to obtain a Nowcast if weather is in the vicinity of their current location. In this case, a Nowcast will only be initiated if the MC&C indicates weather and if a user who has registered for a Nowcast service is in that area.

In the following, we provide a detailed description of each of the building block of our CloudCast architecture.

3.1.1 MC&C Architecture

Meteorological Command and Control (MC&C) [107] is the control part of the CASA network that determines how the radars will scan the atmosphere in each 60 second heartbeat. The scanning of each radar is determined by several factors, including 1) detections from data obtained in present heartbeat, 2) historical detections from earlier heartbeats, and 3) end-user policies. The MC&C architecture takes these different factors into consideration to determine how to control the radars. For our CloudCast approach we are making use of the detection features the MC&C offers. For example, rain sensed by the radars will be detected by the algorithms in the MC&C. CloudCast uses the information to determine when to initiate Nowcasts in the cloud. Thus, cloud-based Nowcasts are automatically initiated and halted without user intervention.

3.1.2 DiCloud

DiCloud [59] is a platform that enables controlled access to EC2 cloud resources. Though cloud services such as Amazon EC2 provide details of the usage for the operations carried out in the cloud, they perform this usage tracking on a per account basis (which may have multiple users) rather than individual application or user basis. For example, if one envisions that an entity would offer Nowcasting (executed in the cloud) as a service to mobile users, all that Amazon would provide the entity with would be the total cost for the execution of Nowcasts in the cloud. Based on this information it could not be determined how much it costs to execute an individual Nowcast for a mobile user. DiCloud on the other hand enables tracking the costs that occur by executing individual Nowcasts and, thus, the final receiver of the Nowcast data (e.g., mobile client) can be precisely charged. The DiCloud server communicates

with EC2 to track each operation carried out in the DiCloud console. Since the DiCloud console is scriptable, pre-defined scripts can be initiated from the MC&C to start or stop a Nowcast instance in the cloud.

3.1.3 Nowcasting

Nowcasting [90, 89] refers to short term (0 - 30min) weather forecasting. Nowcasting is an algorithm for the prediction of high impact weather events, such as flood-producing rainfall, severe storms, and hail, in a specific region with sufficient accuracy within a time frame such that appropriate actions can be taken to effectively mitigate the loss of life and property. Since Nowcasting is a short term weather prediction system, its applications involve warning decision support for detection of potentially severe weather. Its performance is typically measured in terms of categorical yes/no (e.g., rain/no-rain) detection relative to a predetermined measurement threshold representative of a desired threat. This model of measuring performance is well-suited for our Nowcasting application where the ability of the Nowcasting algorithm to predict a sufficiently high reflectivity value in a given region is important for end-user emergency decision support.

In order to measure the quality of a Nowcast we make use of three statistical metrics used by NWS to assess warning programs [101]. The three metrics are False Alarm Rate (FAR), Probability of Detection (POD) and threat score or Critical Success Index (CSI) and there definition is as given below:

$$FAR = \frac{\text{false alarms}}{\text{hits + false alarms}} \tag{3.1}$$

$$POD = \frac{\text{hits}}{\text{hits + misses}} \tag{3.2}$$

$$CSI = \frac{\text{hits}}{\text{hits + misses + false alarms}} \tag{3.3}$$

As shown in [90] these metrics can be used to determine the quality of a Nowcast. One way to assess the quality of a Nowcast is to calculate the three metrics mentioned

above by comparing a Nowcast for time t with the actual observation at time t. For example, in the case of a five minute Nowcast one would calculate FAR, POD, and CSI by comparing the Nowcast for time t (created at $t - 5$ minutes) with the actual observation at t. The actual comparison is performed as follows. A *hit* is defined as the case in which the pixels for both the image created from the Nowcast and the image created from the actual observation are either both active or inactive. Where *active* describes a pixel value that is equal or greater than a predefined threshold level. Similarly, *inactive* is defined as a pixel value being below the predefined threshold level. A *false alarm* is defined as the case where an *active* pixel is presented by the Nowcast image, while an *inactive* pixel is presented by the actual observation image. The opposite case defines a *miss* in the above equations.

In Section 3.3, we use the above metrics to analyze the Nowcasting case study for two different weather conditions for various data aggregation factors in the cloud.

3.2 Network Capabilities of Cloud Services

This section provides an overview of the cloud services considered for the analysis of our CloudCast application. We present the measurements performed to verify the network capabilities of cloud services for our real time application and present the results of our analysis.

3.2.1 Cloud Services

We have chosen four cloud services for the network capability analysis for our real-time application of short-term weather forecasting. We have considered two commercial cloud services—Amazon's EC2 and Rackspace Cloud Hosting—as well as two research cloud testbeds—GENICloud and ExoGENI cloud. In this section, we give a brief description of these cloud services before explaining our measurement methodology in Section 3.2.2.

Elastic Compute Cloud (EC2). Amazon's Elastic Compute Cloud (EC2) [8] is a cloud service which provides resizable compute capacity to execute applications on demand. Amazon EC2 provides a variety of services including cloud servers, storage, Virtual Private Cloud, and CloudWatch. Amazon provides an easy-to-use web service interface which allows users to obtain and configure cloud resources at any of Amazon's AWS data centers. It provides users with complete control of their computing resources and lets users run applications on Amazon's computing environment. Amazon EC2 reduces the time required to obtain and boot new server instances to minutes, allowing users to quickly scale capacity, both up and down, as their computing requirements change.

EC2 provides on-demand resources with pricing depending on the type of resources used and the duration of the usage. The cost of using commercial cloud services also depends on additional factors such as the amount of I/O performed and the amount of storage used, both of which can incur significant costs for researchers using cloud resources. Wang et al. [99] provide a list of example applications that can be executed on Amazon's Elastic Compute Cloud (EC2).

Rackspace Cloud. Rackspace Cloud [32] is one of Amazon's competitors in the area of commercial cloud hosting. Rackspace offers services including cloud servers, cloud storage and cloud-based website hosting. Cloud servers are available in eight different sizes (with respect to available RAM and disk space) and support a variety of Linux and Windows operating systems. In [61] and [74], the authors provide a brief description of Rackspace and compare its services with other cloud providers.

GENICloud. GENICloud [104, 45] is an open source research cloud testbed which is based on the Slice-Based Facility Architecture (SFA) used by PlanetLab [51]. It supports the management of individual VMs or clusters of VMs. GENICloud uses the Eucalyptus [82] open source cloud platform as a base and federates it with SFA to provide a slice-based architecture to acquire cloud instances (virtual machines) as

slivers, similar to acquiring virtual machines on PlanetLab. GENICloud as a platform consists of a small set of nodes at various sites connected internally to provide a cloud testbed for trusted researchers. The GENICloud resources can be acquired using the Sface [33] GUI providing valid credentials, or a Web based GUI similar to the one for PlanetLab.

ExoGENI Cloud. ExoGENI cloud [15] is a software framework and an open-source cloud platform, which allows users to programmatically manage a controllable, shared substrate. Based on this substrate, researchers can create their own cluster infrastructure by combining servers, storage, and network links in an arbitrary fashion. An ExoGENI deployment is a dynamic collection of interacting control servers that collaborate to provision and configure resources for each experimenter according to the policies of the participants. ORCA (ExoGENI's control framework) helps to provision virtual networked systems via secure and distributed management of heterogeneous resources over federated substrate sites and domains. ORCA allows users to create global topologies of nodes connected via layer 2 QoS-provisioned links. Based on these features ExoGENI cloud offers a variety of opportunities for experimentation and research and also for developing new resource control and management policies via plugins. The ExoGENI cloud, similar to GENICloud, uses a slice-based architecture on top of OpenStack [30]. ExoGENI gives researchers more flexibility than other research clouds, as well as commercial clouds, since it allows them to *i)* create their own network topology for a compute cluster, and *ii)* choose between several geographically distributed clusters.

3.2.2 Measurements and Results

The previous section provides an overview of the cloud services we have considered for our analysis. In this section, we investigate the network performance of the cloud services for our real-time scientific application of weather forecasting by performing

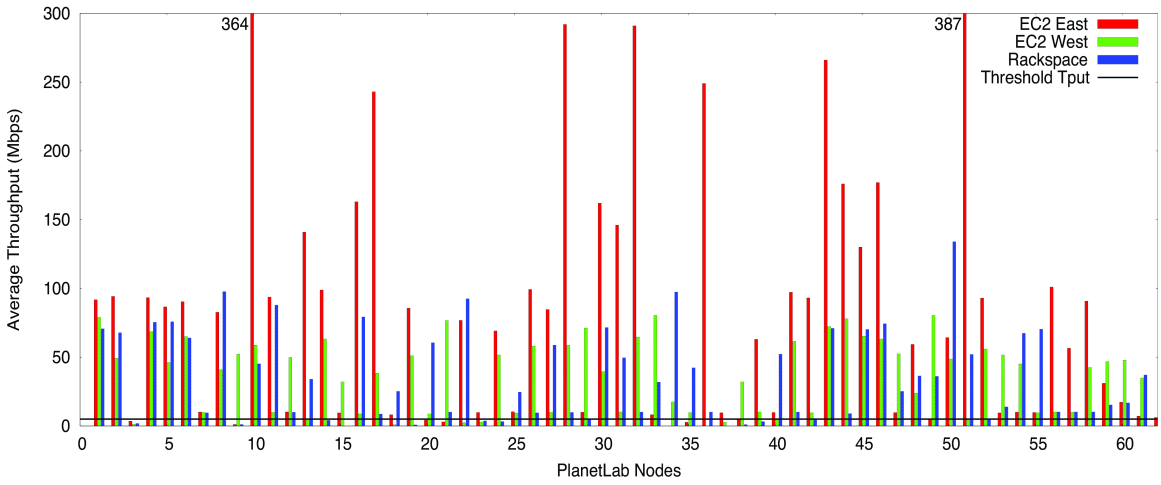
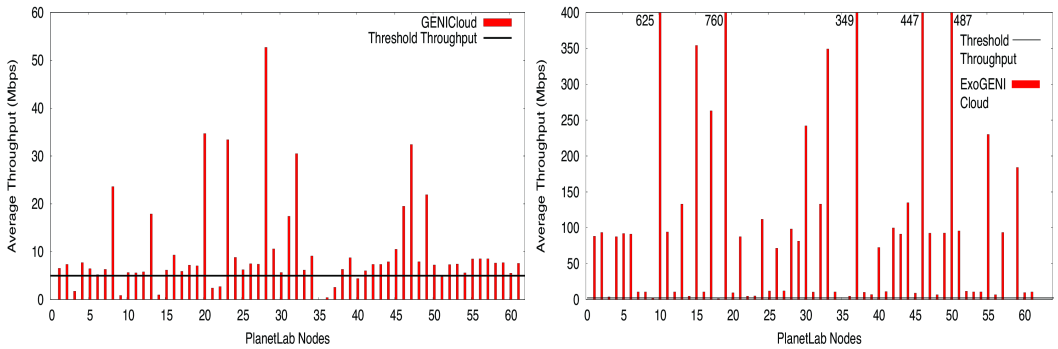


Figure 3.2. Serial Measurement between PlanetLab Nodes and Commercial Cloud Instances. Y-axis is cut off at 300 Mbps to better depict low-throughput results. Throughput values for PlanetLab nodes 10 and 50 are 364 Mbps and 387 Mbps.

a series of measurements. The weather prediction algorithm (Nowcasting) uses radar data as input. NEXRAD radars are the current source of data for weather predictions in the U.S. Since we do not have access to NEXRAD radar data feeds, to perform measurements in a large-scale setting we *replicate* a distribution system that, on the network level mimics the NEXRAD system. For the measurements we make use of PlanetLab [51], a global research network that supports large-scale, distributed experiments. Thus, the PlanetLab nodes take on the role of a radar data source for the Nowcasting application in our measurement.

To make our experiment as realistic as possible we went through the exercise of choosing PlanetLab nodes that are close in physical location to the NEXRAD radars. Unfortunately, close proximity between NEXRADs and PlanetLab nodes is not always given. This is due to the fact that locations for weather radars are chosen based on parameters like coverage and beam blocking, which often places them in remote areas. Although there are 159 NEXRAD radar sites in US, we could find only 103 PlanetLab nodes close to those locations, out of which there were around



(a) Serial Measurement between PlanetLab Nodes and GENICloud Instance

(b) Serial Measurement between PlanetLab Nodes and ExoGENI Cloud Instance.Y-axis is cut off at 400 Mbps to better depict low-throughput results.

Figure 3.3. Serial Measurement between PlanetLab Nodes and Research Cloud Instances

60 PlanetLab nodes active at any given time. Hence, in our measurements we use results from around 60 PlanetLab nodes compared to 159 NEXRAD radars. The measurement results provided in this section were performed during the last week of March 2012. According to [107], radars generate data at a constant rate of roughly 5 Mbps. For the remainder of this work, we will use 5 Mbps as the minimum required throughput between a radar node and the cloud instances to allow real-time data transmission for Nowcasting operation. This threshold can be varied based on the application's need.

In the following sections, we present the results from a series of measurements we performed in which PlanetLab nodes transmit data to a cloud instance. To investigate if the location of the EC2 instance has an impact on throughput we performed the measurement twice, once with an EC2 instance in a West Coast data center and another in the EC2 East Coast data center. Also, to investigate if the time of the day has any impact on our measurement results we perform our measurements twice for each cloud instance, once during the day and once at night time. Approximate time for the day measurement was around noon and for the night measurement was

around midnight (PST). For these measurements we used Iperf [22] to transmit data via TCP.

3.2.2.1 Serial Data Transfer

In this section, we present the results from serial transfer of data from PlanetLab nodes to cloud instances. With these measurements we intend to evaluate the average throughput of each individual path between a radar node and cloud instances in the absence of competing traffic to a specific cloud instance.

Figure 3.2 shows the results of the measurements from PlanetLab nodes to the EC2 East Coast data center, EC2 West Coast data center and Rackspace cloud instances, while Figure 3.3 shows the results of the serial measurement from PlanetLab nodes to GENICloud and ExoGENI cloud instances during the day. During this measurement we transmit data from each individual PlanetLab node to cloud instances for 15 minutes. We would have transmitted for a longer time period but due to high data transmission rate, the overall data volume that can be transmitted by PlanetLab nodes is limited to low bandwidth burst after 10.8 GB of total data transfer. PlanetLab uses this policy to avoid any DDoS attacks. Tables 3.1 and 3.2 show a summary of the measurements performed on the commercial and research clouds. As shown in the tables, the average throughput over all transmissions is 36.248 Mbps for the case of a West Coast EC2 instance, 85.035 Mbps for an East Coast EC2 instance, 35.335 Mbps for a Rackspace instance, 9.744 Mbps for a GENICloud instance and 110.22 Mbps for an ExoGENI instance for the measurements performed during the day. The average throughput on all the cloud instances is greater than the threshold throughput of 5 Mbps required for our Nowcasting application. This implies that when only one radar node is used for our Nowcasting application, the network links between the radar node and cloud instances offer sufficient capacity to execute the application in real time.

Tables 3.1 and 3.2 show the results for the measurement performed at night and it can be seen from the average throughput for the serial measurements row that there is minimal improvement compared to the day time measurements. This improvement shows that there is slightly less network traffic at night but the improvement in average throughput is not drastic. This is good news for our Nowcasting application, since its real time constraints do not allow for a delayed execution. Otherwise, the results show that it makes sense to perform data upload for non time-critical applications to night time to benefit from the slightly increased network performance.

Though the average data throughput for the serial measurement on all the cloud instances is well above the 5 Mbps requirement, it can be seen from Figures 3.2 and 3.3 that for some of the paths from the nodes to the cloud instances the average throughput is well below the threshold. About 13% of the nodes have average throughput of less than 5 Mbps to the EC2 machines for both East and West Coast, while about 18% of the nodes have an average throughput of less than 5 Mbps to the Rackspace cloud instance and GENICloud cloud instances and 10% of the nodes to ExoGENI cloud instance have less than 5 Mbps average throughput. An overview for all measurements is given in Table 3.3.

In the specific case of the serial measurement these cases are most likely below the 5 Mbps threshold due to issues with the specific PlanetLab node or the access link of that node. To affirm our conjecture we take a look at the throughput measured (as shown in Figures 3.2 and 3.3) for node 3. One can see that the throughput is consistently low for all five measurements which leads us to conclude that this is a PlanetLab node or access link related issue. By inspecting nodes 4, 5, and 6, one can also observe nodes that consistently produce good throughput results for all five measurements.

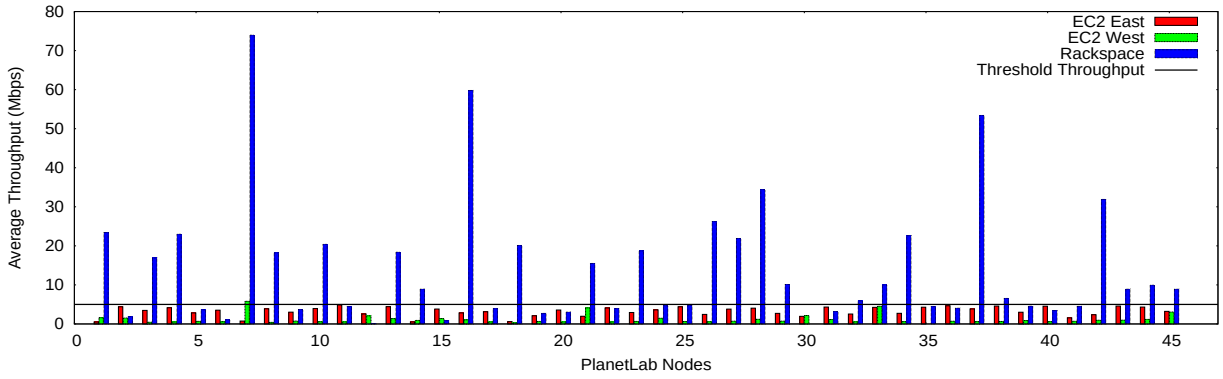


Figure 3.4. Parallel Measurement between PlanetLab Nodes and Commercial Cloud Instances

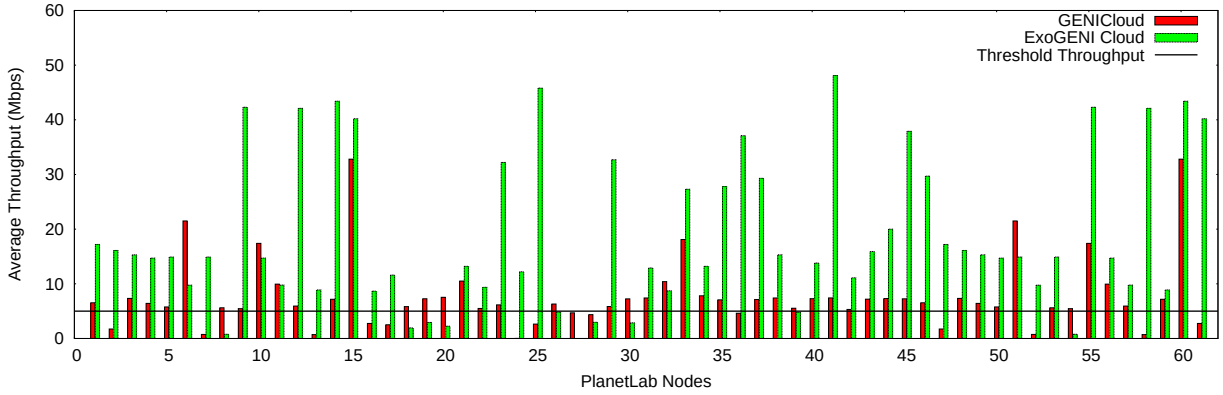


Figure 3.5. Parallel Measurement between PlanetLab Nodes and Research Cloud Instances

3.2.2.2 Parallel Data Transfer

The previous section (Section 3.2.2.1) gives an insight into the data link capacity of radar nodes to cloud instances without any significant, impeding traffic at a cloud instance. In this section, we perform a series of measurements where data were transmitted from the PlanetLab nodes mimicking the NEXRAD radars to a cloud instance in parallel. The intention is to verify if cloud services can handle the traffic generated by a large set of senders and still maintain the threshold throughput of 5 Mbps needed for Nowcasting.

Figure 3.4 shows the average throughput values of the parallel data measurements from PlanetLab nodes to the commercial cloud services and Figure 3.5 shows the average throughput values of the parallel data measurements between PlanetLab nodes and the research cloud instances for the measurement performed during the day. The average throughput over all transmissions is 3.146 Mbps, 1.249 Mbps and 14.122 Mbps for EC2 East Coast, EC2 West Coast and Rackspace cloud instances, respectively. The average throughput over all transmissions is 7.364 Mbps for the GENICloud instance and 17.2 Mbps for the ExoGENI cloud instance. Tables 3.1 and 3.2 show the average throughput results of the parallel measurements on all the cloud instances for both day and night. As it can be seen from the results of the night measurement the improvement on the average throughput is very small compared to the day measurement, which implies that there is not a significant reduction in competing traffic from other sources to the cloud data centers during the night.

The results from the parallel measurements show that the throughput of each individual PlanetLab node to an EC2 instance is lower than the threshold throughput of 5 Mbps. Thus, in this scenario real-time operation of the Nowcasting application cannot be guaranteed. GENICloud, Rackspace and ExoGENI instances perform comparatively well during the parallel measurement producing an average throughput of greater than 5 Mbps. But, we note that there is significant reduction in the average throughput for the parallel measurement from that of serial measurement for Rackspace and ExoGENI instances, which might reduce the efficiency of Nowcasting algorithm in the case of many sources transmitting in parallel. Even though the average throughput of GENICloud, Rackspace and ExoGENI cloud instances are above 5 Mbps threshold throughput, there are about 48%, 22% and 17% of the nodes with throughput of less than 5 Mbps to Rackspace, GENICloud and ExoGENI cloud instances, respectively.

The results from this measurement show that in the case of parallel transmissions and with a large number of senders the average throughput is low. Note, that due to PlanetLab specifics only approximately 60 nodes transmitted in parallel. This is significantly less than 159 radar nodes in which case the average throughput should be significantly worse. These measurement results encouraged us to investigate an alternative approach in which data from a subset of radar nodes are transmitted to a cloud instance instead of the case where *all* radars transmit their data to a single instance. The results from this approach are presented in the next section.

3.2.2.3 Distributed Data Ingest

In Section 3.2.2.2, we present the results of a parallel measurement from PlanetLab nodes to the cloud instances where all the nodes transfer data to the instance simultaneously. In reality, not all radar nodes around the country would transfer their data to one central instance simultaneously. A more likely scenario is the case where a group of radar nodes that belong to a geographic region will transmit their data to a cloud instance that is close to this subset of radar nodes[1]. Assuming we divide the radars into a subset of 10 nodes, we intend to determine the average throughput between the PlanetLab nodes and cloud instances when only a subset of radar nodes (10 nodes) transmit data in parallel. Since some of the cloud instances provide a very low average throughput of less than 5 Mbps (threshold throughput) for the parallel transmission, we intend to verify if cloud instances perform better with our distributed approach. As in Sections 3.2.2.1 and 3.2.2.2, we perform our measurements twice, once during the day and once at night and we use Iperf to transmit data to receiving instances in different clouds.

[1]Since a merging process for data from adjacent radars with overlapping scanning areas is required as a pre-processing step for Nowcasting sub-grouping of radars is a reasonable approach.

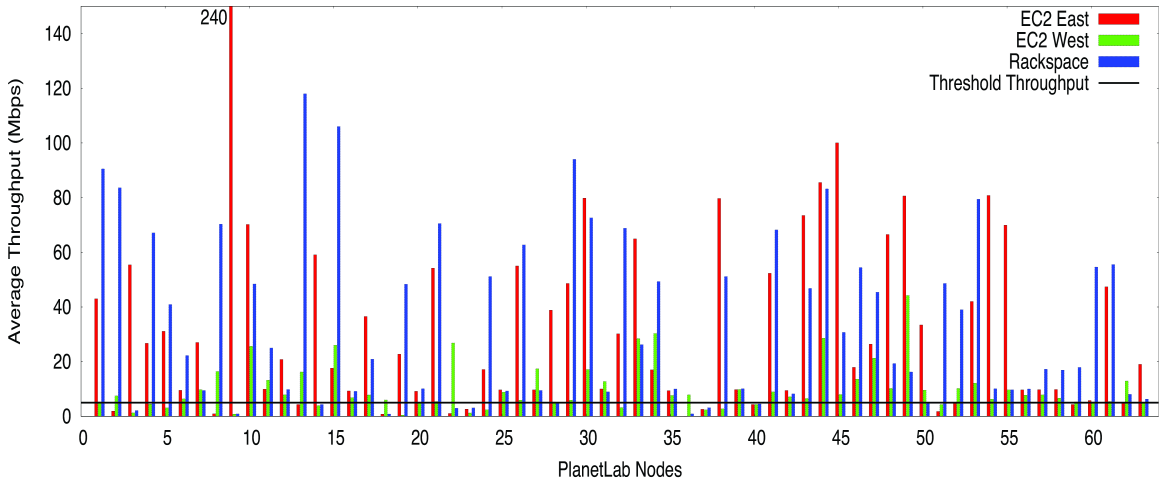
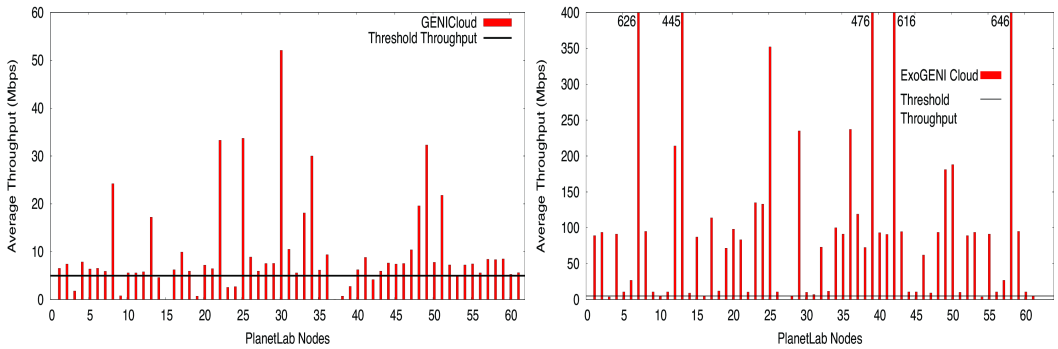


Figure 3.6. Distributed Measurement between PlanetLab Nodes and Commercial Cloud Instances. Y-axis is cut off at 150 Mbps to better depict low-throughput results.

Figure 3.6 shows the average throughput values of the distributed data measurements from PlanetLab nodes to the commercial cloud instances, while Figure 3.7 shows the average throughput values of the distributed data measurements between PlanetLab nodes and the research cloud instances. The average throughput of the measurement during the day is 32.463 Mbps, 9.995 Mbps and 34.159 Mbps for EC2 East Coast, EC2 West Coast and Rackspace instances, respectively, while the average throughput for the distributed measurement on GENICloud and ExoGENI instances is 9.434 Mbps and 112.55 Mbps, respectively. As shown in Figures 3.6 and 3.7, when only 10 nodes are transmitting in parallel to a cloud instance the average throughput is above the threshold throughput of 5 Mbps. The results from our distributed measurements show that the average throughput of the GENICloud instance does not change much from the parallel measurement explained in the previous section while there is a huge difference in the average throughput of the other cloud instances compared to the parallel measurement throughput (Section 3.2.2.2). From the results obtained during this measurement we can infer that when only a subset of radar nodes transmit data in parallel to cloud instances, the average throughput is greater than



(a) Distributed Measurement between PlanetLab Nodes and GENICloud Instance

(b) Distributed Measurement between PlanetLab Nodes and ExoGENI Cloud Instance. Y-axis is cut off at 400 Mbps to better depict low-throughput results.

Figure 3.7. Distributed Measurement between PlanetLab Nodes and Research Cloud Instances

Table 3.1. Summary of average throughput of all measurements in commercial cloud services

Measurement type	Average (Mbps)						Maximum (Mbps)					
	EC2 East		EC2 West		Rackspace		EC2 East		EC2 West		Rackspace	
	Day	Night	Day	Night	Day	Night	Day	Night	Day	Night	Day	Night
Serial	85.035	86.80	36.248	37.058	35.335	50.657	387	360	80.4	84.8	134	145
Parallel	3.146	4.278	1.249	1.064	14.122	12.458	4.87	10.8	10.4	11.2	74	43.9
Distributed	32.463	35.26	9.995	9.98	34.159	32.812	240	245	44.2	64.2	118	105

the threshold throughput and sufficient to execute the Nowcasting application in real time. Tables 3.1 and 3.2 provide the results of the average throughput obtained from the distributed measurement performed at night. As in Sections 3.2.2.1 and 3.2.2.2, there is not a huge difference in average throughput from the measurement performed during the day.

As in Section 3.2.2.1, though the average throughput of the distributed measurement from the PlanetLab nodes to the cloud instances is above the threshold throughput of 5 Mbps, there are about 22% of the nodes have a throughput of less than 5 Mbps to EC2 East and West Coast instances, 15%, 17% and 6% of the nodes with less than 5 Mbps throughput to Rackspace, GENICloud and ExoGENI cloud instances, respectively.

Table 3.2. Summary of average throughput of all measurements in research cloud testbeds

Measurement type	Average (Mbps)				Maximum (Mbps)			
	GENICloud		ExoGENI		GENICloud		ExoGENI	
	Day	Night	Day	Night	Day	Night	Day	Night
Serial	9.744	9.922	110.22	115.40	52.7	53	760	764
Parallel	7.364	8.462	17.2	41.19	32.8	51.1	48.1	417
Distributed	9.434	9.889	112.55	98.527	52.1	52	626	631

3.2.2.4 Dedicated Network Resources

To investigate how dedicated network resources have an impact on the data throughput for the Nowcasting application we performed a measurement that includes a layer 2 connection between a mimicked radar node and a cloud instance. In this measurement we transmitted data in parallel from PlanetLab nodes to an ExoGENI cloud instance over regular IP while, at the same instant, a node from a different ExoGENI cluster also transmits data over a dedicated layer 2 link to the receiving instance. The average throughput of the parallel measurement from PlanetLab nodes to the ExoGENI cloud instance is 34.68 Mbps where as the layer 2 throughput between the ExoGENI cloud instances is 572 Mbps. This result shows that dedicated network resources can provide guaranteed throughput to cloud instances for applications that require a minimum throughput.

3.2.3 Comparison of Cloud Services

In Section 3.2.2, we present the network performance measurement results of cloud instances for our real-time short term weather prediction application. In this section, we compare the network performance of research cloud testbeds for our application with that of commercial cloud services (Amazon's Elastic Compute Cloud and Rackspace).

Table 3.1 and Table 3.2 give an overview of the average throughput measured between PlanetLab nodes and the cloud instances. Table 3.3 shows the percentage

of PlanetLab nodes that have average throughput below the 5 Mbps threshold for all measurements we performed. The results of serial the measurements row in Tables 3.1 and 3.2 show that both, the research cloud nodes and the commercial cloud service nodes perform well without competing traffic with an average throughput above the required threshold of 5 Mbps. ExoGENI instances perform best without competing traffic yielding an average throughput of 110.22 Mbps followed by EC2 East Coast data center instance with 85.035 Mbps, EC2 West Coast data center instance with 36.248 Mbps, Rackspace instance with 35.335 Mbps and then GENICloud instance with an average throughput of 9.719 Mbps. We also note that there is not much improvement in average throughput results for the measurement performed at night.

The parallel measurement row shown in Tables 3.1 and 3.2 provides results that are very different to the serial measurements presented above. ExoGENI, Rackspace and GENICloud instances yield an average throughput of 17.2 Mbps, 14.122 Mbps and 7.681 Mbps, which is greater than the threshold throughput of 5 Mbps required for Nowcasting. EC2 cloud instances yield an average throughput of 3.146 Mbps and 1.249 Mbps for East Coast and West Coast data center respectively, which is well below the threshold throughput of 5 Mbps required for our real-time application of Nowcasting.

The distributed measurement row in Tables 3.1 and 3.2 shows that each of the cloud instances considered perform better when only a subset of nodes are transmitting data in parallel. As in the serial measurement scenario the average throughput results from all cloud instances are greater than the threshold throughput of 5 Mbps. The ExoGENI cloud instance performs better than the other three cloud instances with an average throughput of 112.55 Mbps while the Rackspace cloud instance provides an average throughput of 34.159 Mbps. GENICloud instance provides an average throughput of 9.434 Mbps and EC2 cloud service provides an average throughput of 32.463 Mbps and 9.995 Mbps in the East and West Coast data centers, respec-

Table 3.3. Percentage of PlanetLab nodes with throughput below 5Mbps threshold.

Measurement type	EC2 East		EC2 West		Rackspace		GENICloud		ExoGENI	
	Day	Night	Day	Night	Day	Night	Day	Night	Day	Night
Serial	12.9%	14.5%	12.5%	18.7%	18.7%	15.6%	15.6%	14.6%	9.8%	8.2%
Parallel	100%	68.8%	100%	97.3%	48.8%	17.7%	22.2%	15.5%	17.7%	24.4%
Distributed	12.6%	16.6%	21.8%	35.9%	15.6%	15.6%	17.1%	15.6%	5.7%	13.4%

Table 3.4. Computation time analysis of Cloud Services

Instances	Memory	ECUs	Disk	Cost	Exec. Time
EC2	7.5 GB	4	850 GB	$.34/hr	74.341s
Rackspace	8 GB	4	320 GB	$.48/hr	96.53s
GENICloud	8 GB	4	20 GB	-	67.45s
ExoGENI	8 GB	4	20 GB	-	68.84s

tively. The measurement results show that GENICloud instance is the most consistent of cloud instances providing almost identical average throughput irrespective of the number of nodes used in data transmission.

From the measurement results presented in this section, we can conclude that the networking capabilities of the cloud instances are sufficient for the real-time operation of our CloudCast application. We can also infer that, the network performance of research cloud testbeds are in par with that of the commercial cloud services and can be used as a test instance to execute our Nowcasting application without incurring any additional cost.

We would also like to mention that the measurement results presented in this section can also be used to verify which cloud services offer sufficient network capacity for other applications that require a certain throughput. One such example is a camera sensor network for security or monitoring for which data are transmitted from a set of distributed cameras to a central processing node. Assuming the processing would be performed on a cloud instance and the minimum throughput requirement for a single camera stream is known one can simply use the results presented in this section to determine which cloud instances can support such an application.

Table 3.5. Nowcast algorithm execution time.

Instance Type	Memory (GB)	Disk (GB)	Cost/hr ($)	Total Cost ($)	Exec. Time (Sec)
EC2 Large	7.5	850	0.34	1.13	74.34
EC2 X-Large	15.0	1690	0.68	2.17	73.77
EC2 High Memory X-Large	17.1	420	0.50	1.63	55.90
EC2 High Memory Double X-Large	34.2	850	1.00	3.54	55.40
Rackspace Large	8.0	320	0.48	1.63	96.53
Rackspace X-Large	15.5	620	0.96	3.22	96.80
Rackspace Double X-Large	30.0	1200	1.8	5.39	96.38
GENICloud	8.0	20	-	-	67.45
ExoGENI	8.0	20	-	-	56.83

3.3 Compute Capabilities of Cloud Services

In the previous section, we investigated the network feasibility of commercial and research cloud services for our CloudCast application. In this section, we look into the computation feasibility of cloud services for our CloudCast application. We provide an overview of the weather data used in our analysis. We analyze the execution time and cost taken to compute 15-minute Nowcast from the time data is received by the cloud nodes. We measure the accuracy of our prediction algorithm for lossy and aggregated data to avoid missing data during low bandwidth links. Finally, we analyze our CloudCast application live using our own radar on campus and measure the total time taken to generate 15-minute Nowcast and send the results back to the client.

3.3.1 Weather Data

Since the performance of the Nowcasting algorithm may be affected by the type of weather, we have selected two very different cases to evaluate the performance of a Nowcast algorithm that runs in the cloud. The first case from May 10th, 2010 depicts a supercell thunderstorm forming in the CASA radar domain, strengthening as it moves ENE. This storm went on to produce an EF4 tornado in the city of Norman, OK shortly after it exited the field of view of the radars. Supercells tend to be discrete; They rob moisture from their surroundings and precipitation is contained in a relatively small area. The intense forces of a supercell may also result in non-linear movement over time, deflecting right or left as the storm reaches the upper

levels of the atmosphere and is affected by the coriolis force. Additionally, supercells generally have slower movement than other types of storms. From the perspective of Nowcasting, the slower movement may lead to reduced spatial error, however it is partially balanced by the difficulties predicting non-linear trajectories.

The second case from May 19th 2010 was a squall line moving west to east across the domain. Squalls are fast moving, long lived, elongated areas of precipitation. They may stretch hundreds of miles and reach forward speeds of 100 MPH in rare cases. Squalls do not generally produce large tornadoes as do supercells, but are known to cause weak spin-ups and straight line wind damage. In this specific case, winds were measured at approximately 60 MPH, which qualifies as low-end severe. As squalls are associated with larger air masses, the movement tends to be linear and is thus easier to model, however the fast advection speeds can lead to enhanced spatial errors if network or computation latency is introduced.

We used these *canned* weather data sets for our analysis described in the following sections. The use of identical data sets is necessary to allow for a valid comparison of the performance of different cloud service, since individual weather events can be very different and thus the computational requirements of Nowcasting. In Section 3.3.6, we will present results from an experiment where *live* data is used.

3.3.2 Cost of Operation

To better understand the operating cost of weather forecast models and the advantages of moving the forecasting to the cloud, we provide a brief analysis of the cost for forecasting below. In the Spring of 2011 the CASA Engineering Research Center operated a four radar network in southwest Oklahoma. Beginning April 2nd and ending June 15th, an intensive operation period (IOP) was defined for a total of 75 days (1800 hours), representing the climatological peak season for thunderstorms. During this time the network, covering 10,300 square kilometers, had ongoing con-

vective precipitation for approximately 90 hours, or 5% of the IOP. Several of the derived products generated by CASA, including multi-Doppler winds and 15 minute reflectivity nowcasting, are only useful during these events since X-band Radars are not able to determine winds in clear air and Nowcasting algorithms do not predict convective initiation. Our approach was to dedicate individual computers to each product despite the 95% idle rate and frequent over-provisioning during smaller scale and weaker events. The machines necessary to process the data in a timely manner were purchased in 2011 and cost over $4000 dollars each, not including IT overhead expenses associated with their management.

As a result of this experience, we looked into the Infrastructure-as-a-Service (IaaS) cloud model, a more efficient compute cloud based architecture designed to procure computing resources on demand in an automated fashion. A lightweight command and control server differentiates between clear-air, stratiform rain, and convective regimes and issues Java-based in-line spot requests to Amazon EC2. Disk images pre-configured with the various processing algorithms are uploaded in advance, triggered and released as weather enters and exits the radar domain. The routines responsible for triggering more resource intensive algorithms are integrated on-board the radar and require no additional maintenance or siting overhead. These include reflectivity thresholding (RT) and storm cell identification and tracking (SCIT) with local radar data, as well as external monitoring routines such as XML based RSS feeds for WFO issued watches and warnings.

Based on current spot prices for similar machines as those used in the 2011 IOP (45 cents/ hour), 90 hours of active use would cost $40 per product, plus $2 per user to stream the resultant data out of the cloud. This represents significant cost savings over the dedicated compute model assuming a 5-year lifecycle. In addition to computing, long-term data storage of the radar data is another substantial cost. The 90 hours of moment data containing storms from 4 radars combined with the

derived merged products amounts to roughly 700GB for the IOP. Current rates of 10 cents per GB per month yield ongoing $70/month costs to keep this data online. Disk arrays are expensive to purchase and maintain and the cloud storage model appears to be cheaper, though the advantages are less than for the computing.

3.3.3 Cost and Computation Time

In this section, we discuss the measurement procedure to calculate the cost and computation time of the Nowcasting operation for one hour of weather data described in Section 3.3.1 with various instance types offered by Amazon EC2 and Rackspace cloud services. We also calculate the computation time of the Nowcasting operation for the same weather data with the instances offered by GENICloud and ExoGENI research cloud services. As mentioned in Section 3.1.2, we use DiCloud as an interface to start the instances, attach storage, and calculate cost of operation for Nowcasting in EC2 cloud service. For each of the EC2 instance types in Table 3.5, DiCloud is used to bring up the instances with the Nowcasting image, attach EBS storage and start the ingest of weather data from the radars. Once the cloud-based Nowcast instance receives the first set of radar scans, it starts to generate 1-to-15 minute Nowcasts which are stored in EBS.

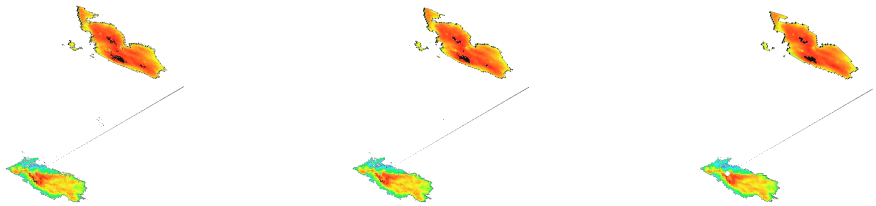
We have carried out this operation for one hour of weather data and with DiCloud's cost tracking mechanism (see Section 3.1.2) we determine the cost for running this 1-hour Nowcast operation. In addition, the execution time for 15 minute Nowcasting of each weather data from the radar in the EC2 instances is measured and the mean is calculated over the whole 1-hour interval. We carry out the same operation on other EC2 instance types and show the results in Table 3.5. As it can be seen from Table 3.5, the computation time for Nowcasting on EC2 cloud service decreases as the instance types selected becomes larger and more expensive. The last column in Table 3.5 shows that, the execution time of the 15-minute Nowcast

algorithm decreases by $\sim 30\%$ by executing it on a high-performance instance while the cost almost triples. Thus, choosing an appropriate EC2 instance for executing a Nowcast is a trade-off between faster execution time and cost.

Similar to EC2 cloud service, Rackspace commercial cloud service also offers different types of instances as shown in Table 3.5. We carried out the Nowcasting operation on each of the Rackspace instance types mentioned in Table 3.5 for one hour of weather data and calculated the computation time taken by the Rackspace instances to generate Nowcasts for the weather data. We also noted the cost for Nowcasting operation on Rackspace cloud instance using the web interface offered by Rackspace which allows the users to keep track of their instances.

As it can be observed from Table 3.5, the computation time taken by Rackspace cloud instances is about 96.50 secs on average to generate 15-minute Nowcasts. Comparing the computation time taken by the two commercial cloud instances (EC2 and Rackspace) to generate 15-minute Nowcasts, it can be seen that the computation time taken by EC2 cloud instances are lesser than those taken by Rackspace cloud instances. Another interesting result shown in Table 3.5 is that with high performance instance types in EC2, the computation time decreases, while the computation time remains about the same on Rackspace cloud service. This observation shows that it might not be beneficial to pay more for high performance instances on Rackspace for computation intensive applications.

We also performed the computation time analysis on research cloud service (GENICloud and ExoGENI) instances for the same weather data used to analyze the computation time of commercial cloud services. The result from our analysis is presented in Table 3.5. Both the research cloud services offer only one type of instance which is sufficient for the standard operation of Nowcasting application. From the computation time results presented in Table 3.5 for GENICloud and ExoGENI instances, it can be seen that both the research cloud instances take less time (67.45 secs and



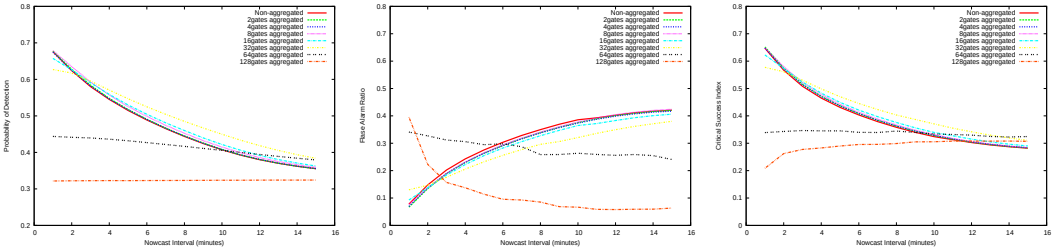
(a) Non-aggregated radar data (b) 50% aggregated radar data (c) 75% aggregated radar data

Figure 3.8. Example for simple, lossy radar data aggregation

56.83 secs respectively) to compute 15-minute Nowcasts compared to the EC2 and Rackspace cloud instances (74.34 secs and 96.53 secs respectively) of the same characteristics. It can also be noted that, ExoGENI research cloud instance is the fastest instance to compute 15-minute Nowcasts in just 56.83 secs compared to any of the other cloud instances considered.

3.3.4 Data Aggregation

The results from the measurements described in Section 3.2.2 shows that the link capacity from the radar nodes and the cloud instances is not always feasible for our Nowcasting operation with certain link throughput always lower than the threshold throughput of 5 Mbps (See Table 3.3). One way to encounter for the link capacity limitations between the radar nodes and the cloud is to reduce the amount of data through aggregation. In this section, we look at a very simple, lossy aggregation method to reduce the amount of data that has to be transmitted between the radars and an instance in the cloud. The method takes a standard radar moment data file and down-samples it to a specified amount. The down-sampling simply merges a specified number of *gates* in a *radial*. For example, a down-sampling factor of two averages two gates into one and, thus, reduces the file size and the required bandwidth for real-time transmission by a factor of two. Figure 3.8 shows an example where this aggregation



(a) Probability of detection (POD) (b) False Alarm Ratio (FAR) (c) Critical Success Index (CSI)

Figure 3.9. Nowcasting analysis with various aggregation factor for weather data from May19th 2010.

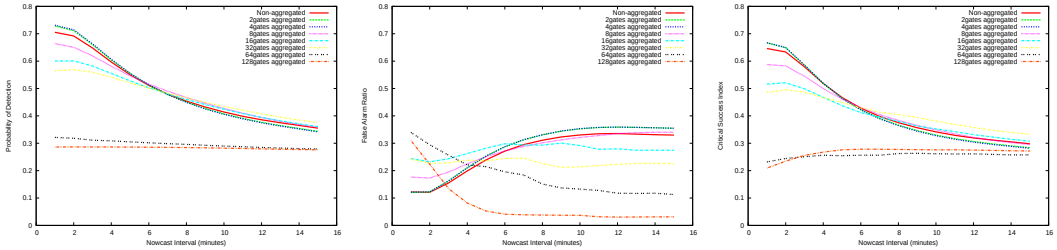
method is applied to single radar data. We observe that even with a relatively high aggregation rate (75% in Figure 3.8(c)) the visible degradation is marginal.

To analyze the impact of lossy data aggregation with an objective metric, we use FAR, POD, and CSI, described in Section 3.1.3. In the next section, we analyze the impact of lossy aggregation for our two weather scenarios.

3.3.5 Lossy Prediction

We now look into Nowcast prediction accuracy from the aggregated radar data for the two weather scenarios explained in Section 3.3.1. We compare the scores of the 15-minute Nowcast data for the aggregated data with that of the non-aggregated data for the two weather scenarios in the following.

Figures 3.9(a), 3.9(b) and 3.9(c) show the POD, FAR and CSI respectively for the 15-minute Nowcasts generated for weather data collected on May 19th 2010. Similarly, Figures 3.10(a), 3.10(b) and 3.10(c) shows the POD, FAR and CSI respectively for the 15-minute Nowcasts generated for weather data collected on May 10th 2010. The metrics are calculated for Nowcasts generated for weather data aggregated up to 128 gates. From the two weather scenarios presented in the Figures, we can infer that the reduction in prediction accuracy is minimal due to radar data aggregation up to 32 gates. From the CSI or threat scores for the two scenarios, we observe that for an aggregation factor of 32 gates, the prediction scores are similar to the non-aggregated



(a) Probability of detection (POD) (b) False Alarm Ratio (FAR) (c) Critical Success Index (CSI)

Figure 3.10. Nowcasting analysis with various aggregation factor for weather data from May10th 2010.

Table 3.6. Nowcast algorithm execution time for live measurement

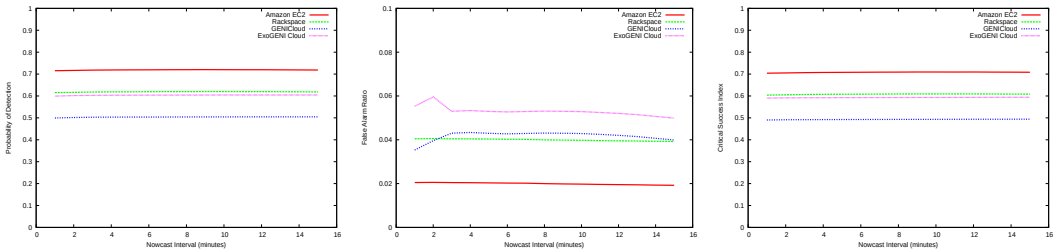
Instances	Memory (GB)	Disk (GB)	Exec. Time (s)	Total Time (s)
EC2	7.5	850	71.98	95.08
Rackspace	8	320	102.48	120.33
GENICloud	8	20	67.37	78.60
ExoGENI	8	20	56.10	72.07

radar data, while further aggregation degrades prediction accuracy. Our observation allows us to aggregate radar data up to 32 gates for those radars whose link bandwidth is below threshold, and still maintain high prediction accuracy.

3.3.6 Live Measurement

In this section, we present the results from a live, end-to-end measurement that was performed with our own radar on campus as a proof-of-concept for our CloudCast application.

We carried out a live measurement on each of the cloud instances considered to calculate the overall time taken for the Nowcasting process from the time data is generated by the radar, transmitted to the instance executing the algorithm, generating 15-minute Nowcast images and sending the predicted images to a central web server to be used by clients. The overall duration of the sum of the individual steps mentioned above determines how much time a user has between when a severe weather situation is indicated by the Nowcast and when it actually occurs. Obviously, it is the goal to maximize that time interval. For the live measurement analysis we used the data from our own radar on campus which is a prototype CASA radar [81]. Table 3.6 shows



(a) Probability of detection (POD) (b) False Alarm Ratio (FAR) (c) Critical Success Index (CSI)

Figure 3.11. Nowcasting analysis with live weather data on each of the cloud services.

the result from the live measurement carried out on the cloud instances. The average overall time taken for the whole Nowcasting process was about 95.08 seconds for the EC2 cloud instance, out of which 71.98 seconds were consumed by the generation of 15-minute Nowcasts by the algorithm running on the cloud instance. This means, it takes about 23.10 secs for the data to be sent from the radar to the receiving instance, create the predicted images and transfer the images back to the central server to be accessible by clients. Similarly, the total time taken for the whole Nowcasting process on Rackspace, GENICloud and ExoGENI cloud instances is 120.33, 78.60, 72.07 seconds, respectively. The overall time taken for the whole Nowcasting process on ExoGENI cloud instance is much lower than the time taken on the other cloud instances considered which is also the case for the time taken only for the generation of 15-minute Nowcasts as explained in the Section 3.3.3.

Figure 3.11 shows the POD, FAR and CSI metrics (defined in Section 3.1.3) calculated for the live weather data analysis carried out on each of the cloud services. The difference in the values for each cloud instance is due to the different weather that occurred during each live measurement. Since we performed the measurement for each cloud instance at different points in time atmospheric condition can be quite different. In summary, we demonstrated with this live measurement the potential and feasibility of performing short-term weather forecasts for mobile devices in the cloud. From the timing analysis we found that, for 15-minute Nowcasting it takes

only approximately 2 minutes to generate the Nowcast images and disseminate it to the client, which leaves the clients with 13 minutes to take any necessary action based on the 15-minute prediction.

3.4 Summary and Conclusion

In this chapter, we presented CloudCast, an application for personalized short-term weather forecasting. CloudCast uses instances from commercial and research cloud services to execute a short-term weather forecasting application which is based on a Nowcasting algorithm. We demonstrate the network feasibility of using cloud services for our CloudCast application by performing a series of measurements. Our results show that, serial transmission of data from radars to cloud instances is suitable for our application, while parallel transmission from a large set of radar nodes is a bottleneck for real-time operations of CloudCast. We also infer from our results that, using a small set of radar nodes for parallel transmission to a single cloud instance is suitable for our application in the cloud.

We also compare the compute feasibility of Nowcasting in the cloud with real weather data on various instance types offered by cloud services. We calculate the computation time and the cost to generate 15-minute Nowcasts in the cloud. Our results show that computation time for generating 15-minute Nowcasts reduces by $\sim 30\%$ if executed on a high-performance instance, but with an almost 300% higher cost than a low-performance instance in EC2 cloud service. It can be observed from the results that ExoGENI cloud service provides lower computation time to generate 15-minute Nowcasts compared to other cloud services considered. We also performed a live experiment with our CloudCast architecture based on data from a weather radar that is located on our campus. The results from our live measurement show a very high prediction accuracy whereas the delay between data generation at the radar

to the delivery of 15-minute Nowcast image to a mobile client is less than 2 minutes on average.

In summary, we have shown that commercial and research cloud services are feasible for the execution of our real-time CloudCast application. With this approach, accurate short-term weather forecasts can be provided to mobile users. We believe that CloudCast has the potential to support emergency managers and the general public in severe weather events by promptly providing them with potentially life-saving information.

CHAPTER 4

IMPROVING CACHE EFFICIENCY OF YOUTUBE

The previous chapter talked about the feasibility of using clouds for real time scientific application of short-term weather forecasting. In the following chapters, we will look into the feasibility of using cloud services to improve the cache efficiency of VoD services, YouTube in particular and managing the long tail popularity distribution effect of YouTube.

To improve the efficiency of YouTube caches and manage the effect of long tail popularity distribution for YouTube, we will look into the feasibility of using YouTube recommendation system. We present a YouTube recommendation list reordering approach which modifies the related list offered to the client by YouTube based on the contents already present in the cache. To investigate if this approach is feasible, we perform a related list chain and loop analysis and related list differences analysis based on regions and users. Based on passive and real user trace analysis, we prove that modifying the related list based on the contents already in the cache improves the efficiency of YouTube caches. Finally, we present a technique of using cloud services to store YouTube videos based on their popularity and use the related list reordering approach to increase the hits on the popular servers and reduce the effect of long tail distribution of YouTube videos. This in turn reduces the resources required to host the long tail videos which are requested rarely, thus reducing the energy required to serve videos from the long tail popularity distribution of YouTube.

In this chapter, we present the impact of the related list offered by YouTube, the passive network traces we use for our analysis of reordering related list. We also

present the related list chain and loop count analysis to show that modification of related list does not interfere with YouTube's ordering of related list.

4.1 Impact of Recommended List

YouTube's related video list, shown in Figure 1.1, is a way to influence the decision of viewers on which video to watch next. When a viewer selects a video, a list of recommended videos are offered to the viewer on the same web page (in YouTube jargon defined as *"watch page"*), and the viewer may choose a video from that list.

It has been shown in related work that the related video list actually influences the viewer's video selection [105, 65]. Such behavior, causing the selection of a video through the related video list, is also defined as click-through rate. It is specified as the probability that a user who views item i will click on item j from its related video list. It is our goal to investigate if the related video list can be used to improve video distribution and delivery within YouTube. But before we try to answer this question, we investigate how related lists influence what a viewer is watching.

4.1.1 Data Set Analysis

For the analysis of our proposed cache reordering approach we analyze two network traces obtained from monitoring YouTube traffic entering and leaving a campus network. The monitoring device is a PC with a Data Acquisition and Generation (DAG) card [14] which can capture Ethernet frames. The device is located at a campus network gateway, which allows it to see all traffic to and from the campus network. It was configured to capture a fixed length header of all HTTP packets going to and coming from the YouTube domain. The monitoring period for these traces was 72 hours each, and the general statistics are shown in Table 4.1.

In trace T1, 47,986 video requests had the **related_video** tag out of total 105,339 requests ($\sim$ 45%), which indicates that this video was selected from a viewer by

Trace file	T1	T2
Duration	3 days	3 days
Start date	Feb 6th 2012	Jan 8th 2010
# Requests	105339	7562
# Videos with "related_video" tag	47986	2495

Table 4.1. Trace characteristics

choosing it from the related video list. In the case of trace T2, the numbers are significantly lower since the trace was taken during winter break when the student population on campus is much lower, i.e., $\sim$ 33% of the 7562 requested videos are selected from the related video list.

To investigate whether users choose videos from the top of the related video list with higher probability, we further analyze data from these traces to determine the position of the related video selected by the users. We determine the position of a video that was selected by a viewer from the related video list as follows. We take the related video list for video A requested by a viewer and check if video B, requested by the same user, is in video A's related video list. If so, we determine the position of video B in video A's related video list. Figure 4.1 show the results of this analysis for our trace (Trace T1), as well as the trace data used by Khemmarat et al. in [65] (Trace T2). Both traces show that about 50% of the viewers select an entry from the top five related videos. For the top ten videos, this number increases to over 70%. This observation confirms our assumption about YouTube user behavior, that a user (potentially out of laziness) usually selects videos ranked higher on the related video list independent of the video content.

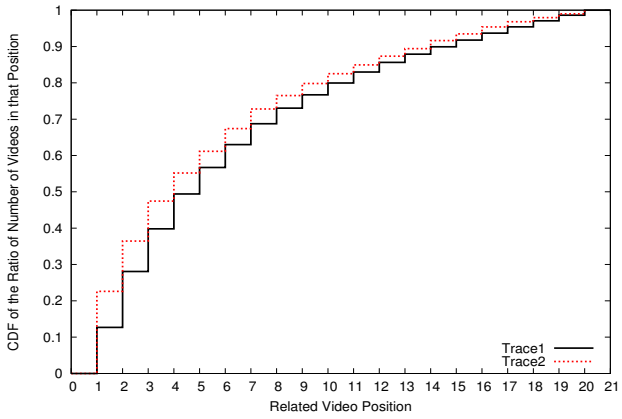


Figure 4.1. Position of video selected by users from related video list for traces T1 and T2.

4.2 Chain and Loop Count Analysis

To better understand YouTube's recommendation system and the impact of re-ordering the related list on YouTube, we first investigated how repeated selections of videos from the related lists behave. This has let us to define the concept of *chains* and *loops*.

Our definition of the *Chain Count* and *Loop Count* is as follows; *Chain Count* is the number of consecutive videos requested by the user from the recommended list until she requests a new video by other means (e.g., a search). *Loop Count* is defined as the number of consecutive videos a viewer requests from the recommended list before she requests the same video as the initial one and thereby forms a loop. Figure 4.2 provides an example of the definition of both Chain Count and Loop Count. In the example shown in Figure 4.2, both Chain Count and Loop Count is two.

The goal of this section is to gain better insight into the characteristics of related lists and how they influence the performance of caching.

4.2.1 Observed Chain Counts

In this section, we provide the results of the chain count analysis performed for the campus network traces described in Table 4.1.

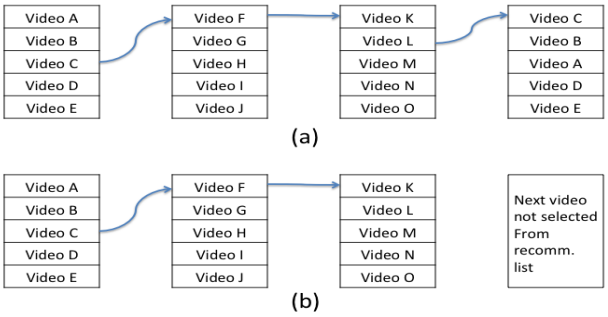


Figure 4.2. Chain and Loop count example. (a) Example for a loop of length 2, (b) Example of a chain with length 2.

For the chain count analysis, we go through the requests from each user in our trace to determine the number of consecutive requests made by the user from the related list of the previous video requested. The results from our chain count analysis on the two traces are summarized in Table 4.2. For trace T1, the viewers choose videos from the related list at least once in 84.76% of the cases, and in 15.24% of the time, the chain count is larger than one, leading to an average chain count of 1.195. The maximum chain count seen in trace T1 is 8, i.e., a user selects videos consecutively 8 times from the related list before selecting a video by other means. For trace T2, the values are a bit higher. In 48.20% of the cases, the chain count is at least one, and in 51.80% of the cases, it is more than one, leading to an average chain count of 2.304. The maximum chain count seen in trace T2 is 21.

The results in this section strengthens our assumption that users select video from the related list. In order to understand if this assumption stands good on a global scale, we also perform the loop count analysis on PlanetLab which is presented in the following section.

Approach	Trace T1	Trace T2	Global
Avg. chain count	1.195	2.304	-
Max. chain count	8	21	-
Avg. loop count	-	-	1.243
Max. loop count	-	-	19

Table 4.2. Chain and Loop count analysis results

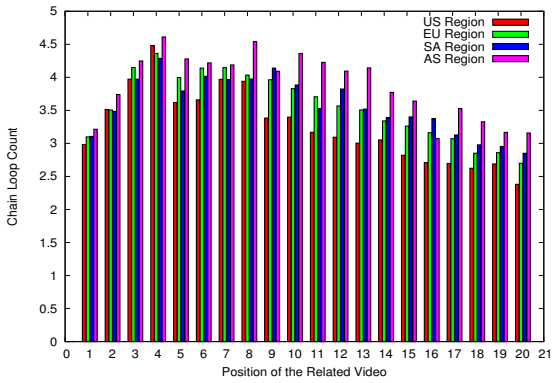


Figure 4.3. PlanetLab-based loop count measurement for fixed position selection from related list

4.2.2 Loop Length at Fixed Positions

To investigate the loop count on a global scale we performed a PlanetLab-based measurement. In a first experiment, 100 videos were accessed from each PlanetLab node. Then, we followed the chain of related movies by repeatedly selecting the same position. We did that for each of the related video list positions 1 through 20. Making these deterministic selections, we continued to follow the chain until the loop closed and we arrived at the original video again. For example, if always the top video is selected from the related list and the initial request was for video A and this video is selected again after four rounds, then this case results in a loop count of three. The results of the experiment are shown in Figure 4.3.

The experiment was performed with possible regional deviations in mind, and the results shown in Figure 4.3 are subdivided by region. As can be seen from the figure,

the general trend persists throughout the regions, and yields some unexpected, rather surprising results.

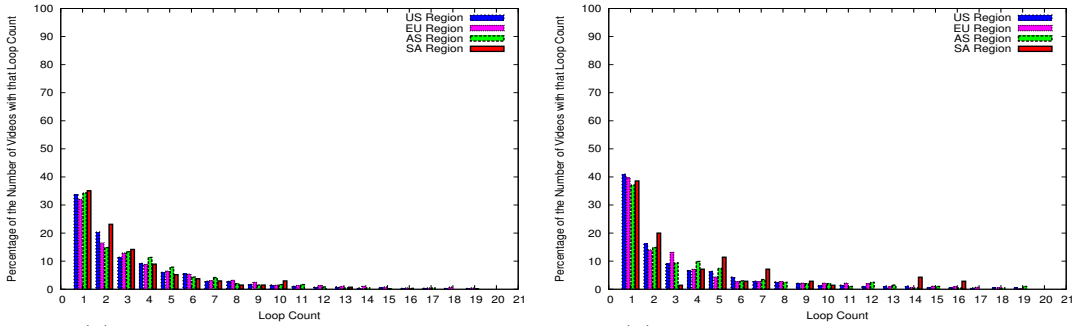
It is understood that the related list is constructed from a search for related videos, and the result is then sorted by popularity before it is presented to the user. This understanding implies that videos that appear on lower positions in the related list are less closely related to the current video than earlier ones. Considering the large library of videos held by YouTube, we would expect that the set of less closely related videos is growing with distance; consequently, following the chain in the less closely (lower) related position should find a larger number of videos, leading to a longer loop. According to Figure 4.3, that is not the case.

Since there could be many reasons for this behavior, we are trying to get a better understanding why the chain does not increase for lower related list positions. For that reason, we investigate the loop length for the case in which related list positions are randomly chosen.

4.2.3 Loop Length at Random Positions

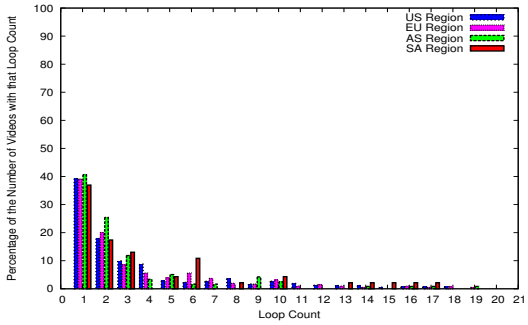
Section 4.2.2 indicates that the pool of videos from which related lists are selected is rather small for each position. To better understand the impact of always choosing from a fixed position, we perform another experiment which is based on random positions. We vary the number of maximum positions that can be selected.

For this measurement, each PlanetLab node initially requests the same video and obtains the recommended list associated with this video. From this list, a new video is randomly selected and this procedure is repeated 50 times. This experiment is repeated four times where the selection process for videos from the recommended list changes for each experiment. In the first experiment, only the top 5 videos are randomly selected, in the second experiment the top 10, and in the third and fourth

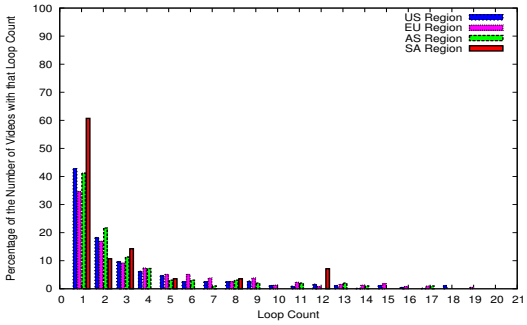


(a) Loop count for Top 5 videos

(b) Loop count for Top 10 videos



(c) Loop count for Top 15 videos



(d) Loop count for Top 20 videos

Figure 4.4. PlanetLab-based loop count measurement for random selection from related list

the top 15 and top 20 are selected, respectively. This procedure results in 50 video requests per node per experiment, and we use this result to determine the loop count.

The results from this measurement are shown in Figure 4.4. We decided to repeat the experiment with Top 5, 10, 15, and 20 video selection to investigate if different user behavior has an impact on the chain and loop length. The X-axis in Figure 4.4 shows the loop count and Y-axis shows the percentage of videos with the corresponding loop count in x-axis for Top 5 to Top 20 video selection. As can be seen in Figure 4.4, the trend for the loop lengths is relatively similar, independent of the range of top videos from which the selection is made.

It is noteworthy that the average loop length in case of random selection from a set of positions is similar to that of a fixed position, with minor changes for an increased number of maximum positions to choose from. The likely explanation for this effect is that the related list is mostly constructed from a limited and closed pool of videos.

The search features that keep videos semantically close to each other dominate over features that would promote drift towards other videos. This is probably for the benefit of the user: users selecting from the related list may not be interested in drift; the small pool prevents a "lost-in-hyperspace" effect. However, the limited pool size is also advantageous for the effectiveness of proxy caches. A request entering a pool through any video is apparently going to promote staying in the pool, increasing the chance that a chosen related video is in the cache.

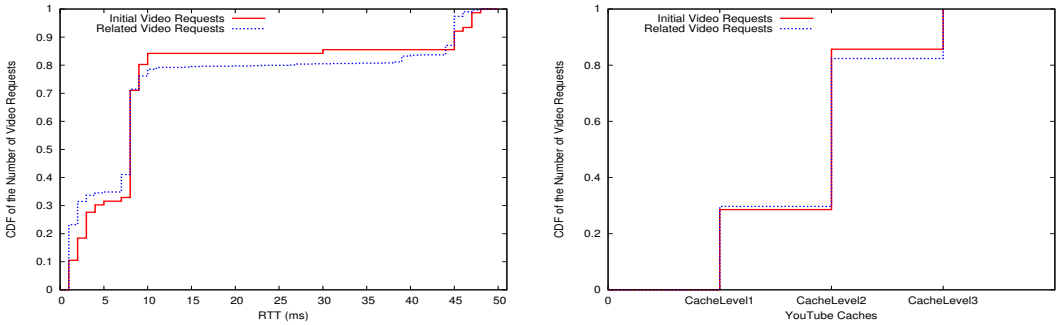
Figures 4.3 and 4.4 indicate that YouTube does not assign the video to positions for some internal purpose. In particular, it does not seem to be built to make their caching more efficient. If that was the case, then loop lengths shown in Figure 4.3 should be different from loop lengths in Figure 4.4, which is not the case. Therefore, we can conclude that we can reorder the related list without interfering with the original goal of the related list.

4.2.4 Video Origin

Since we are aware of the fact that YouTube is already employing caches in its video distribution infrastructure (see, e.g., [39] or [44] for more details), we executed the following experiment to investigate if a video and its related videos are served from a close by cache or not.

Our intention behind this idea is the conjecture that a larger average RTT for the TCP streaming session means a larger distance (in hops) between source and client. To support this conjecture, we performed the following measurement from our campus network. First, we randomly selected 100 videos from trace T1 and retrieved each of the 100 videos while capturing a packet trace by using tcpdump [34]. We then analyzed each of the 100 resulting packet traces and determined the average RTT of the TCP session that transmits the actual video data. We then repeated this

experiment for the top 20 videos from the related list of each of the initial 100 videos, resulting in 2000 data points.



(a) CDF of average RTTs for TCP streaming session

(b) CDF of video requests based on cache levels

Figure 4.5. Distribution of video delivery based on response times and cache levels.

Figure 4.5(a) shows the CDF of the average RTTs for each of the YouTube video requests as described above. As it can be seen from the figure, there are three different ranges of average RTTs (<3ms, between 3ms and 12ms and >12 ms). Mapping these RTT ranges to IP addresses gives us three different subnet ranges. Thus, we can safely say that, we see three different cache levels in our experiment and the CDF plot for the video requests from each of the cache levels (Cache Level1 is <3ms, Cache Level2 is between 3ms and 12ms and Cache Level3 is >12ms) is shown in Figure 4.5(b).

As shown in Figure 4.5(b), the majority for both groups of videos (initially selected and selected from the related list) are served from caches that are on level 2 (55% and 52%). Only $\sim$ 30% of the video from both groups are served from a level 1 cache. Further analysis revealed that all sessions with an average RTT of 3ms or less have a sender IP address from an address range that is administered by the University of Massachusetts, while address ranges for the level 2 and 3 caches are administered by Google (the owner of YouTube).

Based on the results presented above, we are now able to identify which of the videos selected from the related list are served from a close by cache (level 1) and which are served from a caches further away (levels 2 and 3). With that information

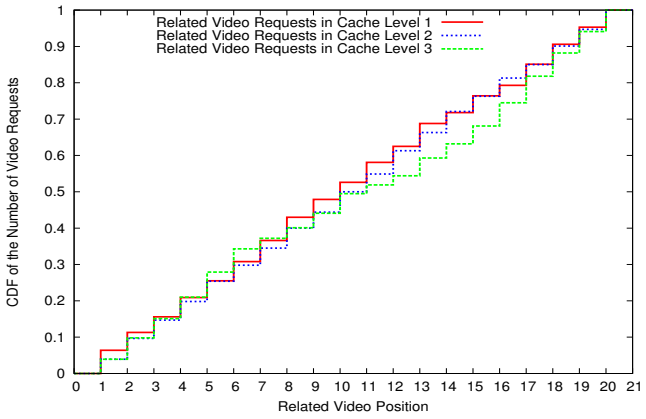


Figure 4.6. Distribution of position of video in related list compared to all requests originating from the related list.

we determined the distribution of the position of the requested video in the related list for videos served from a close by cache and for those served from more distant sources. This distribution (in form of a CDF) is shown in Figure 4.6. The nature of the CDFs shown for both cases is almost uniform. This is a strong indicator that YouTube is not giving any preferences, in terms of caching based on the position of the video in the related list. It rather confirms the results presented in [44] and [43] that claim YouTube mainly focuses on load balancing between different cache levels and not on the goal of moving content as close to the client as possible.

4.3 Summary

To summarize the chapter, we presented the network campus YouTube traces collected on University of Connecticut gateway. We looked into the YouTube viewers video selection behavior and found that about 50% of the requests made by users come from the related list offered by YouTube. Further analysis showed that, of the requests made by YouTube users from the related list 70% of the requests are from the top of the related list (Top 10). This showed that, our idea of reordering the related list offered by YouTube based on the contents in the caches has its merits.

To better understand the YouTube recommendation system and to verify that YouTube does not order its related list based on some characteristics, we performed a related list chain and loop count analysis using PlanetLab. We performed the analysis by requesting YouTube videos and their related videos based on their position. Results from our analysis showed that YouTube forms its related list based on small pool of closely related videos. This is advantageous for the effectiveness of caches as any video entering the pool continues to stay in the pool, thus increasing the chances of a chosen related video being in the cache. This behavior helps reduce the use of long tail of YouTube videos by increasing the hit rate using the small pool of videos by reordering based on contents in the cache.

We can also confirm that, YouTube does not order its related list for some internal purpose and we can reorder the related list without affecting YouTube. Also, our study on the origin of YouTube videos and related videos tells us that YouTube follows a 3-tier cache hierarchy in storing their videos and does not gives preference to store the related videos of a video in the same cache.

CHAPTER 5

RELATED LIST DIFFERENCE

Chain count and loop count analysis of requesting videos from related list showed that the related videos are formed from a small pool of videos and YouTube does not assign the videos to any particular position to make caching efficient. To make a stronger point on this statement, in this chapter, we look into the differences in related list offered to the clients by YouTube. We look into the related list differences to understand if YouTube maintains the same related list across all clients in a region or clients across regions. It also helps us understand the amount of related videos to cache or prefetch to take advantage of caching related videos as shown by Khemmarat et. al [65]. The differences in related list also maintains our stance that YouTube does not provide a position to the related list videos offered to the client, hence, reordering of related list based on the contents in the cache is not in any case an interference to YouTube's intentions of offering related list in a particular order.

In the following we present the experiment setup used in our analysis, the results obtained on the differences in the related list offered by YouTube and the impact of the related list differences on the effectiveness of caching of YouTube related list.

5.1 Experiment Methodology

In this section, we describe the experiment setup and the measures we use to analyze the related list differences for a set of videos requested from PlanetLab nodes around the globe. Such related list changes can have a significant impact on caching and prefetching performance.

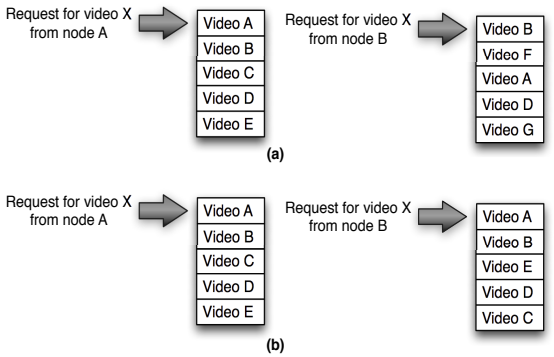


Figure 5.1. Recommended list changes between requests from different clients: (a) Content Change count, (b) Order Change count

The goal of this measurement is to evaluate if the related list is always identical for all client requests for a specific video or if it changes (either between requests from different clients or between requests for the same video from the same client at different points in time). If the list changes frequently, this can lead to a significant amount of traffic between servers and caches. Thus, it is important to know how much of the related videos change from request to request, which allows us to identify how much of the related videos to cache or prefetch to improve the efficiency of caching and prefetching related videos. We decided to use PlanetLab nodes for our experiment, since it allows us to obtain global and regional information about related list changes.

To investigate differences between the related list for requests that originate from different nodes, we performed the following experiment. We first selected a set of PlanetLab nodes from four different regions (US nodes - 197, Europe (EU) nodes - 243, Asia (AS) nodes - 62, South America (SA) nodes -17, 519 in total). Each of these 519 PlanetLab nodes requests 100 YouTube videos randomly chosen from a trace collected in the US (see Section 5.3), and for each video request we obtain the related list recommended by YouTube on its video *watch page*. To analyze the difference in related lists obtained from YouTube, we make use of the following two measures:

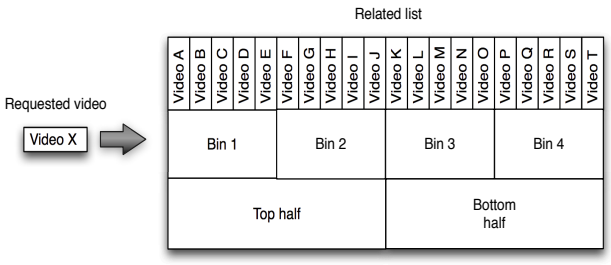


Figure 5.2. Related Video List abbrevations used in the paper.

- Content Change (CC) count: For this measure, we count the number of different videos between two related lists for a specific video irrespective of the position of the video in the list.

- Order Change (OC) count: This measure focuses on the change in the order videos appear in the related list. This measure counts the differences in the related list position wise.

For each video we perform a complete comparison between the related lists retrieved by all PlanetLab nodes. In this specific case, if n PlanetLab nodes retrieve the related list of 100 YouTube videos, then $100 * n(n-1)/2$ comparisons between related lists are performed and the average for both measures (CC and OC) is calculated. Figure 5.1 shows how we determine the differences between related lists in terms of Content Change and Order Change counts. In this case, the same video (Video X) is requested from two different PlanetLab nodes (node A and node B). Figure 5.1(a) shows an example where $CC = 2$ and $OC = 4$, while for 5.1(b) $CC = 0$ and $OC = 2$.

For our analysis we obtained each of the two measures mentioned above for bins of 5 videos from the related list, i.e., 1-5, 6-10, 11-15, and 16-20 related videos (see Figure 5.2). The goal of determining these measures for subsets of the related list is to identify if the differences are higher or lower in the top half of the related list or bottom half. Also, binning in subsets of 5 videos has an additional effect. The number of video related to the current video must be expected to be rather big in many cases,

while the list is always only a few videos long (usually 20). By performing a loop count analysis on the video requested by a user from related lists (Section 4.2), we have learned that not all potentially related videos are offered in the related lists by YouTube. By binning, we can understand whether the recommended list is derived from a single small pool of related candidates, ordered by some mechanism, or whether videos higher on the recommended list are chosen from a smaller pool of more closely related videos.

In addition to analyzing related list differences between nodes in a region, we also analyze the number of related list differences in the same node for the same video requests for 5 consecutive days. We perform this analysis to better understand two characteristics; How often does the related list change even if the videos are requested from the same node? If it differs, how much of the related video list changes for the video requested from the same node? This investigation allows us to analyze how client based proxy caching and prefetching of related videos gets affected by the related list changes.

5.2 Analysis Results

In this section, we present the results from the experiment described in Section 5.1. We present the related list differences in terms of bins of 5 videos (1-5, 6-10, 11-15, 16-20), and for each of the bins, we provide the percentage of videos with related lists difference (N) from 0 to 5.

In the Content Change count analysis, N=0 indicates that the related video list of the bins are the same in both nodes, whereas N=5 indicates that the related list does not match at all between the nodes for that bin. Similarly, in the Order Change count analysis, N=0 indicates that the related video list is the same in all positions of the related list between nodes, while N=5 indicates that the related list content differs in all the positions between the nodes for that bin.

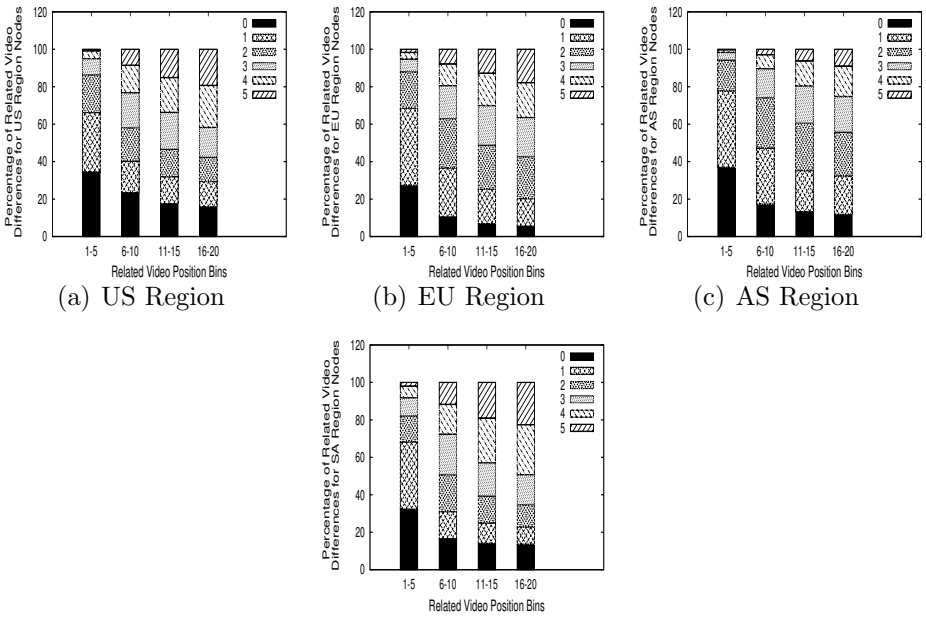
5.2.1 Regional Related List Differences

Figure 5.3 shows the results for the Content Change related list differences between nodes within a region (US, EU, AS, and SA), while Figure 5.4 shows the results for the Order Change related list differences. It can be seen from Figure 5.3 that, in each region, related list differences occur between nodes of the same region and the differences grow larger for the 11-15 and 16-20 bins. Figure 5.3 also shows that for bin 1-5, N=0 between the nodes of a region occurs for 25% to 35% of the comparisons. This percentage of similarity reduces as we move towards higher order bins. For example, in the US region (see Figure 5.3(a)), the percentage of video comparisons yielding N=5 is 0.84% for bin 1-5 whereas it is 19.34% for bin 16-20. This trend remains the same across all regions and nodes from around the globe.

The Order Change for each of the bins for each region shown in Figure 5.4 provides result that are different from the Content Change results presented in Figure 5.3. Also, the percentage of videos with N=5 in Order Change for each related video position bin is higher than the Content Change results presented in Figure 5.3. For example, the percentage of videos with N=5 in Content Change for US region (Figure 5.3(a)) for bin 1-5 is about 0.84%, whereas it is about 19.76% for the N=5 in Order Change (Figure 5.4(a)). This shows that, for the same nodes in each region, though the related video list provided might be same, the position of the videos in the list vary considerably from node to node. Another interesting observation is that the percentage of videos with related list differences are increasing as we move from lower bins to higher bins in the same fashion for Content Change (Figure 5.3) and Order Change (Figure 5.4).

5.2.2 Same Node Differences

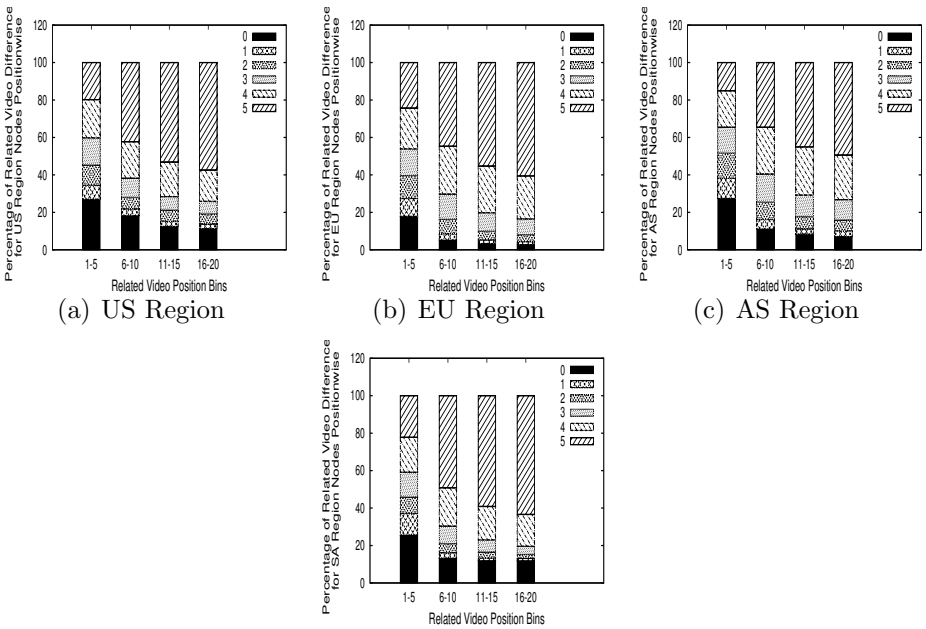
As mentioned in Section 5.1, we are also interested in the related list differences in the case of daily video requests from the same node. We analyze the related list



(a) US Region (b) EU Region (c) AS Region

(d) SA Region

Figure 5.3. Average Content Change Related Video Differences



(a) US Region (b) EU Region (c) AS Region

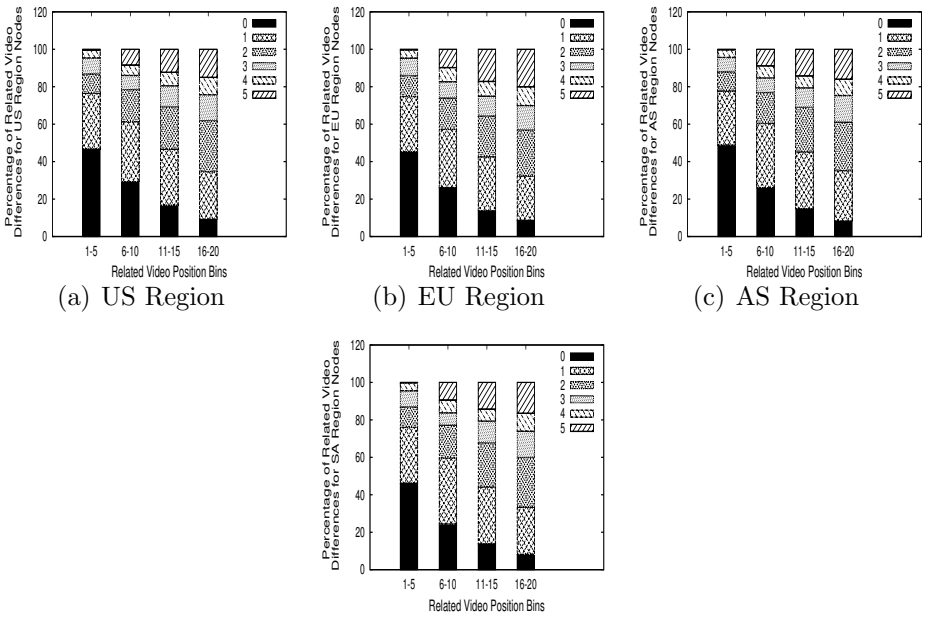
(d) SA Region

Figure 5.4. Average Order Change Related Video Differences

differences for this case to understand how much these differences would affect the efficiency of client-based caching and prefetching. Client based caching and prefetching have been shown to reduce the latency of video requests [65] and improve the viewer's experience.

Figures 5.5 and 5.6 show the related list differences for the same node on five consecutive days for Content Change and Order Change counts, respectively. The results show that the related lists change on a daily basis and the trend is similar to the one shown in Figures 5.3 and 5.4. I.e., the related video differences are low for the upper related list bins and increases for lower bins. The interesting observation from Figures 5.5 and 5.6 is that the number of related list differences remains the same for all related list bins across all regions. The number of differences of related lists for the same set of videos in the US region is the same as that in the EU, AS, and SA region.

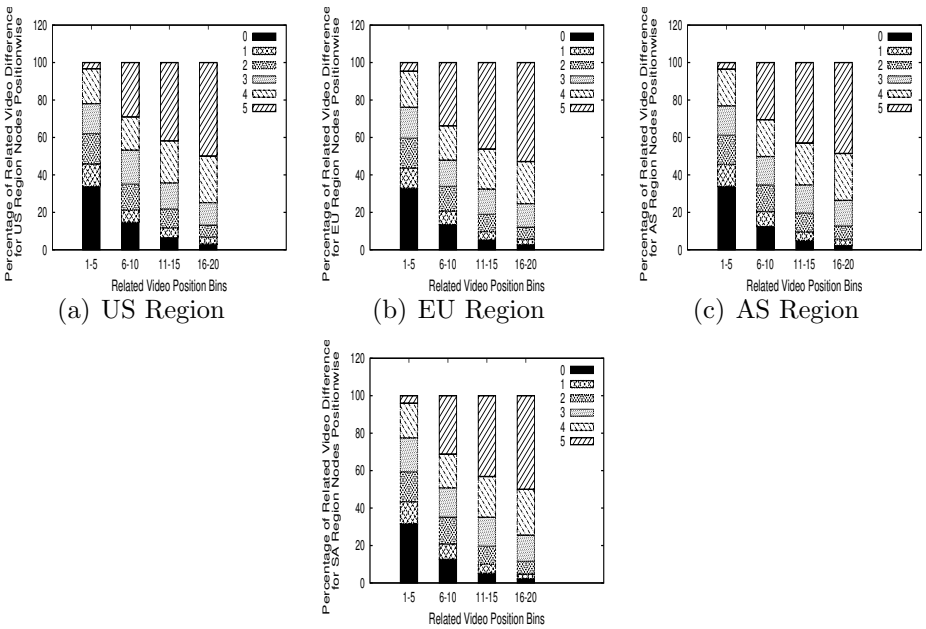
This is interesting because it shows that on one hand YouTube uses its regional caches to prepare the related list and does not generate these list centrally. But, on the other hand, the number of list changes for a video that is requested twice from the same node in a 24-hour interval is always the same. In combination with the user behavior mentioned above, this mechanism will prevent the low-popularity videos (the ones from the long tail of the popularity distribution) from being stored on regional caches. But it is counterproductive for the performance of caches since changes in the related list decrease the potential for cache hits. However, such changes do not affect our approach of reordering related lists to improve cache efficiency. As a matter of fact, it aides our approach as the related list is formed regionally and forms a small pool compared to a centrally generated related list.



(a) US Region (b) EU Region (c) AS Region

(d) SA Region

Figure 5.5. Daily Average Content Change Related Video Differences



(a) US Region (b) EU Region (c) AS Region

(d) SA Region

Figure 5.6. Daily Average Order Change Related Video Differences

5.3 Impact of Related Video Differences

In this section, we analyze the implications of the related list difference results presented in Section 5.2 on the efficiency of caching and prefetching in terms of reducing bandwidth consumption and latency. The advantages of caching and prefetching of related list was presented in detail by Khemmerat et.al [65].

The question we want to answer is, how do the related list differences between the nodes in a region and the related list differences on the same node for consecutive days affect the efficiency of caching and prefetching related videos? How much of the related list should be cached or prefetched to improve the efficiency of caching and prefetching?

5.3.1 Impact of Regional Differences

To answer these questions, we analyze the results from Figure 5.3 and Figure 5.7. In Figure 5.3, we can see that about 35% of the time there are related list differences of at least 2 for the top related list bin, which includes the most requested position of the related list as shown in Figure 5.7 (60%). Assuming an approach in which the Top 5 related videos of every video requested are cached or prefetched, $\sim 21\%$ of the time three of the five cached videos will be of no use for subsequent viewers who are served from the same cache. This would require the download of an additional 21% of related videos from a higher level cache or an origin server, increasing backbone network resource consumption. This number increases for lower related list bins. For example, in the bottom half bins there is a related list difference of at least 3 about 65% of the time. About 20% of all requests account for these bins, which leads to the fact that approximately 13% of the cached videos are either replaced or never requested. Though the percentage of additional load due to caching or prefetching is less for the bottom half of the related list, the percentage of related requests suggests that caching or prefetching the top half of the related list increases the hit rate. Hence,

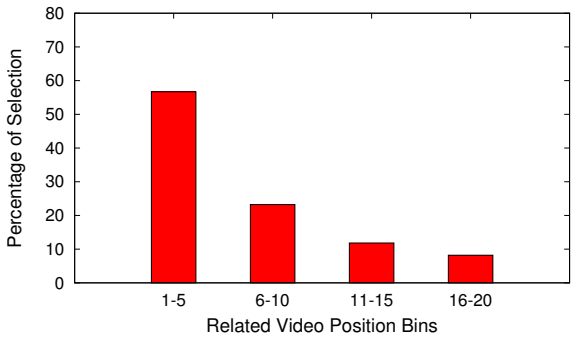


Figure 5.7. Related Video Position Analysis

less load is imposed on the backbone network compared to caching or prefetching the bottom half of the related list.

One important impact we are particularly interested is the impact of related list differences based on their order presented in Figure 5.4 on our proposed related list reordering approach. We would like to mention that the differences in the order of the related list will not have any affect on our approach as we order the related list after it is offered by YouTube based on the contents in the cache before it is sent to the clients. Hence, the order will be based on the contents of the related list already present in the cache. Therefore, any order change done by YouTube will not have any affect on our approach of reordering the related list.

5.3.2 Impact of Client Differences

So far, we have only looked into situations where the cache is located at the edge or gateway of a network, but it is sometimes advantageous to perform caching at the client to take advantage of the prefetching of related videos as shown in [65]. The results from the caching and prefetching of related videos for a cache located on the client shows a hit rate of 42% for related videos prefetched up to the top 10 and 48% hit rate when top 20 related videos are prefetched. These results show significant reduction in video latency for each request as we know that clients tend to request videos from the related videos list recommended by YouTube. But, as

shown in Figure 5.5, the related video list changes daily for the same set of video requests from the same clients. This result leads to an additional set of videos being prefetched on a daily basis, which increases the load on the network. As shown in Figure 5.5, the percentage of related list differences of at least 3 is about 20% for the top half of the related video list and it is about 40% for the bottom half of the related video list. From the results presented by [65] for the client based prefetching, the hit rate improves by $\sim 8\%$ by prefetching the top half of related video list compared to prefetching the bottom half. But, this also results in $\sim 20\%$ additional prefetching of videos when we consider the related video differences that occur daily for each client. Combining these two results, we suggest that prefetching only the top half of the related videos leads to a higher hit rate and also reduces the amount of videos that have to be additionally prefetched.

5.4 Summary

In this chapter, we presented an in-depth analysis into the differences in YouTube related list offered to the clients. Our analysis showed that there are significant changes in the related list delivered to clients, which has a diminishing effect on efficiency of caching and prefetching. The good news is that, changes are smaller for higher parts of the list (positions 1-5), which are also the most popular ones in terms of viewer selection. Our analysis also confirm the results from previous chapter that the YouTube related list is formed from a small pool of closely related videos. This small pool of videos help reduce the effect of long tail distribution of YouTube videos by offering users videos from a small pool of related list. The results from this analysis also confirms that the order in which the related list is offered does not make any difference as they change based on users and their region. Hence, the differences in related list has no effect on our reordering effect but it aides our approach as we

have a small pool of related videos to choose from the cache to reorder and keep the videos from long tail popularity distribution at bay.

CHAPTER 6

LOCAL RECOMMENDATION TO IMPROVE CACHING

In this chapter, we present the general approach of related list reordering based on the contents in the cache, a study with 40 volunteer YouTube users to validate the hypothesis of users selecting from top of the reordered related list, and the results of our approach using real user traces.

From the results obtained in Section 4.1.1, we infer that the distribution of user requests on related video position is innate in the way users interact with the GUI. It should therefore be possible to use the order of the related video list for the purpose of increasing the cache hit rate. The basic idea is to perform cache-based reordering, where the order of videos in the related video list is modified based on the cache's content. Figure 6.1 shows an example of the cache reordering approach. In this case, the related video list is sent from the server through the cache to the client. Upon receipt, the cache modifies the order of the related video list according to the videos it has currently cached. (Videos are neither removed from nor added to the list.) After modifying the list, the cache forwards the reordered list to the client.

The reordering maintains the order that the cached videos have in the original list as well as the order among the uncached videos. For example, as shown in Figure 6.1, this means, while B, C, and D are moved to the top of the related video list that is transmitted to the client, they (amongst themselves) keep the order they had in the related video list the cache originally received from the server.

Implementing the proposed cache-based reordering of the related video list is straight-forward and will add only a small amount of overhead on the cache. As shown

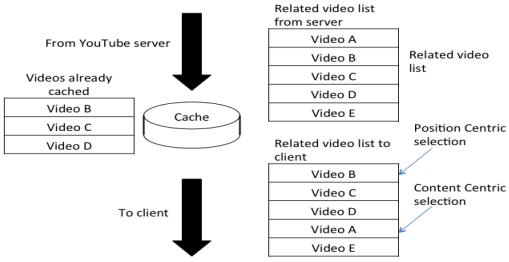


Figure 6.1. Reordering of the related video list by the cache

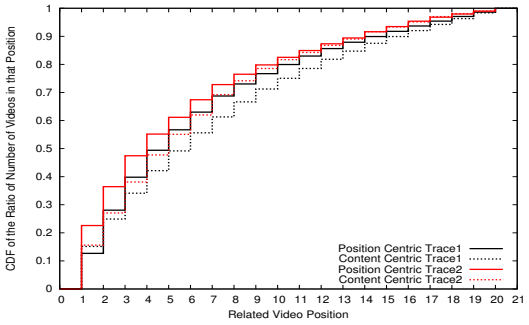


Figure 6.2. Position of video selected from related video list after reordering.

in Figure 6.1, the reordering approach requires the cache to provide the following two functionalities: *a)* maintain a list of the videos that are stored in the cache, and *b)* a process that is able to modify the related list. Functionality *a)* will be provided by any ordinary cache since it needs to determine if a client request can be served locally or has to be forwarded to a server. The additional functionality *b)* that is required for cache-based reordering is a simple manipulation of an HTML page that represents the related video list as, e.g., shown in Figure 1.1. Such simple modification of HTML code should increase the computational load on the cache insignificantly.

6.1 Related List Video Selection

In this Section, we discuss the two different choices of video selection from the reordered related list and their impact on the reordering approach. The two choices are i) Content Centric selection and ii) Position Centric selection.

Content Centric. In the Content Centric approach, the requested video in the original trace before reordering the related list is fixed, i.e., the sequence of videos that are requested is identical to the original trace. The position of the video that is requested from the modified related list, however, is different after reordering.

For the example shown in Figure 6.1, the viewer would select the video at position four of the related video list received from the cache, if the interest is absolutely in video A's content and not simply in one of the top listed videos. The content centric approach leads to a different probability distribution of the selected related video list position as shown in Figure 6.2. The dashed line in this figure shows the distribution of the positions that were chosen by a viewer in the case of content centric caching and a comparison with an unmodified related video list (see Section 4.1.1) as indicated by the solid line, shows the difference in the two probability distributions. The comparison shows that the content centric approach leads to the selection of lower ranked videos. This is caused by the fact that non-cached videos will be pushed down by the cache reordering scheme in the related video list forwarded to the client.

Position Centric. With the position centric approach, the related video list position selected by the viewer stays fixed. This clearly might result in a different content to what the viewer would have watched had the cache not modified the related video list (see Figure 6.1). In Figure 6.2, the solid line shows the distribution of position centric video selection from the reordered list for both the traces used in our analysis.

Compared to the content centric approach, the position centric approach introduces two inaccuracies that we cannot account for without real-world testing. First, with more hits, the set of cached videos will be more stable, and more videos will be cached in competition with other content. This leads to an underestimation of reordering on cache hits in our study. Second, reordering keeps popular videos at the top of the related video lists. The high probability of choosing videos from the top of

the list leads the emulated user into cycles, which most real users would avoid. This leads to an overestimation of reordering on cache hits in our study.

To overcome these short comings, we perform a user study to show that users select from the top of the related video list even after reordering and avoid entering cycles by searching different videos. We present this analysis in the next section.

6.2 User Study

Figure 4.1 showed that YouTube users select 80% of their next video from the top of the related list according to data collected from our campus trace. Our hypothesis is that users select from the top of the related list even after being provided with a modified or reordered related list, which suggests that we can modify the YouTube related list based on the contents in a cache to improve the cache efficiency of YouTube recommendation system. To verify our hypothesis, we perform a study with 40 actual YouTube users to investigate if a change of the related list is significantly changing the user behavior, in terms of most popular positions of which videos are selected from the related list. In the following, we describe the experiment we performed to gather data for this investigation and the results of the experiment.

6.2.1 Experiment Methodology

Our goal of this experiment is to collect empirical data that allows us to analyze how viewer behavior changes (in terms of selecting videos from the related list) if the original order of the related list is modified. We realize this experiment with the aid of a chrome browser plugin [79], which intercepts the YouTube webpage at the client, randomly reorders the related list shown on the watch page and loads the modified page on the browser. The video currently requested by the client is not affected by the plugin. Apart from modifying the related list, the plugin also logs the video requested by the user and the modified related list offered to the user to our server on

campus. Once the log is received we determine if the current video requested by the user belongs to the related list of the previous video requested from the same user. If so, then the position of the video requested on the modified related list is logged for our analysis. Since our goal is to verify if the user behavior of requesting videos from the top of the related list holds even after a modified related list is offered to them, the data collected from our experiment suffices.

In our experiment, no content is added or removed from the original related list sent by the YouTube server, only the order of the videos on that list is randomly changed. Since we modify the related list at the client using a plugin and not a central proxy which intercepts the whole traffic from YouTube, we decided to reorder the related list randomly (in contrast to the approach presented in Section 6). Also, the goal of the experiment is to prove our hypothesis that users request videos from the top of the related list even after the related list is reordered. Hence, building an actual proxy cache that would implement the mechanism for local recommendation is beyond the scope of this work.

6.2.2 Test Candidates and Results

To perform the experiment with a wide set of audience, we distributed the plugin among friends, colleagues, and students without informing them about the intentions of our investigation. We simply asked them to install the plugin in their browser and watch YouTube videos. We have 40 active users using the plugin and collected the logs for about 2 months with 11872 YouTube requests made by the end of January 2014. Out of the 11872 YouTube requests, 39.49% of requests are from the YouTube related list. This confirms the results from our trace analysis presented in Table 4.1.

As mentioned earlier, our objective of this user study is to verify our hypothesis that YouTube users do select videos from the top of the related list even after provided with a modified related list. Figure 6.3 shows the CDF of the percentage of requests

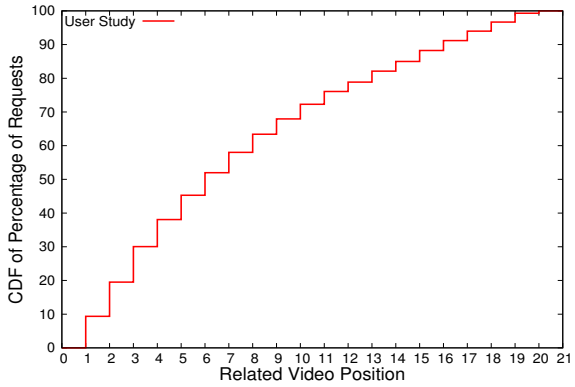


Figure 6.3. CDF of the percentage of requests per related video postion for each video requested from reordered related video list.

per related list position for the 4688 requests made by YouTube users from the related list using our plugin. Figure 6.3 shows that 73% of the YouTube requests from the related list offered are from the Top 10 positions of the related list. From the traces captured from a university campus, the percentage of requests from related lists constitutes 80% of the requests, whereas this percentage is 73% in our user study that presents users with reordered related lists.

Visually, it is obvious from figure 6.3 that the probability with which users select videos from the related list is similarly distributed as in the orignal trace. Consequently, we can say that even worst-case changes, i.e., random permutation of the related list, does not alter user behaviour in a major way.

However, we cannot claim statistical identity. We performed the Durbin test [52] on 4500 requests observed in the user study and on 4500 requests in trace T1, which was presented in Table 4.1. The Durbin test rejects the hypothesis of identical distributions for original and permutated related lists, suggesting that lower entries in the related list are chosen more frequently after a random permutation reordering. Still, the dominance of the entries higher in the related list remains clear.

Trace	Content Centric	Position Centric
T1	6.71%	11.83%
T2	4.71%	22.90%

Table 6.1. Comparison of hit ratio

6.3 Cache-based Reordering Results

We used the data from the campus network traces presented in Section 4.1.1 and information retrieved through the YouTube API [38] to perform a simulation of the cache-based reordering approach. We simulate the caching of videos which were requested in the traces. For example, if we detect a request for Video A in the trace, we log this video as stored in the cache. Since each YouTube video is assigned a unique identifier which is included in the video requests in the network trace, we can accurately simulate the behavior of the cache. Via the YouTube API, we retrieve the related video list for each video that is requested in the network traces. Each time we simulate a video request we simulate the modification of the related list (retrieved via the API) based on what is stored in the cache. For example, if a video in the related list obtained from the YouTube API is present in the cache, the video is placed at the beginning of the re-ordered related list and the videos that are not present in the cache are pushed downwards. It is important to note that we do not change the content of the related list obtained from the YouTube API at all. Only the order of the videos in the related list is manipulated. We simulate user behavior where the viewer selects the video from the same position in the related list as presented in the original trace. In this scenario, the content the user selects at that position in the re-ordered related list might be different from the original content that was requested in the trace. For this investigation, we use a custom-built simulator implemented in perl. The results from our analysis are summarized in Table 6.1.

Simulating a cache that employs the modification of the related video list with the content centric selection results in 7055 hits or a 6.71% hit ratio for trace T1 and 397

hits or a 4.71% hit ratio for trace T2. However, simulating a cache that employs the reordering of the related video list with the position centric selection results in 12431 hits or a 11.83% hit ratio for trace T1 and 1735 hits or a 22.90% hit ratio for T2. Although the hit ratio is not very high, this result along with our user study results from 6.2, affirming that users usually select from the top of the related list, shows the potential of our related list reordering approach to improve the performance of YouTube caches.

6.4 Discussion

In this section, we discuss the advantages and disadvantages of cache centric video recommendation. We address the issues of increased cache complexity, server load reduction and alternative sorting of the related list.

6.4.1 Why only YouTube?

Our work is mostly concentrated on YouTube video delivery system and taking advantage of YouTube related list to improve the effectiveness of caches. While our work uses only YouTube traces to analyze the impact of our related list re-ordering approach, the idea remains valid for any video on demand system which offers short video clips and related lists to their users to choose their next video. There are a few VoD systems in the web which offers similar services to YouTube such as DailyMotion, Vimeo, Vine etc. However, none of them are as popular for hosting short video clips as YouTube. With over 1 Billion average monthly visits, 1 Billion registered users and presence in over 61 countries around the globe[1], YouTube is by far the most popular and widely used user-centric video delivery system on the web. For these reasons, we have centered our approach and analysis of our re-ordering approach

[1]http://www.business2community.com/youtube/vimeo-vs-youtube-will-winner-emerge-2014-infographic-0864787#!bbzYQ8

around YouTube. However, the idea is still valid for any video delivery system which offers short-clips and provides a related list for each video watched by their users.

6.4.2 Cost of Recommendation List Reordering

We have shown that the cache hit rate is significantly increased when the related video list is reordered according to the cache content (see Table 6.1 for detailed data on hit rates), using the access patterns from our campus network traces. The cost of this process is a more complex cache server. For each request, the server must identify URLs from YouTube and upon success, the cache must be inspected to find whether the requested video is cached or not. The cost of an additional cache lookup is dependent of the cache structure and the size, but assuming a lookup structure based on a plain hash table, the average and worst case lookup times are O(1+n/k) and O(n) respectively, where n is the number of elements in the cache and k is the number of buckets in the hash-table. Adding for example a second data structure for the bucket list, like a self balanced tree, the theoretical worst-case time of common hash table operations can be brought down to O(log m) rather than O(m) where m is the number of elements in the hash-bucket.

Thus, the reordering comes at the described extra cache server cost. However, taking into account the very long tail distribution of videos, the gain in cache hit rate is substantial compared to the added processing. Additionally, there are several systems today that already rewrite web-page content. For example, Opera is currently rewriting and rearranging the content in web-pages using proxies for their opera mini browser [31]. Similarly, proxies like FilterProxy [17], Muffin [25] and WebCleaner [36] have the capability to modify content on the fly. We therefore claim that the proposed reordering of items in the related list is worth the additional complexity at the cache since this drawback is outweighed by the benefit of a significantly increased hit rate.

6.4.3 Reduction in Server Load

Another advantage of our related list reordering approach based on the content in the cache is the reduction in the server load, and hence, the reduction in back-end bandwidth consumption. With the reordering of the related list and pushing the contents from the bottom of the list to the top, we take advantage of the user behavior in selecting the videos from the top of the related video list. With reordering the list to contain the videos from the cache in the top of the related list, we reduce the load on the YouTube server (or higher level caches) proportional to the number of cache hit gains. From the analysis in Section 6.3, we show that with our reordering approach, the cache hits increase using the T1 trace is from 6.71% to 11.83% which turns into an approximately doubling of the hit rate. Similarly, the bandwidth consumption reduces from 93.29% to 88.17%, which provide a 5.12% reduction in server load and back-end bandwidth reduction from our related list reordering approach. Similarily, from the analysis of trace T2, we see a cache hit increase from 4.71% to 22.9% which is almost a five times increase on the hit rate. From the YouTube server point of view, this is a server load reduction of 18.19%.

6.4.4 Local Popularity Based Sorting of Related List Items

So far, the reordering of the related list does not take differences of local popularity of cached videos into account. As mentioned earlier, the related list reordering is based on a stable sort. This reordering could be further evolved by slightly modifying the sorting algorithm such that it also sorts the locally cached videos by their local popularity. Similar to the original reordering approach, the locally cached video would still be pushed to the top of the related list. But the order in this group of videos would now be ordered by local popularity. For example, if video C is more popular than video B, then video C would be ranked first in the related video list that is sent to the client. This approach, however, needs further investigation to study its

feasibility. First of all, the actual differences in popularity for videos that are stored on the local cache and belong to the related list of a video have to be determined. We believe that only significant differences in the popularity of the respective videos would render this approach feasible.

6.5 Summary

In this chapter, we took advantage of the YouTube user behavior to select their next video from the related list provided by Youtube, to modify the related list provided by YouTube based on the content already present in the cache. In our approach, the related list is modified only in the order of appearance on the user's webpage, not the content itself. During the modification, the related videos present in the cache, which are the videos already requested by other users, are moved to the top of the list and subsequently the related videos not cached are moved down the order. By analyzing a campus network trace filtered for YouTube traffic, we find that users generally request videos from the top half of the related list and hence moving the related videos already present in our cache makes the approach more advantageous.

We perform a user study to verify our hypothesis that users select from the top of the related list even after provided with a reordered related list. We perform our user study with 40 actual YouTube users to verify our hypothesis. The results from our user study indicates that even with reordered related list, 73% of the video requests from the related list are from the top half (Top 10) of the related list provided. Hence, we claim that the YouTube related list can be reordered based on the videos present in the cache without modifying the user behavior in selecting videos from the YouTube related list.

We analyze our approach in a simulation based study on the network trace captured on a campus gateway. We define two different approaches of video selection from the re-ordered related list. Content Centric selection, where the user selects the

same video as he would have originally selected and Position Centric selection, where the user selects a video from the same position on the related list before reordering, which might change the original content. From our simulation study, we find that Position Centric selection of the reordered related list yields a better hit rate than Content Centric selection, which does not take advantage of the content in the cache.

From the analysis of the caching techniques used by YouTube and our related list reordering approach, we conclude that reordering the related list presented to the user based on the content already requested by other users, yield a better hit rate of the caches, thereby reducing the bandwidth consumption by multimedia requests and hence the latency in content delivery.

CHAPTER 7

IMPROVING ENERGY EFFICIENCY OF YOUTUBE

Previous chapters showed that 50% of the YouTube video requests are selected from the related list offered by YouTube on a video's watch page. We have shown that this user behavior can be used to improve YouTube caching and presented the advantages of a new related list reordering approach which modifies the related list offered to YouTube users based on the contents already present in the YouTube caches. Recent reports [42] showed that in US alone data centers used a total of 91 billion kWh of electrical energy in 2013 and that number is expected to increase to 139 billion kWh by 2020, with an estimated annual electricity bill of greater than $13B primarily composed of "dirty" energy sources. These costs are unsustainable for even the highest value applications, with VoD applications being one of them. Hence any approaches or techniques to improve the energy efficiency of these applications reduces the overall energy footprint.

In this chapter, we will look at how the elasticity of cloud services, in conjunction with the related list reordering approach presented in Chapter 6 can be used to manage the YouTube long tail popularity distribution problem and reduce the energy of serving videos from the long tail popularity distribution of YouTube.

7.1 Related Work

In this section, we provide some background info on YouTube cache hierarchy to host videos and blinking of servers to maximize utilization of available power. We use

these two approaches in our work to improve the energy efficiency of serving YouTube videos.

7.1.1 YouTube Caching Hierarchy

Research [44, 43] has shown that YouTube employs a 3-tier caching hierarchy, which consists of a large set of primary caches located closer to the client, a smaller set of secondary caches and minimal number of tertiary caches around the world to serve billions of video requests every year. We verified this behavior from our own measurements on the origin of YouTube videos and their related videos as presented in Section 4.2.4. Figure 7.1 shows the architecture of video hosting by YouTube using their 3-tier cache hierarchy. Any requests from the clients goes through a YouTube proxy which selects a cache based on the location of the client, availability of the video and load balancing of requests. After the proxy selects a cache to serve the request made by the client, it sends a re-direction message to let the client know where to find the video. Once the client receives the cache hostname from where to download the video, it sends a HTTP request to the cache to play the video. In this dissertation, we use the same 3-tier hierarchy to simulate the scenario of hosting YouTube videos based on global popularity in our approach to reduce the energy required to host YouTube videos.

7.1.2 Blinking Servers

Blinking [91] refers to an abstraction where the server clusters adapt to intermittent power. Blinking dynamically regulates the energy footprint of servers, to match available power, by rapidly transitioning servers between high-power active state and a low-power inactive state. In this dissertation, we use the same abstraction of blinking, not to serve YouTube videos with intermittent power but to reduce the energy footprint by blinking servers or cloud instances to low-power inactive state when it is

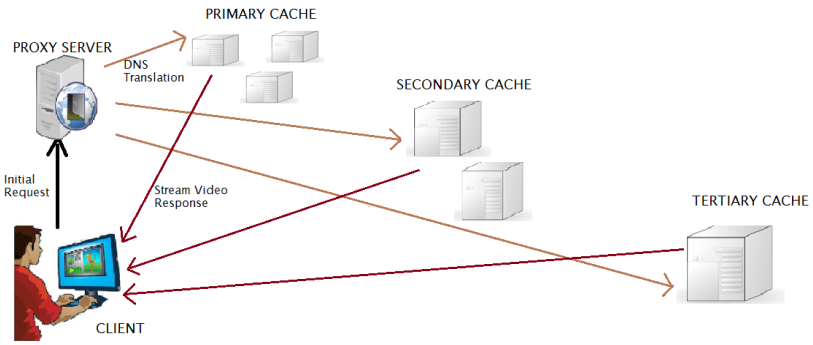


Figure 7.1. YouTube Cache Hierarchy Architecture

not serving any video requests. The instances are brought back to high-power active state when it is in need to serve the video requests.

In [92], we presented GreenCache, which leveraged the blinking abstraction to serve multimedia data off renewable energy sources. We presented different data hosting strategies based on user watching pattern to improve the hit rate of caches and reduce latency for clients despite fluctuating power conditions. However, the work presented in [92] had the goal to maximize the hit rate of serving videos with intermittent power. In this dissertation, we are interested in saving power by serving videos from cloud with no power limitations.

As mentioned in Section 1.2, 70% of the YouTube videos constitute the long tail of the overall popularity distribution, which are requested very few times. Therefore, it wastes power/energy to keep the servers which hosts the videos from the long tail popularity distribution running constantly. In this dissertation, we use the blinking abstraction and video hosting strategy based on global popularity of YouTube videos to reduce the energy footprint to serve video requests from the long tail popularity distribution of YouTube.

7.2 CloudTube

Having seen that YouTube uses a 3-tier cache hierarchy to host and serve their videos, we employ the same hierarchy to host YouTube videos on cloud instances. Since our objective is to reduce the energy consumed to serve YouTube videos and since highly popular videos (in terms of number of views) are requested more often than less popular videos, our approach is to host YouTube videos on cloud instances based on their global popularity. We call our architecture of hosting YouTube videos on cloud instances based on their global popularity *CloudTube*.

In our CloudTube architecture, the idea is to distribute the videos in 3 cloud instances based on the global popularity of the videos. We represent global views (number of views) of each video on YouTube as the global popularity of the videos. We obtain the global views of a video using the YouTube data API [38] and use our campus network YouTube traffic trace presented in Section 4.1.1 for our CloudTube analysis.

We divide the videos into 3 different popularity levels. i) Less than 100,000 global views stored in "low"cloud instance, ii) between 100,000 and 1,000,000 global views stored in "mid" cloud instance and iii) greater than 1,000,000 global views stored in "high" cloud instance. We chose these values to divide the videos based on popularity as they distribute the videos to all three cloud instances almost evenly. I.e., "low" cloud instance holds 41.59% of videos, 33.95% videos in "mid" cloud instance and 24.46% videos in "high" cloud instance. Once we have the videos distributed among the cloud instances, we step through the video requests in our dataset trace to determine i) the hit rate obtained by our video hosting strategy on each cloud instance and ii) the *uptime* of each cloud instance. Uptime refers to the amount of time the instance needs to be in active state consuming maximum power to serve all the requests. In our architecture, we blink the cloud instances hosting less popular videos ("mid" and "low" cloud instances), i.e., when these cloud instances are not

Global Popularity (Views)	Cloud Instance Type	Percentage of Videos	Storage Size	Hit Rate	Uptime (minutes)
< 100,000	Low	41.59%	1.09 TB	34.3%	2199
>= 100,000 and < 1,000,000	Mid	33.95%	1.05 TB	31%	2092
>= 1,000,000	High	24.46%	1.18 TB	34.7%	-

Table 7.1. Cache Hit Rate for CloudTube Architecture

serving any video requests, they are switched to low-power inactive state and those minutes are not counted as uptime of the instance. As we are interested in reducing the energy required to serve less popular videos, we measure the uptime of only "mid" and "low" cloud instances and keep the "high" cloud instance always active. Analysis results of our CloudTube architecture are presented in Table 7.1.

As seen from the results in Table 7.1, the hit rate (percentage of video requests served) obtained by the videos stored on each cloud instance based on the global popularity is 34.7%, 31%, 34.3% for "high", "mid" and "low" instance types respectively. This shows that, irrespective of the popularity of the videos and the number of videos stored, each of the cloud instance provides almost the same hit rate for the video requests in our dataset. This hit rate and video requests translates to 2092 and 2199 minutes of uptime for "mid" and "low" cloud instance types respectively. This uptime converts to 35 and 37 kWh (assuming, power for 1 hour of operation is 1kW) of power required to serve the videos from "mid" and "low" cloud instance types respectively.

The results from our video hosting strategy shows that highly popular videos yield the same hit rate as less popular videos, which constitute the long tail. This result shows that popularity based hosting of videos from VoD services does not yield any improvement in hit rate or saves energy in serving less popular videos. However, in [92], we showed that depending on the user watching pattern placing the first chunk (60 seconds) of all the videos on the server having the maximum power increases the hit rate and reduce latency in serving videos during intermittent power.

Also, in Chapter 6, we showed that YouTube users are more likely to select from the related list and from the top order of the related list provided by YouTube. We use these results to our advantage and reorder the video placement in our CloudTube architecture by moving the related videos of each video request already present in the cloud instances to "high" cloud instance (the instance with popular videos). Our hypothesis is that, users who are likely to select from the reordered related list are more likely to select the videos that are already in the cache, as shown from the results presented in Section 6.3.

In the next section, we make use of these strategies based on users pattern of watching YouTube videos and reordering of related list to host videos in our CloudTube architecture and analyze the advantages. Before we look at the modified CloudTube analysis, we present the user behavior of watching YouTube videos.

7.3 User Watching Pattern

In this section, we analyze user behavior in watching YouTube videos by looking into the user watching pattern once the viewer selects a video. To analyze the user watching pattern, we investigate if viewers switch to a new video before they completely finish watching the current video. In order to analyze this behavior, we look into the timestamps of a user requesting two consecutive videos from the YouTube trace introduced in Section 4.1.1. We calculate the interval between these timestamps and compare it with the duration of the initially requested video to determine if the user has switched between videos before the first video is completely viewed[1]. We perform this analysis for all the 16,284 unique users in our campus trace and the result is shown in Figure 7.2.

[1]We ignore the last video requested by a user, as we cannot affirmatively say if it was completely watched.

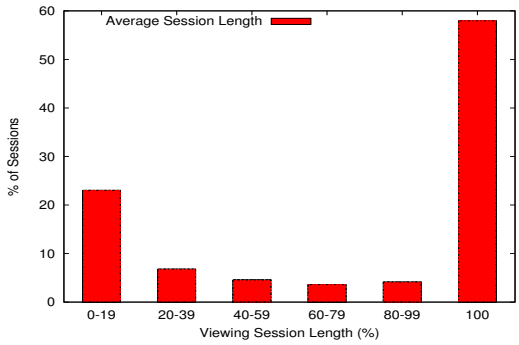


Figure 7.2. Session Length for Each Request

The figure shows the number of occurrences (in percent out of the total number of videos watched) a video is watched for $x\%$ of its total length. Results from Figure 7.2 show that, only in 58% of the cases videos are watched completely (this number is similar to the global study performed by Erman et al. [55]). In all other cases only part of the video is watched, with the majority of these cases ($\sim 25\%$) falling in the 0-20% session length category.

In the next section, we take advantage of this user watching behavior along with our related list reordering approach presented in Chapter 6 to improve the hit rate of "high" cloud instance and reduce the uptime of "mid" and "low" cloud instances in our CloudTube architecture.

7.4 CloudTube with Related List Reordering

The results in Section 7.2 showed that popularity based hosting of YouTube videos does not yield any savings in uptime or improve the hit rate of caches. However, results from the user watching behavior showed that only 58% of the video sessions are watched for completion and 25% of the video sessions are ended within the first 20% of the session length. Hence, in this section, we use this user watching pattern and the related list reordering approach presented in Chapter 6 to improve the hit

rate of “high” cloud instance of our CloudTube architecture and reduce the uptime of “mid” and “low” cloud instances.

Based on the user watching pattern analysis results, which suggests that 25% of the video sessions do not complete 20% of the session length, we place the first chunk (60 second segment) of all videos in the “high” cloud instance along with the video placement strategy (based on global popularity) presented in Section 7.2. Also, we reorder the related list for each video request based on the contents in each cloud instance and if the video is present, it is moved to the “high” cloud instance. Our hypothesis is that this placement of related videos which are already present in the cloud instances and the first chunk of all videos in the “high” cloud instance increases the hit rate of “high” cloud instance and also reduces the uptime of “mid” and “low” cloud instances, which hold the less popular videos. This reduction in uptime reduces the power usage of both “mid” and “low” cloud instances. We analyze our new video hosting strategy of placing the first chunk of all videos and moving related videos already in cache to ”high” cloud instance using our YouTube dataset and the results are presented in Table 7.2.

As seen from the hit rate column in Table 7.2, the hit rate for “high” cloud instance increases significantly from 34.7% to 66.24%, which is a 90% improvement over the previous video placement strategy. This increase in hit rate results in shorter uptime, 1713 and 1731 minutes for both “mid” and “low” cloud instances respectively. This is because 25% of the user sessions do not complete the first 20% of the video session length. Therefore the amount of time “mid” and “low” cloud instances are required to be in active state to serve the content reduces considerably with this video placement strategy. This reduction in uptime translates to reduction in power required to serve the content from “mid” and “low” cloud instances by almost 20%.

Hence, we suggest that placing the first chunk of all videos and highly popular videos in the primary caches and the less popular long tail videos on secondary or

Global Popularity (Views)	Cloud Instance Type	Storage Size	Hit Rate	Uptime (minutes)
< 100,000	Low	1.09 TB	16.79%	1731
>= 100,000 and < 1,000,000	Mid	1.05 TB	16.97%	1713
>= 1,000,000	High	1.85 TB	66.24%	-

Table 7.2. Cache Hit Rate for CloudTube Architecture with Reordering

tertiary cache cloud instances yield higher hit rates and reduces the uptime and power required to serve unpopular videos, which constitute the long tail in VoD services such as YouTube.

7.5 Summary

In this chapter, we presented our CloudTube architecture to host YouTube videos in the cloud based on 3-tier cache hierarchy used by YouTube. In this hierarchy, we suggest hosting the videos based on their global popularity (number of global views) on each cloud instance. The videos with higher popularity are hosted closer to the client and the videos with low popularity, which constitute the long tail farther away from the client.

We evaluate the percentage of video requests, hit rate and the uptime of each cloud instance for this video hosting strategy based on our YouTube dataset. We obtain almost the same percentage of video requests for videos hosted on each cloud instance irrespective of their popularity. In order to improve the video hosting strategy and increase the hit rate for videos with high popularity and reduce the uptime for cloud instances hosting medium and low popular videos, we looked at the user watching pattern of YouTube videos. We found that 58% of the video sessions are not completed and 25% of the sessions are ended within the first 20% of the video session length. We try to use this user watching behavior and the related list reordering approach to improve the effectiveness of our CloudTube architecture.

Based on the user watching pattern results, we place the first chunk (60 seconds) of each video in our trace in the high popularity cloud instance, along with the video

placement strategy already used. Also, we use the related list reordering approach to move the related videos which are already present in our cloud instances to high popularity cloud instance if they are not already present. We evaluate this video hosting strategy and find that the hit rate for the high popular cloud instance increases by 90% and the uptime for medium and low popularity cloud instance decreases by almost 20%, which in turn reduces the power required to serve the contents by almost 20%.

Hence, we suggest that placing the first chunk of all videos and highly popular videos in primary caches and the low popular long tail videos on secondary or tertiary cache cloud instances yield higher hit rate and reduces the uptime and power required to serve low popular videos, which constitute the long tail in VoD services such as YouTube.

Our approach of hosting videos in the cloud based on their popularity does not have any cost saving advantages. Current cloud services have a pay model where the cloud instances can be rented on per-hour basis. The time interval between video requests in the cloud instances is not more than few minutes. Hence, even if the instances are shut down when there are no video requests to be served, the cloud services bills the user for the whole hour. However, if the cloud services change their pay model to per-minute usage or even smaller periods then our approach can also be used to reduce cost.

CHAPTER 8

ONLINE TRANSCODING

Previous chapters looked at using the related list provided by VoD applications (especially YouTube) to improve the effectiveness of caches deployed to serve their content. With the rise in popularity of adaptive video bit rate streaming, the amount of content to cache and store has increased significantly. Adaptive bit rate streaming (ABR) involves maintaining multiple copies of one video file to serve clients with different devices, network conditions, screen sizes etc.,. However, since VoD applications follow a long tail popularity distribution, not all videos are requested in all the different bit rates supported by the VoD content provider. Hence, online transcoding of videos whenever requested is a possibility to reduce the transcoding workload and reduce storage space required to support ABR streaming.

In the next two chapters, we present the need for online transcoding, explain the transcoding architecture and the transcoding policies possible. We analyze the workload reduction possible based on our transcoding policies using a dataset from a large CDN provider. We present the prediction models used to predict the future segment ahead and analyze the performance of online transcoding at the client.

8.1 Adaptive Bit Rate (ABR) Streaming

ABR streaming is realized through video streaming over HTTP where the source content is segmented into small multi-second (usually between 2 and 10 seconds) segments and each segment is encoded at multiple bit rates. Before the actual streaming process starts, the client downloads a manifest file that describes the segments and

the quality versions these segments are available in. After receiving the manifest file, the client starts requesting the initial segment(s) using a heuristic that depends on the video player implementation. For instance, it may start by requesting the lowest bit rate version for the first segment. If the client finds that the download bandwidth is greater than the bit rate of the current segment, it may request future segments in the next higher quality version. In the case where the client estimates that the available bandwidth is lower than the bit rate of the current segment, it may request the next segment in a lower quality version. With this approach the streaming process can be adapted to the available download bandwidth, which minimizes the amount of rebuffering that might have to be performed at the client.

Several different implementations of ABR streaming exist, including Apple HTTP Live Streaming (HLS) [9], Microsoft Live Smooth Streaming (Silverlight) [24] and Adobe Adaptive Streaming (HDS) [2]. Each have their own proprietary implementation and slight modifications to the basic ABR streaming technique described above. Recently, an international standard was accepted for HTTP-based adaptive bit rate streaming called MPEG-DASH [96]. DASH is an open source MPEG adaptive streaming standard developed for the streaming of media content from web servers via HTTP. The basic approach of DASH is similar to all other proprietary ABR streaming standards described above.

DASH consists of two components: A Media Presentation Description (MPD) manifest file which contains the description of the available content, their URL addresses, different bit rates or qualities available, segment lengths and other characteristics; and video segments, which contain the actual multimedia bit streams in the form of single or multiple files based on the encoder [72]. With the DASH standard, a video is encoded into different representations (e.g., different video qualities). The DASH client downloads the MPD in order to retrieve a complete list of components, including video, audio and subtitles. The DASH client then determines the pre-

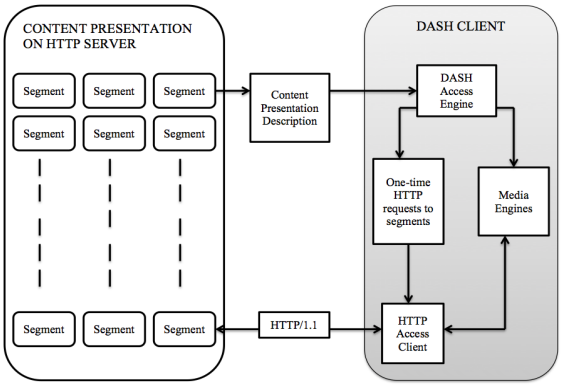


Figure 8.1. DASH Flow Process

ferred set of representations based on its capability (e.g., resolution), user preference (e.g., language), and network bandwidth (e.g., high or low bit rate). And finally, the DASH client downloads the desired spliced content from the HTTP server and plays the content. Figure 8.1 depicts the streaming flow process in DASH.

DASH videos are usually encoded with different segment lengths. Commercial approaches use segment lengths ranging from 2 seconds per fragment (e.g., Microsoft Smooth Streaming [24]) to 10 seconds per fragment (e.g., Apple HLS [9]). The segment length is an important factor when providing content for adaptive streaming. Shorter segments may provide a larger overhead in term of requests and result in more quality switches. Longer segments are not efficient in high bandwidth fluctuating environments like mobile networks, because the client cannot adjust the video stream as fast as necessary in case of a significant bandwidth change. Hence, it is important to decide on the segment lengths which suit an application or network connection best.

8.2 Transcoding Challenges

ABR streaming requires the creation of video content in multiple bit rates, which translates to multiple video files for the same video content. The primary transcoding challenge is that the number of formats and bit rates that need to be supported is

very large, given the wide variety of users and devices. As a result, transcoding is very resource intensive and any reduction in the transcoding work can lead to significant cost savings. We take as examples three large video providers (YouTube, Netflix, and Hulu) and the largest video CDN (Akamai) to demonstrate the wide range of formats and bit rates supported by these services.

YouTube. YouTube, the world's largest provider of user-generated videos, offers a variety of qualities and encoding formats for the same video as presented in [41]. The different formats for a video include Flash (flv/mp4), HTML5 (webm), Mobile (3gp), DASH (mp4) and 3D (mp4). Depending on the original source of the video uploaded by a user, each of these formats may be available in 5 to 6 different qualities. Regular Flash and DASH videos are available in 144p, 240p, 360p, 480p, 720p, and 1080p qualities. In rare cases, videos are even available in 4096p quality. Hence, to serve the same video in different formats and qualities, the original content has to be transcoded to more than 20 different versions. With over 1 billion videos in YouTube's library, converting all videos to multiple formats before they are requested is not effective. Considering the extreme long-tail popularity distribution of YouTube videos, immediately transcoding videos in all potential format and quality versions wastes storage space and transcoding resources.

Netflix & Hulu. Netflix and Hulu two of the largest entertainment video sites in the world. Both of these video providers use ABR streaming standards to serve their content. Netflix uses Microsoft Smooth Streaming whereas Hulu employs Adobe Flash. Netflix offers video qualities that require download speeds ranging from 1.5 Mbps to 25 Mbps whereas Hulu video qualities range from 640 Kbps to 1.4 Mbps. Each of these providers make their content available on multiple codecs, screen resolution, devices, etc. According to Netflix, the vast number of codec and bit rate combinations can result in up to 120 transcoding operations for the same title before it can be delivered to all potential client platforms in all supported quality ver-

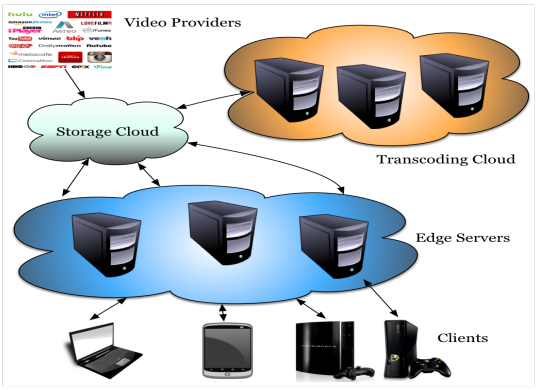


Figure 8.2. Online Transcoding Architecture and SLA

sions [27]. Having a large number of different quality versions for each video imposes a high transcoding workload and requires significant storage.

Akamai. As a CDN, Akamai supports the formats and bit rates required by the hundreds of major video providers who are their customers [4]. Video providers upload each video in one of the supported input formats that include aac, avi, dif, f4v, flv, m4a, m4v, mov,mp4, mpeg, mpegts, mpg, Mxf, ogg, webm, wmv, etc. Each input video needs to be transcoded to multiple bit rates and multiple output formats that include fragmented MP4, HDS, HLS and Smooth.

8.3 Transcoding Architecture

We provide an overview of the transcoding architecture that is typical in a CDN. As shown in Figure 8.2, the transcoding architecture consists of the following components:

1. *Storage Cloud.* A video provider publishes a video in a single high quality format by uploading it into the storage cloud. Further, the transcoding cloud could download videos from the storage cloud, transcode those videos to multiple bit rates, and upload it back to the storage cloud.

2. *Transcoding Cloud.* The transcoding cloud consist of a set of servers that run software that can perform the task of transcoding of the video segments.

3. *Edge Servers.* These servers are widely deployed by the CDN in hundreds of data centers around the world and is used for delivering the videos to clients from proximal locations. Each edge servers has a cache for storing the video segments.

When a video provider wants to publish a video, the provider uploads a single high quality version of the video to the cloud storage of the CDN. When a client plays a video, the following occur.

1. The CDN directs the client to a nearby edge server from which video segments can be downloaded. The client makes a sequence of requests for video segments at specific bit rates to that server as play progresses.

2. If the edge server has the requested segment in cache, it is delivered to the client. Otherwise, the edge server downloads it from the storage cloud, caches that segment, and serves it to the client.

3. When the storage cloud receives a request from an edge server, it checks to see if it has the requested video segment in the requested bit rate. If it does not, it sends the uploaded version of the video segment to the transcoding cloud. The transcoding cloud transcodes the segment to the requested bit rate and sends it back to the storage cloud.

8.3.1 Transcoding Policies

A *transcoding policy* is a scheduling policy that dictates when video segments are transcoded by the cloud transcoder. Note that any policy should meet the *transcoding SLA* that is an agreement between the video provider and the CDN that determines how quickly a newly uploaded video is available for access by users. A typical SLA guarantees that a video of duration D is available within time D/s for users to

download and is termed an `1/s` SLA, e.g., a `1/2` SLA guarantees that a 30-minute video uploaded at the time t is available for users at time $t + 15$ minutes.

We explore three types of policies: offline, online, and hybrid. There are two key dimensions on which a policy can be optimized. First, a policy can minimize the amount of transcoding work that it performs. Note that a reduction in transcoding work directly translates to a lesser amount of resources that need to be provisioned for transcoding and storage. Next, the transcoding policy should maximize video performance by reducing the likelihood of rebuffer events in the play out. Exploring the tradeoff between the transcoding work and video delivery performance is the focus of this work.

1) Offline Policy. The current defacto standard for transcoding in the industry is the offline policy. When the video provider uploads a new video to the storage, it is sent to the transcoding cloud where the video is transcoded into *all* the bit rates specified by the video provider. The transcoded videos are then uploaded back to cloud storage. The CDN delivers the video to users after the transcoding process is complete. As seen in Section 8.5, offline could do a substantial amount of extra transcoding work, but has good video performance since the video segments requested by clients are always immediately available in the requested quality level.

2) Online Policy. At the other extreme, we propose the *online policy* where nothing is transcoded proactively when the video provider uploads a new video to the storage cloud. When a client plays a video, it requests video segments in sequence from a "proximal" edge server chosen by the CDN. If the requested segment $S(x, y)$ (where x is the bit rate of the segment and y is the segment number) is not present in the edge server or in the storage cloud, a segment transcoding request is sent to the transcoding cloud. The transcoding cloud, upon receiving the request for transcoding segment $S(x, y)$ downloads the original video file uploaded by the video provider from the storage cloud and starts the transcoding process. Once the transcoding of segment

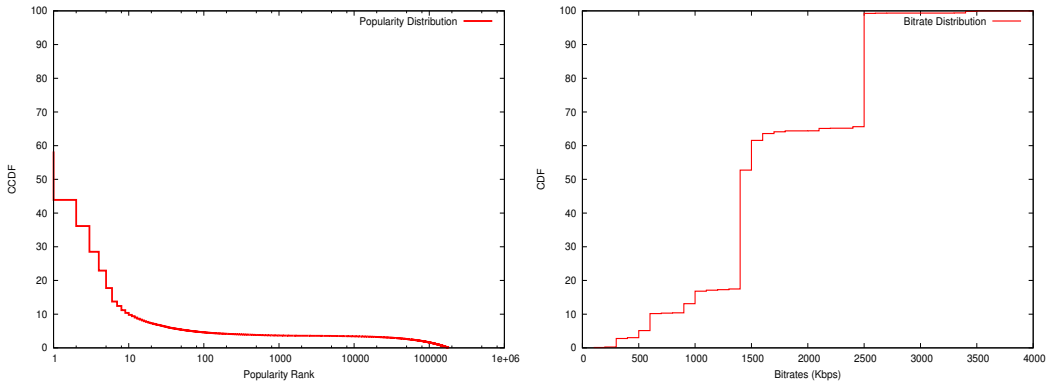
$S(x, y)$ is completed, the segment is stored in the storage cloud, which is then pulled by the edge server and delivered to the client. The video segment $S(x, y)$ is now available in the storage cloud permanently.

It is clear that the online policy performs much less transcoding work than the offline policy, as it seldom transcodes a segment to a bit rate that is not requested by the client. But, the challenge is to minimize the video performance degradation it might cause. Note that the policy needs to perform the transcoding in real time or even faster than real time. This is required to assure that no additional rebuffering – which might eventually lead to pauses in the video play out – occurs at the client. Earlier work [69] has shown that rebuffering has a strong adverse effect on the viewer experience.

3) Hybrid Policies. Offline and online transcoding are two extremes. We propose a family of hybrid policies that combine aspects of both. Specifically, an `x/y` transcoding policy transcodes the first `x`% of the video to *all* the desired bit rates in an offline fashion before delivery begins. Further, it transcodes the remaining `y`% of the video segments in an online fashion to only those bit rates that are (or, likely to be) requested by the client. Note that `100/0` hybrid policy is simply the offline policy and `0/100` is the online one. Besides the above family of hybrid policies, we also propose and study a specific hybrid policy called `1Seg/Rest` which transcodes only the first video segment of all videos to all the desired bit rates in an offline fashion, and transcodes the rest of the segments in an online fashion.

8.4 Our Data Sets

To analyze the benefits of online transcoding, we collected extensive, anonymized logs of how users access videos from Akamai's video CDN. Akamai [83] is one of the largest CDNs in the world and delivers 15–30% of global Internet traffic consisting of videos, web, software downloads, social networks, and applications. Akamai has a



(a) CCDF of the popularity of videos requested in the trace.

(b) CDF of the segment video bit rates requested in the trace.

Figure 8.3. Popularity and Bit rate distribution of video requests in our trace.

large distributed platform of over 150,000 edge servers deployed in 90+ countries and 1200 ISPs around the world. The anonymized data sets that we use for our analysis were collected from a large cross-section of actual users around the world who played videos using video players that incorporate the widely-deployed Akamai's client-side media analytics plug in.

Our client-side measurements were collected using the following process. When video providers build their video player, they can choose to incorporate the plugin that provides an accurate means for measuring a variety of video quality and viewer behavioral metrics. When the user plays a video, the plugin is loaded by the user's video player. The plugin "listens" and records a variety of events that can then be used to stitch together an accurate picture of the play out. In our case, we are primarily interested in the url of the video being played and the bit rate of each video segment fetched by the video player. This provides us with complete information on what video segments were accessed, when they were accessed, and what bit rate versions of these segments were downloaded. Once the metrics are captured by the plugin, the information is "beaconed" to an analytics backend that we can use to process the huge volumes of data.

8.4.1 Data Set Characteristics

We extracted a characteristic slice of user video requests from across Akamai's global CDN over a 3-day period in June 2014. When collecting the traces we ensured that we had a representative sampling of all types of videos, including short-form (e.g, news clips, sports highlights), medium-form (e.g, TV episodes), and long-form (e.g, movies) videos. We also only included video providers who use ABR streaming, such as HLS, HDS, Silverlight, etc. Overall, we analyzed traces from 5 million video sessions originating from 200 thousand unique clients who were served by 1294 video servers from around the world. The videos requested belong to about 3292 unique video providers and include every major genre of online videos.

Figure 8.3(a) shows the complementary cumulative distribution function (CCDF) of the popularity of the requested videos. The figure shows that there are about 45% of the videos watched multiple times and a long tail of videos which are watched only once. Hence, transcoding the videos in the long tail in an offline fashion to all the bit rates wastes both transcoding and storage resource.

Figure 8.3(b) shows the distribution of the video bit rates requested by the clients in this trace. We see that the bit rates of the videos requested range from 100 Kbps to 4000 Kbps. Also, the figure shows that most of the video segment requests are for medium (1500 Kbps) and high (2500 Kbps) bit rates. In particular, about 70% of the video segment requests are only for two bit rate ranges. This observation provides motivation for constructing a good markovian predictor for the bit rate of the next video segment that we discuss in detail in Section 9.1.

We also investigate how much of the video a user watches by measuring the total time the user watches a video in comparison with the total duration of the video. Figure 8.4 shows percent of viewers who abandoned the video at each stage in the video. We see that 70% of the video sessions reach the very end of the video and watch beyond the 80% mark. However, 18% of the video sessions are abandoned in the first

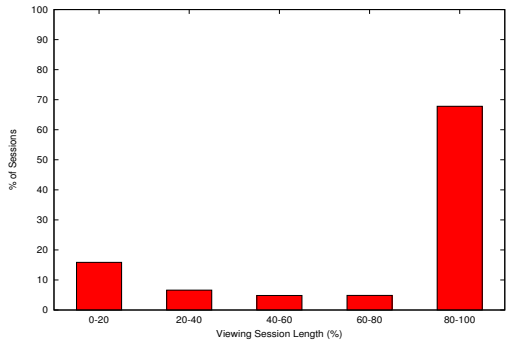


Figure 8.4. An analysis of what fraction of a video is watched by what fraction of users.

20% of the video. This suggests the hybrid schemes that we study in Section 8.5 where the initial portion of the video that is watched more often can be transcoded in an offline fashion to all possible bit rates, while the rest can be transcoded in an online fashion only as needed.

8.5 Transcoding Workload Analysis

Using the CDN traces described in Section 8.4, we simulate several transcoding policies and evaluate the workload induced by each policy on the transcoding cloud. Given that transcoding is resource intensive, any reduction in workload leads to a significant decrease in the transcoding cloud resources that have to be provided, further leading to significant cost savings. Throughout the analyses of our online transcoding policies, we make the following assumptions.

- When the transcoding cloud receives requests for transcoding multiple video segments, it must schedule these requests using a scheduling policy. We employ the Earliest Deadline First (EDF) [94] policy for each video transcoding request. EDF is a natural choice since it schedules the process that is closest to its deadline first, where the deadline for the transcoding request is dictated by the transcoding SLA that must be satisfied. EDF has been successfully applied for other problems in multimedia like disk scheduling [95].

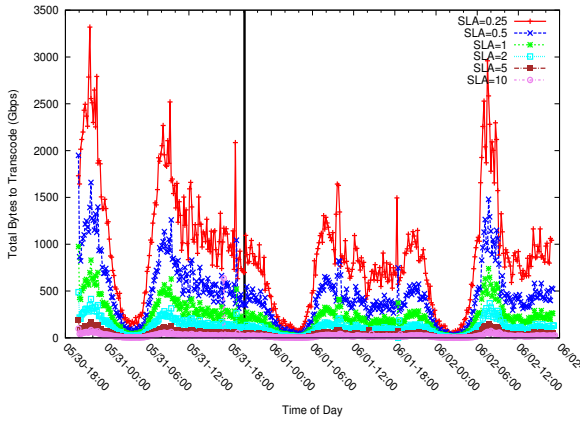


Figure 8.5. Workload of offline transcoding per SLA.

- All our analyses are performed for the whole 3-day dataset. However, barring the plot for Figure 8.5 which has the results for all 3 days (end of day 1 is represented by a vertical line), all other plots are shown only for the next 2 days. This is because we start the simulation with no transcoded video segments in the storage cloud, inducing much additional transcoding workload for the hybrid and online policies on the first day. This additional workload is not typical in a real system, since the storage cloud will always possess some video segments that have already been transcoded in the past. Thus, we let the cloud storage warm up in the first day and show the results for the next two days which is much more typical.

- We assume a segment length of 6 seconds for each video segment request. This is because typically segment lengths are between 2 and 10 seconds. We use the median of the range (6 seconds) as our segment length for our analyses.

8.5.1 Offline Transcoding

Figure 8.5 shows the workload of offline transcoding for six different SLAs as a time sequence. For notational convenience, we express SLA's as a fraction, e.g, a `1/4` SLA is also represented as .25. An SLA of 1 is real-time transcoding, whereas an

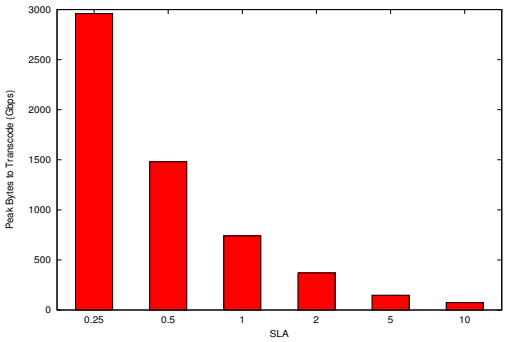


Figure 8.6. Peak workload of offline transcoding per SLA.

SLA less than 1 is faster than real-time[1]. Note that the amount of bytes to transcode follows a diurnal pattern, where there is a small amount of bytes to transcode in the night and peak throughout the day. This pattern is due to the fact that there are fewer video uploads from video producers during the night and this pattern is seen throughout our results.

The results in Figure 8.5 also show that a more stringent SLA (e.g., $SLA = .25$) generates significantly more workload than a more flexible one. In fact, transcoding videos faster than real time is quite costly, since the reduction in workload between $SLA = .25$ and $SLA = .5$ is the largest.

In addition to the workload generated over time we show the peak workload for $SLA = .25, .5, 1, 2, 5, 10$ in Figure 8.6. Besides the total workload, the peak workload is also important since the transcoding cloud (like other deployed systems) is provisioned for the peak. This figure shows the rapid decrease of the peak as the SLA becomes more flexible. Note that faster than real time transcoding ($SLA = .25, .5$) comes at a very high cost, which the CDN will most likely have to pass on to the video provider.

[1]By "faster than real time" we mean that the transcoding of a video segment is performed faster than the actual play out of that segment. E.g., in the case of a 6 second segment the transcoding would be performed in less than 6 seconds.

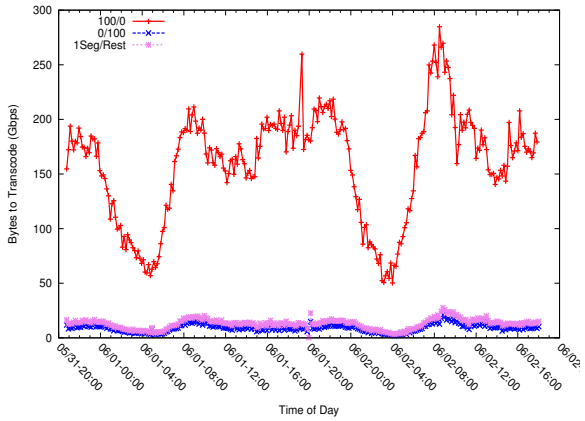


Figure 8.7. Workload of the 0/100, 1Seg/Rest and 100/0 transcoding policies.

8.5.2 Online and Hybrid Transcoding

Figure 8.7 shows the total amount of bytes to transcode for 0/100 and 1Seg/Rest transcoding in comparison to offline transcoding (100/0). The difference in amount of bytes to transcode is significantly higher in the offline case, which shows that not all videos are requested in all the bit rates supported by the video providers. Also, the 1Seg/Rest policy adds minimal extra load compared to the online (0/100) approach. The peak total bytes to transcode for 100/0 transcoding is about 300 Gbps, whereas the peak for 0/100 transcoding is about 11 Gbps. This huge difference in workload indicates the significant amount of transcoding resources and storage space that can be saved by employing pure online transcoding. Keeping in mind that the transcoding cloud must be provisoned for the peak case, online or 1Seg/Rest transcoding has the potential to lead to immense cost reduction.

Also, as seen from Figure 8.7, the peak total amount of bytes to transcode with the 1Seg/Rest policy is about 15 Gbps, which is only 4 Gbps more than the online (0/100) transcoding policy. In Section 9.2.2, we look at the performance in terms of rebuffer ratio at the client and show that 1Seg/Rest provides a much better viewing experience than the pure online policy. Thus, it can be argued that the extra work induced by 1Seg/Rest might be worthwhile.

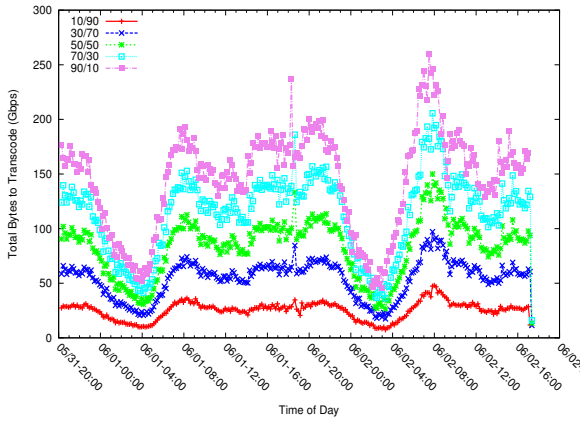


Figure 8.8. Workload of hybrid transcoding policies.

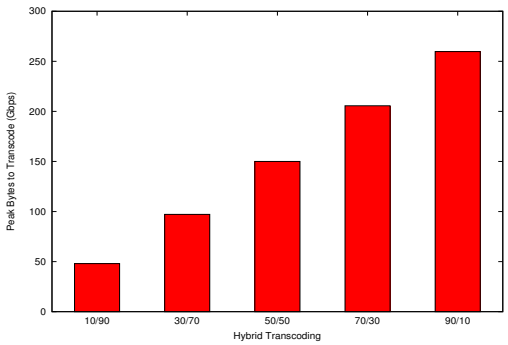


Figure 8.9. Peak workload of hybrid policies.

Figure 8.8 shows the workload induced by other hybrid transcoding (`Offline`%/`Online`%) strategies. As expected, the higher the percentage of offline transcoded segments, the higher the amount of bytes to transcode. However, since not all videos are completely watched and not all videos are requested in every bit rate the videos are offered in, the increase in bytes to transcode for each hybrid strategy is not linear. As we move to higher offline and lesser online hybrid approach, e.g., 70/30 or 90/10, the amount of savings in bytes to transcode decreases compared to, e.g,10/90 or 30/70 hybrid transcoding. Figure 8.9 shows the peak bytes transcoded by each hybrid policy. As seen in the figure, the peak increases linearly with increase in the percentage of video segments offline transcoded.

8.6 Summary

In this chapter we presented our transcoding architecture and policies to transcode video segments on-demand. We presented 3 different transcoding policies, offline, online and hybrid transcoding policies. We analyzed the workload generated from each of these policies using a 3 day dataset of adaptive bit rate video requests collected from Akamai Technologies. Our dataset contains 5 million video requests from 200K clients and 3292 video providers. Based on the analysis of our dataset, we found that only 2 bit rates make up 70% of the video segment requests. Also we found that 30% of the video sessions are not completed by the clients. This showed the potential for online transcoding and savings in transcoding workload and storage space.

Our workload analysis for each of the 3 different transcoding policy showed that online transcoding policy has the potential to reduce the transcoding workload significantly. According to our analysis, the hybrid transcoding policy with the first segment of each video pre-encoded to all the bit rates has the potential to reduce the transcoding workload by 95% of that required for pure offline transcoding.

CHAPTER 9

PREDICTIVE TRANSCODING AND PERFORMANCE

The previous chapter provided an overview of the transcoding architecture and motivated the need for online transcoding and different online transcoding policies. Also, we presented the data set we use for the analysis of our online transcoding policies and presented the analysis of the workload generated by the different transcoding policies. In this chapter, we present the prediction models we use in our online transcoding policies and present the performance analysis of our online transcoding policies at the client.

9.1 Predictive Transcoding

Results from the previous chapter showed that online and hybrid transcoding approaches can reduce the workload significantly. While this can result in a huge savings in the cost of transcoding, it comes with the drawback that online or hybrid transcoding might lead to impairments at the client. For example, if a segment is not transcoded on time it might not arrive at the client on time to be played out. This can cause the client to rebuffer, resulting in an inferior video viewing experience for the user.

One potential approach to prevent this issue is to predict the bit rate for the next segment the client might request. The prediction of future bit rate requests allows us to transcode the segment to that bit rate one segment ahead, so the client does not have to wait for the next segment to be transcoded once requested. Our proposed predictive transcoding involves the following steps.

1) Prediction Step. When the client plays a video, the prediction algorithm is used to predict the bit rate of the next video segment that will be requested by the client.

2) Transcode Step. Based on this prediction, we check to see if that video segment is already available in cloud storage at the predicted bit rate. If not, we proactively transcode the next video segment to the predicted bit rate.

3) Delivery Step. If the prediction was accurate and the transcoding of the next video segment completes *before* the client requests that segment, the CDN can deliver that segment to the user without triggering a rebuffering event. However, if the prediction is incorrect and the video segment is not available in the required bit rate, a rebuffer might occur.

The main challenge of this approach is to achieve a prediction accuracy that results in high-performance online or hybrid transcoding. To achieve this goal, we analyze two prediction methods, a simple one and a complex one. The simple method predicts the bit rate of the next segment to be identical to the bit rate of the current segment requested. The complex method is based on a Markov model which uses a state machine that keeps state on previous video request bit rates to determine the most likely bit rate to be requested in the future.

In Section 9.1.1, we introduce the Markov model used in our work in more detail and then provide the prediction analysis results for online transcoding using the Akamai dataset in Section 9.1.2.

9.1.1 Markov Prediction Model

A Markov model involves maintaining a finite state machine, which is used to predict future states based on the current state of the process. In our Markov prediction model, the states represent the different quality versions of a video and the state change probability is the probability with which a video switches from one bit rate

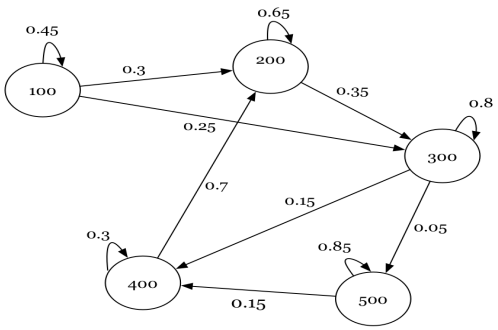


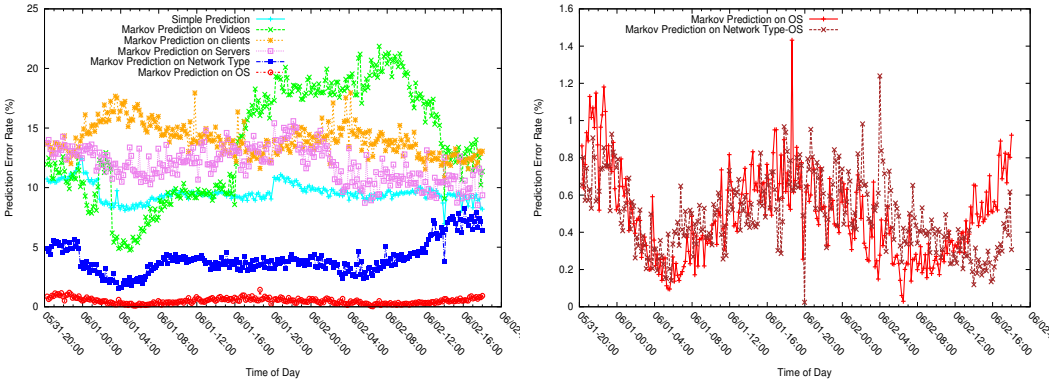
Figure 9.1. Markov Finite State Machine Example.

to another. We make use of the Markov prediction model by maintaining a $N \times N$ matrix, where N represents the maximum number of quality versions each video is transcoded to. To populate the Markov state matrix, we step through the video requests in the Akamai dataset. For each video request, we look into the per segment quality version requested and modify the matrix probability for each current bit rate. Based on the state change probabilities of all the bit rates for the current video, we predict the future bit rate based on the most probable state change based on current state (bit rate).

Figure 9.1 shows an example finite state machine for a video with quality versions (100, 200, 300, 400, 500 Kbps). Each state change has a certain probability associated with it based on past requests. The probability matrix for the finite state machine in Figure 9.1 is a 5×5 matrix as shown below.

$$P = \begin{bmatrix} 0.45 & 0.3 & 0.25 & 0 & 0 \\ 0 & 0.65 & 0.35 & 0 & 0 \\ 0 & 0 & 0.8 & 0.15 & 0.05 \\ 0 & 0.7 & 0 & 0.3 & 0 \\ 0 & 0 & 0 & 0.15 & 0.85 \end{bmatrix}$$

If the current bit rate is 300 Kbps, then according to Markov prediction model, the maximum probability of the next bit rate state is staying at 300 Kbps, with a



(a) Prediction error of each prediction model. (b) Prediction error of top prediction models.

Figure 9.2. Prediction Error of different prediction models.

probability of 0.8. Hence the prediction model predicts 300 Kbps as the next bit rate to be requested from the client.

We use this prediction model for our online transcoding analysis and the results from the prediction techniques are presented in the next section.

9.1.2 Prediction Analysis Results

In this section, we analyze the prediction techniques mentioned in the previous section using our dataset described in Section 8.4. For each prediction model, we analyze the prediction accuracy in terms of the error rate and the amount of bytes to transcode for each instance of video requests, where each instance is a 10-minute sequence of video segment requests from the trace[1]. A prediction error is said to occur when the predicted bit rate during the current video segment request is not the same bit rate requested next by the client during the same video session. Error rate is defined as the total number of prediction errors over the total number of predictions.

For the prediction model analysis we investigate four different categories of Markov models which are based on per video bit rate requests. We chose the categories based on network and client parameters and they are presented in Table 9.1. For each of

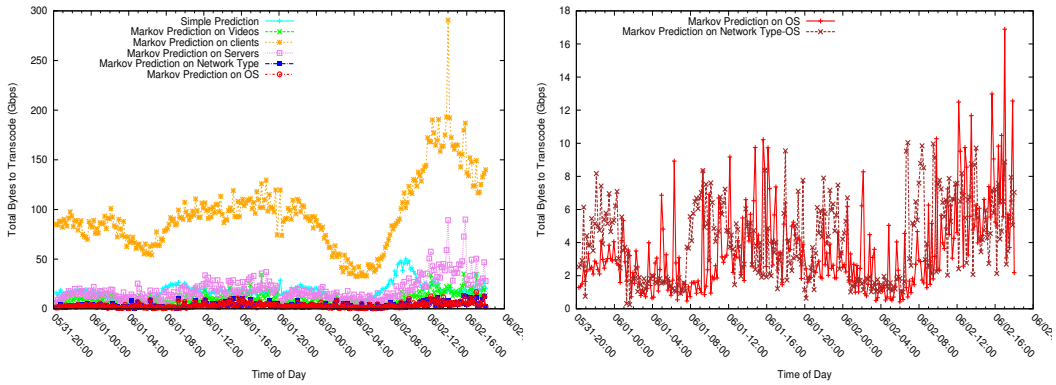
[1]The 10-minute video sequence as an instance is only used for data representation and does not interfere with our analysis.

Category	Description
Client	The IP address of the client requesting the video.
Server	The IP address of the edge server serving the video request.
Network Type	Type of access network client is connected to.
OS	Operating system used by the client making the video request.

Table 9.1. Network and Client Categories used in Markov model.

these categories, we create a Markov model on a per video request. For example, client category results in a client-video tuple requests (per client per video) model and server category results in a server-video tuple request model. For each of these models, the states are represented by the requested video bit rates but for each approach there will be n Markov state machines, where n reflects the number of groups per category (e.g., network, OS, etc.) For example, in the OS category case we have about 10 different OS groups (OS X, Windows, Android, IOS, etc.), which results in the same number of Markov models. E.g., If a request from a iPhone is made, the model for iOS is used. We start the prediction from the second video requested for each group in each category, as the first video is used to model the state machine. Also, we use 1Seg/Rest transcoding policy throughout our prediction model analysis.

Figure 9.2(a) shows the prediction error rate for all prediction models. As seen from this figure, the Markov model based on client OS (per OS per video) yields the lowest prediction error rate of 0.5% followed by the Markov model on network type at around 4%. However, all the other Markov prediction models yield almost the same prediction error at around 10% to 15%. The simple prediction model, the most naïve of all models, which predicts the next bit rate to be the same as the current bit rate results in error rates around 10%, which is lower than some of the Markov-based prediction models. We conjecture that this is caused by the fact that (as shown in Figure 8.3(b)) 70% of the requests are made up of just two bit rates. Hence in most cases predicting the next requested bit rate to be same as the current bit rate yields



(a) Total workload for online transcoding of each prediction model.

(b) Total workload for online transcoding of top prediction models.

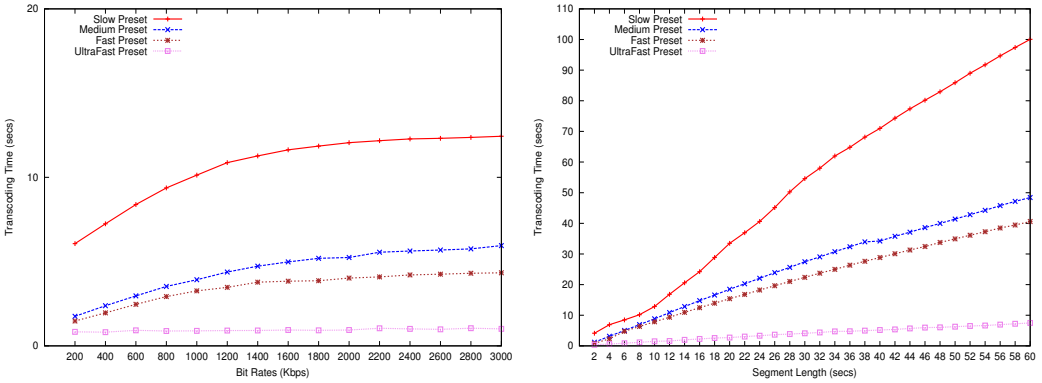
Figure 9.3. Total workload for online transcoding of different prediction models in Gbps.

a low prediction error rate assuming that the quality switches in a streaming session are not performed as often as one might expect in an adaptive streaming scenario.

Since, the Markov models based on OS and network type yield the lowest prediction error rates, we combine these two parameters and create a new Markov model based on network type and OS (per network-type per OS per video). The results for this combined model are shown in Figure 9.2(b). As seen from Figure 9.2(b), the prediction error rate for the combined model of network type-OS decreases slightly compared to the OS only model by 0.2%.

Figure 9.3 shows the total bytes to transcode for each of the prediction models we propose. As seen from these figures, the Markov model based on clients has more bytes to transcode compared to other models because the trace consists of a higher number of clients as described in Section 8.4 compared to servers or network or client operating systems in the trace.

The results presented in this Section show that for online transcoding with the first segment pre-encoded (1Seg/Rest transcoding), a combined Markov prediction model based on network type and client OS yields the lowest average prediction error



(a) Transcoding Times for 6 second chunk

(b) Transcoding Time for different video segment lengths

Figure 9.4. Transcoding Times for 100 HD videos with different preset options.

rate of 0.3% and also reduces the amount of bytes to transcode to less than 10 Gbps for every instance video requests in our trace.

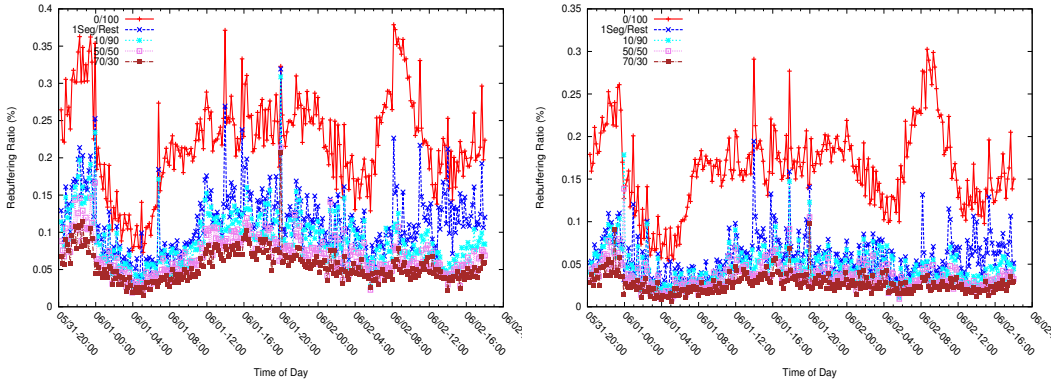
9.2 Impact of Transcoding Policy on Rebuffers

We now show how our Markov prediction model from Section 9.1 can be used to perform online and hybrid transcoding without a significant degradation in the viewing experience of the user. We use the two best prediction models that had the smallest error rates: OS alone and a combination of network type and OS.

9.2.1 Transcoding Time Analysis

To get an accurate picture of the time it takes to transcode a video segment, we measure the time taken to transcode a 6 second chunk of a video into multiple bit rates. We use a sampling of 100 HD videos and take the average transcoding time across those videos. We use this transcode time in the performance analysis of our online transcoding system.

For the transcoding time analysis, we use the FFmpeg utility [16] to divide the videos into specific segment lengths and x264 encoder [37] to encode the video segments into different bit rates. In our analysis, we transcode 100 sample HD videos by dividing them into segments of 6 seconds each. The transcoding was performed



(a) Rebuffer Ratio with OS tuple for prediction.

(b) Rebuffer Ratio with network type-OS tuple for prediction.

Figure 9.5. Rebuffer Ratio analysis for different transcoding approaches and prediction models.

on a ExoGENI cloud [15] instance with 4 Cores and 12 GB RAM. We transcode the 6 second segments of 100 HD videos into bit rates ranging from 200 Kbps to 3000 Kbps, with different encoding speed (preset) options (slow, medium, fast, ultrafast) available in the x264 encoder utility. The different encoding speed options result in different compression efficiencies and qualities of the videos. The default preset option in the x264 encoder is "medium". We also measure the transcoding time taken to encode a video to a fixed bit rate for different segment lengths. In this analysis, we fix the video bit rate at 1000 Kbps and measure the time it takes to transcode 100 HD videos to segment lengths ranging from 2 to 60 seconds.

Figure 9.4(a) shows the average time taken to transcode a 6 second segment of 100 HD videos into different bit rates ranging from 200 to 3000 Kbps at different preset settings. The results show that, with a default x264 encoder preset option of "medium", a 12 GB RAM cloud instance can transcode a 6 second segment in less than 6 seconds. With the increase in transcoding speed (Fast or UltraFast option), the average time to transcode the segments is reduced considerably and with the "slow" preset option, the average time to transcode is larger than the segment length.

Figure 9.4(b) shows the average time taken to transcode 100 HD videos of different segment lengths into 1000 Kbps bit rate. The segment lengths of videos range from

2 to 60 seconds. As seen from Figure 9.4(b), the increase in time taken to transcode a segment is almost linear with the increase in segment length. Yet, irrespective of segment length, the time taken to transcode the segment is less than the segment length. E.g., with the "medium" preset option (default), the average time taken to transcode a video with segment length 20 seconds is less than 19 seconds and it takes about 35 seconds to transcode a 40 seconds segment. Hence, irrespective of the segment length chosen by the content provider, we conclude that online transcoding of segments is feasible and allows the delivery of video on time to avoid rebuffering.

9.2.2 Rebuffering Analysis

We measure the performance of our transcoding architecture in terms of the *rebuffer ratio* which is simply the ratio of the rebuffer time and the duration of the video, where rebuffer time is the amount of time spent in the rebuffer state. For instance, if a video that is played for 50-minutes experiences a 30-second freeze, the rebuffer ratio is 1%. We compute rebuffer time as follows. The number of video segments that could experience a misprediction is simply the number of segments in the video times the prediction error rate. For each misprediction, the video segment must be transcoded and delivered to the client which takes time T_{tot} as shown in Equation 9.1 below.

$$T_{tot} = T_{transcode} + T_{comm}, \tag{9.1}$$

where $T_{transcode}$ is the time to transcode the video segment studied in Section 9.2.1 and T_{comm} is the total time for all the communication that needs to performed. Note that T_{comm} includes the time for the edge server to request the segment from cloud storage, the cloud storage to request and receive the transcoded output from the transcoding cloud, and for the video segment to be sent from the storage cloud to the edge server. Since the communication time T_{comm} is typically in the order of hundreds

of milliseconds compared to the transcoding time that is in the order of seconds, we can approximate T_{tot} by $T_{transcode}$. Thus,

$$\text{Rebuffer Time} = \#\text{segments} \times \text{prediction error rate} \times T_{transcode}.$$

To analyze the performance of our online transcoding architecture, we use the Akamai data set mentioned in Section 8.4. Figures 9.5(a) and 9.5(b) show the results of our rebuffer ratio analysis for different transcoding policies (100% online and hybrid transcoding) and different Markov prediction models (OS and network type-OS). As we saw in Figure 9.2(b), the network type-OS Markov model leads to a slightly lower prediction error compared to the other models. This is emphasized by the results shown in Figure 9.5. As seen from Figure 9.5(a), 100% online transcoding results in the worst performance with an average rebuffer ratio of 0.33%, whereas with the network type-OS Markov prediction model, it reduces to 0.22% as seen in Figure 9.5(b).

However, the hybrid approach of pre-transcoding (offline) the first segment of each video to all the bit rates and online transcoding rest of the video, represented as 1Seg/Rest transcoding in Figures 9.5(a) and 9.5(b) yields lower rebuffer ratio of 0.16% and 0.09% respectively. Also, as seen from both figures, other hybrid transcoding approaches with a large offline transcoding portion yield slightly better performance with the lowest rebuffer ratio of 0.02% with 90/10 transcoding (Not shown in figure) using the Markov prediction model based on network type and client OS.

Based on the results, we suggest that a hybrid transcoding approach with pre-transcoding the very first segment to all quality levels (as requested by the content provider) and online transcoding of the remainder of the video (1Seg/Rest) is the best hybrid transcoding strategy. This strategy reduces the transcoding workload by an order of magnitude (see Figure 8.7), while degrading the performance at the client only slightly compared to other hybrid transcoding approaches.

9.3 Summary

In this chapter we presented prediction models to predict the next bit rate of the video segment likely to be requested by the client. We presented a simple prediction technique and a complex prediction technique based on Markov process. We analyzed the prediction accuracy in terms of error rate for the different prediction models. Our analysis showed that the Markov prediction model based on OS and network type yields lowest error rate than other parameters. Hence, we combined the two parameters and formed a new Markov prediction model based on OS and network type. According to our analysis, Markov prediction model based on both OS and network type yields the lowest error rate of 0.3%, where as Markov prediction model based on only OS yields an error rate of 0.5%. Hence, we conclude that to obtain a lower prediction error rate and reduce the amount of online video transcoding requests Markov model based on both OS and network type is the best prediction model.

We used this prediction model to analyze the performance of our online transcoding policies in terms of rebuffering at the client. We determined the time taken to transcode video segments to different bit rates and to different segment intervals on a cloud instance. We used this time in our rebuffering analysis of online transcoding policies. Our analysis suggests that hybrid transcoding policy with first segment pre-transcoded yields the lowest rebuffering ratio of 0.09%. Also with increase in the percentage of video segments offline transcoded the buffering ratio decreases to about 0.02% with 90/10 transcoding. However, we suggest that hybrid online transcoding with the first segment pre-transcoded is the best transcoding policy as it reduces the transcoding workload by 95% and has only rebuffering ratio of just 0.09%.

CHAPTER 10

CONCLUSION AND FUTURE WORK

10.1 Dissertation Summary

Popularity of cloud services has led them to be used in variety of applications. The Infrastructure-as-a-Service (IaaS) model of the cloud services provides opportunity to run applications which are used infrequently, such as weather forecasting and applications which require high elasticity, such as Video on Demand application. In this dissertation, we looked into the efficiency of the cloud IaaS model for two applications: A real time scientific application of short-term weather forecasting and VoD services.

10.1.1 Scientific Application

In this dissertation, we presented CloudCast, an application for personalized short-term weather forecasting. CloudCast uses instances from commercial and research cloud services to execute a short-term weather forecasting application which is based on a Nowcasting algorithm. We demonstrate the network feasibility of using cloud services for our CloudCast application by performing a series of measurements. Our results show that, serial transmission of data from radars to cloud instances is suitable for our application, while parallel transmission from a large set of radar nodes is a bottleneck for real-time operations of CloudCast. We also infer from our results that, using a small set of radar nodes for parallel transmission to a single cloud instance is suitable for our application in the cloud.

We also compare the compute feasibility of Nowcasting in the cloud with real weather data on various instance types offered by cloud services. We calculate the

computation time and the cost to generate a 15-minute Nowcasts in the cloud. Our results show that computation time for generating 15-minute Nowcasts reduces by $\sim$ 30% if executed on a high-performance instance, but with an almost 300% higher cost than a low-performance instance in Amazon EC2 cloud service. It can be observed from the results that ExoGENI cloud service provides lower computation time to generate 15-minute Nowcasts compared to other cloud services considered. We also performed a live experiment with our CloudCast architecture based on data from a weather radar that is located on our campus. The results from our live measurement show a very high prediction accuracy whereas the delay between data generation at the radar to the delivery of 15-minute Nowcast image to a mobile client is less than 2 minutes on average.

In conclusion, we have shown that commercial and research cloud services are feasible for the execution of our real-time CloudCast application. With this approach, accurate short-term weather forecasts can be provided to mobile users. We believe that CloudCast has the potential to support emergency managers and the general public in severe weather events by promptly providing them with potentially life-saving information.

10.1.2 VoD Application

The rise in popularity of VoD services has been attributed to the popularity of Internet connectivity at homes. There are variety of VoD services which are popular for their content, among those YouTube comes out as the world's widely used user-generated video service. YouTube hosts millions of videos and serves billions of requests every year, which translates to a huge amount of network traffic. The characteristics of YouTube and hence many other VoD services is that it follows a long tail popularity distribution of videos, where only a few videos are popular and hence effectiveness of caches hosting them are minimal.

Also, video streaming in the Internet has moved from traditional fixed bit rate video streaming approach to adaptive bit rate streaming depending on the network connection between client and server. To support ABR streaming multiple copies of the same video has to be created depending on the supported bit rate and client devices. With the long tail popularity distribution of VoD services, creating multiple copies of all the videos before it is requested wastes the resources used for transcoding the videos and also storage space to store the transcoded copies.

In this dissertation, we looked at the feasibility of using the elasticity of cloud services to solve two problems of long tail popularity distribution in VoD services i) Improving the efficiency of the caches hosting the long tail videos of VoD services. ii) Online transcoding of videos into multiple versions based on different bit rate and device characteristics to reduce the transcoding resources and storage space needed.

Improving Efficiency of Caches

In this dissertation, we take advantage of the elasticity of cloud services to manage the long tail popularity effect of YouTube videos by hosting the videos based on their popularity. We use a new related list reordering approach to increase the hit rate of the popular cache and reduce the amount of requests served from the long tail. Thus, using our approach, we reduce the energy foot print, increase the hit rate of popular cache and hence the efficiency of caching videos from the long tail popularity of YouTube.

Recent works have shown that YouTube viewers select their next video request from the related list provided by YouTube on the watch page of the current video with high probability. In this dissertation, we took advantage of this user behavior to modify the related list provided by YouTube based on the content already present in the cache. In our approach, the related list is modified only in the order of appearance on the user's webpage, not the content itself. During the modification, the related videos present in our cache, which are the videos already requested by the users, are

moved to the top of the list and subsequently the related videos not cached are moved down the order. By analyzing a campus network trace filtered for YouTube traffic, we find that users generally request videos from the top half of the related list and hence moving the related videos already present in the cache makes the approach more advantageous.

We performed a user study with actual YouTube users provided with a modified related list and showed that YouTube users select from the top half of the related list even after provided with a modified related list. We analyzed our reordering approach in a simulation based study on a network trace captured on a campus gateway. We define two different approaches of video selection from the re-ordered related list. Content Centric selection, where the user selects the same video as he would have originally selected and Position Centric selection, where the user selects a video from the same position on the related list before reordering, which might change the original content. From our simulation study, we find that Position Centric selection of the reordered related list yields a better hit rate (2 to 5 times) than Content Centric selection, which does not take advantage of the content in the cache.

From the analysis of the caching techniques used by YouTube and our related list reordering approach, we conclude that reordering the related list presented to the user based on the content already requested by the users, yield a better hit rate of the caches, thereby reducing the bandwidth consumption by multimedia requests and hence the latency in content delivery.

Further, we presented our CloudTube architecture of hosting YouTube videos on cloud instances in 3-tier cache hierarchy. We host the videos based on the global popularity of the videos. Based on the user behavior of watching video sessions on YouTube, we found that 58% of the video sessions are not completed and 25% of the sessions are ended within the first 20% of the video session length. Based on the user watching pattern results, we placed the first chunk (60 seconds) of each video in our

trace in the high popularity cloud instance, along with the video placement strategy based on global popularity. Also, we used the related list reordering approach to move the related videos which are already present in our cloud instances to high popularity cloud instance if its not already present. We evaluate this video hosting strategy and found that the hit rate for the high popular cloud instance increases by 90% and the uptime for medium and low popularity cloud instance decreases by almost 20%, which in turn reduces the power required to serve the contents by almost 20%.

Hence, we suggest that placing the first chunk of all videos and highly popular videos closer to the client and the less popular long tail videos on secondary or tertiary cache cloud instances yield higher hit rate and reduces the uptime and power required to serve less popular videos, which constitute the long tail in VoD services such as YouTube.

Online Transcoding

In this dissertation, we look at the feasibility of online transcoding in the cloud to reduce the transcoding resources and storage space required to support ABR streaming and maintain the performance at the client. We proposed several online and hybrid policies that transcode video segments in a timely manner such that a segment is transcoded only to those bit rates that are actually requested by the user. To establish the feasibility of such transcoding, we first showed that the bit rate of the next video segment requested by a user can be predicted ahead of time with an accuracy of 99.7% using a Markov prediction model. This allows our online and hybrid transcoding policies to complete transcoding the required segment ahead of when it is needed by the user, thus reducing the possibility of freezes in the video playback. To derive our results, we collected and analyzed a large amount of request traces from one of the world's largest video CDNs consisting of over 200 thousand unique users watching 5 million videos over a period of three days. From our analysis we conclude that an online transcoding scheme with the first segment pre-encoded along

with a Markov prediction model based on network type and client OS can reduce transcoding resources by over 95% without a major impact on the users' quality of experience.

10.2 Future Work

In this section we present some ideas on how work presented in this dissertation can be extended in the future.

- **Green Transcoding.** Previously we have looked at running multimedia cache cluster powered by renewable energy sources with intermittent power supply [92]. Similarly, the online transcoding work presented in this dissertation can be extended to analyze different hybrid transcoding polices when transcoding needs to be performed in a data center powered by renewable energy. Depending on the available power, a trade-off between the amount of offline bytes to transcode and the preset to use during transcoding can be determined. To make green transcoding feasible, we need to determine if compromising compression efficiency yields any power savings, so a policy based on the transcoder presets can be used to use the available power to transcode the video segments in time.

- **Popularity based transcoding policy.** In this dissertation, we presented a hybrid transcoding policy where the offline transcoding is performed on parts of each video to all the bit rates supported by content provider. However, other hybrid transcoding policies could be investigated where the popular videos are offline transcoded to all the bit rates supported and the videos which are requested rarely (long tail) are transcoded on-demand. The two hybrid transcoding policies can be compared based on the trade-off each policy offers in terms of transcoding workload reduction and performance in terms of buffering ratio at the client.

- **Better prediction models.** In this dissertation, we presented prediction models to predict the next bit rate likely to be requested by the client. In the future, we want to investigate if the prediction models can be used to generate client requests. Can we influence the client to send video requests depending on the transcoded segments present in the cache or edge servers? Current adaptive video players simply use individual bandwidth measurement information and the current buffer level to determine at which quality the next segment should be requested. We want to investigate if prediction models can be used as one of the factors client players considers before they send the video request.

CHAPTER 11

LIST OF PUBLICATIONS

- Dilip Kumar Krishnappa, Samamon Khemmarat, Lixin Gao, Michael Zink, "On the feasibility of prefetching and caching for online TV services: a measurement study on Hulu, *12th International Conference on Passive and Active Measurements, Atlanta, March 2011.*

- Dilip Kumar Krishnappa, Samamon Khemmarat, Michael Zink, "Planet YouTube: Global, Measurement-based Performance Analysis of Viewers Experience Watching User Generated Videos, *6th IEEE LCN Workshop on Network Measurements, Bonn, Germany, 2011.*

- Samamon Khemmarat, Renjie Zhou, Dilip Kumar Krishnappa, Lixin Gao, Michael Zink, "Watching User Generated Videos with Prefetching, *Elsevier Journal on Signal Processing: Image Communication, Nov 2011.*

- Dilip Kumar Krishnappa, Eric Lyons, David Irwin, Michael Zink, "Network Capabilities of Cloud Services for Real Time Scientific Application", *37th IEEE LCN Conference, Tampa, Florida, 2012.*

- Dilip Kumar Krishnappa, David Irwin, Eric Lyons, Michael Zink, "CloudCast: Cloud Computing for Short-term Mobile Weather Forecasts", *31th IEEE IPCCC Conference, December, 2012.*

- Dilip Kumar Krishnappa, Divyashri Bhat, Michael Zink, "DASHing YouTube: An Analysis of Using DASH in YouTube Video Service", *38th IEEE LCN Conference, Sydney, Australia, 2013.*

- Dilip Kumar Krishnappa, Michael Zink, Carsten Gridwoz, Paal Halvorsen, "Cache-centric Video Recommendation: An Approach to Improve the Efficiency of YouTube Caches", *4th ACM MMSys Conference, February, 2013.*

- Dilip Kumar Krishnappa, Michael Zink, Carsten Gridwoz, "What should you Cache? A Global Analysis on YouTube Related Video Caching", *23rd ACM NOSSDAV Workshop, February, 2013.*

- Navin Sharma, Dilip Kumar Krishnappa, David Irwin, Michael Zink, Prashanth Shenoy, "GreenCache: Augmenting Off-the-Grid Cellular Towers with Multimedia Caches", *4th ACM MMSys Conference, February, 2013.*

- Dilip Kumar Krishnappa, David Irwin, Eric Lyons, Michael Zink, "CloudCast: Cloud Computing for Short-Term Weather Forecasts", *CiSE Cloud Computing Article, August, 2013.*

- Dilip Kumar Krishnappa, Michael Zink, Carsten Gridwoz, Paal Halvorsen, "Cache-centric Video Recommendation: An Approach to Improve the Efficiency of YouTube Caches", *ACM Transactions on Multimedia Computing, Communications, and Applications (TOMM), Under Review.*

- Dilip Kumar Krishnappa, Michael Zink, Ramesh Sitaraman, "Optimizing the Video Transcoding Workflow in Content Delivery Networks", *ACM MMSys Conference 2015, Under Review.*

- Dilip Kumar Krishnappa, Cong Wang, Divyashri Bhat, Michael Zink, "DACHE: Cache Support for Adaptive Video Streaming", *ACM MMSys Conference 2015, Under Review.*

BIBLIOGRAPHY

[1] AccuWeather. http://www.accuweather.com/downloads.asp.

[2] Adobe HTTP Dynamic Streaming. http://www.adobe.com/products/hds-dynamic-streaming.html. Accessed: September, 25, 2014.

[3] Akamai Media Delivery Solution. http://www.akamai.com/mediadelivery. Accessed: October, 3, 2014.

[4] Akamai Transcoding. `http://www.akamai.co.jp/enja/dl/brochures/sola_vision_transcoding_brief.pdf`. Accessed: September, 25, 2014.

[5] Amazon CloudFront. `http://aws.amazon.com/cloudfront/`.

[6] Amazon Elastic Transcoder. `http://aws.amazon.com/elastictranscoder/`. Accessed: September, 25, 2014.

[7] Amazon Simple Storage Service. `http://aws.amazon.com/s3/`. Accessed: September, 25, 2014.

[8] Amazon's Elastic Compute Cloud. http://aws.amazon.com/ec2/.

[9] Apple HTTP Live Streaming. https://developer.apple.com/resources/http-streaming/. Accessed: September, 25, 2014.

[10] Azure CDN. `http://www.windowsazure.com/en-us/`.

[11] Brightcove Zencoder. `https://zencoder.com/en/`. Accessed: September, 25, 2014.

[12] Clouds Can't Move Fast Enough for Weather Service. http://www.hpcinthecloud.com/blogs/Clouds-Cant-Move-Fast-Enough-for-Weather-Service-100353244.html.

[13] Encoder Cloud. http://www.encodercloud.com/. Accessed: September, 25, 2014.

[14] Endace DAG Network Monitoring Interface. http://www.endace.com/.

[15] ExoGENI. http://wiki.exogeni.net.

[16] FFmpeg. https://www.ffmpeg.org/. Accessed: September, 25, 2014.

[17] Filter Proxy. http://filterproxy.sourceforge.net/.

[18] Global Internet Phenomena Report. https://www.sandvine.com/downloads/general/global-internet-phenomena/2014/1h-2014-global-internet-phenomena-report.pdf. Accessed: September, 25, 2014.

[19] Google Transparency Report. http://www.google.com/transparencyreport/traffic/?r=US&l=YOUTUBE&csd=1345235172273&ced=1346728973645/.

[20] Internet Traffic Forecast. http://www.cisco.com/en/US/solutions/collateral/ns341/ns525/ns537/ns705/ns827/white_paper_c11-481360.pdf.

[21] Internet2 ION. https://geni-orca.renci.org/trac/wiki/flukes.

[22] Iperf. http://sourceforge.net/projects/iperf/.

[23] MetService Rejects Cloud. http://computerworld.co.nz/news.nsf/technology/metservice-rejects-cloud.

[24] Microsoft Smooth Streaming. http://www.iis.net/downloads/microsoft/smooth-streaming. Accessed: September, 25, 2014.

[25] Muffin. http://muffin.doit.org/.

[26] My-Cast. http://www.digitalcyclone.com/products/mobile-my-cast/.

[27] Netflix. http://www.netflixprize.com/.

[28] NEuca. https://geni-orca.renci.org/trac/wiki/NEuca-overview.

[29] NOAA Supercomputer Solicitation. https://www.fbo.gov/utils/view?id=cc031d6f3b30b8e6f872e4e53136c851.

[30] OpenStack. http://www.openstack.org.

[31] Opera Mobile. http://www.opera.com/mobile/.

[32] Rackspace Cloud. http://www.rackspace.com.

[33] Sface. http://svn.planet-lab.org/wiki/SfaceGuide.

[34] tcpdump. http://www.tcpdump.org/.

[35] The Weather Channel App. http://www.weather.com/mobile/.

[36] Web Cleaner. http://webcleaner.sourceforge.net/).

[37] x264 Encoder. http://www.videolan.org/developers/x264.html. Accessed: September, 25, 2014.

[38] YouTube API. https://developers.google.com/youtube/.

[39] YouTube Keynote at MMSys 2012. https://docs.google.com/presentation/pub?id=1bMLitOe_fxARBbgcu1v1xaJj89hb_JGXYse17Xvgwro&start=false&loop=false&delayms=3000#slide=id.g47538e9_2_210.

[40] YouTube Press Statistics. http://www.youtube.com/t/press_statistics/.

[41] YouTube Video Formats. http://en.wikipedia.org/wiki/YouTube#Quality_and_codecs. Accessed: September, 25, 2014.

[42] Data centers are the new polluters. http://www.computerworld.com/article/2598562/data-center/data-centers-are-the-new-polluters.html, Accessed November 2014.

[43] V. Adhikari, S. Jain, Y. Chen, and Z.-L. Zhang. Vivisecting YouTube: An Active Measurement Study. In *INFOCOM*, March 2012.

[44] V. Adhikari, S. Jain, and Z.-L. Zhang. Where Do You "Tube"? Uncovering YouTube Server Selection Strategy. In *ICCCN*, August 2011.

[45] A. C. Bavier, M. Yuen, J. Blaine, R. McGeer, A. A. Young, Y. Coady, C. Matthews, C. Pearson, A. Snoeren, and J. Mambretti. Transcloud - Design Considerations for a High-performance Cloud Architecture Across Multiple Administrative Domains. In *CLOSER*, May 2011.

[46] O. Celma. *Music Recommendation and Discovery: The Long Tail, Long Fail, and Long Play in the Digital Music Space*. Springer, 2010.

[47] M. Cha, H. Kwak, P. Rodriguez, Y. Ahn, and S. Moon. I Tube, You Tube, Everybody Tubes: Analyzing the World's Largest User Generated Content Video System. In *IMC*, October 2007.

[48] J. Chakareski. Browsing Catalogue Graphs: Content Caching Supercharged!! In *ICIP*, September 2011.

[49] X. Cheng and J. Liu. NetTube: Exploring Social Networks for Peer-to-Peer Short Video Sharing. In *INFOCOM*, April 2009.

[50] X. Cheng, J. Liu, and H. Wang. Accelerating YouTube with Video Correlation. In *WSM*, November 2009.

[51] B. Chun, D. Culler, T. Roscoe, A. Bavier, L. Peterson, M. Wawrzoniak, and M. Bowman. Planetlab: an overlay testbed for broad-coverage services. *SIGCOMM Comput. Commun. Rev.*, 33:3–12, July 2003.

[52] W. Conover. *Practical Nonparametric Statistics*. Wiley, 3rd. edition, 1999.

[53] E. Deelman, G. Singh, M. Livny, B. Berriman, and J. Good. The Cost of Doing Science on the Cloud: The Montage Example. In *SC*, November 2008.

[54] A. Elberse. Should You Invest in the Long Tail? *Harvard business review*, 86(7/8):88, 2008.

[55] J. Erman, A. Gerber, K. K. Ramadrishnan, S. Sen, and O. Spatscheck. Over the Top Video: The Gorilla in Cellular Networks. In *Proceedings of the 2011 ACM SIGCOMM conference on Internet measurement conference*, IMC '11, pages 127–136, New York, NY, USA, 2011. ACM.

[56] C. Evangelinos and C. N. Hill. Cloud Computing for Parallel Scientific HPC Applications: Feasibility of Running Coupled Atmosphere-Ocean Climate Models on Amazon's EC2. In *CCA*, October 2008.

[57] A. Finamore, M. Mellia, M. M. Munafò, R. Torres, and S. G. Rao. Youtube Everywhere: Impact of Device and Infrastructure Synergies on User Experience. In *IMC*, November 2011.

[58] S. Hazelhurst. Scientific Computing Using Virtual High-performance Computing: A Case Study Using the Amazon Elastic Computing Cloud. In *SAICSIT*, October 2008.

[59] D. Irwin, P. Shenoy, E. Cecchet, and M. Zink. Resource Management in Data-Intensive Clouds: Opportunities and Challenges (invited paper). In *Workshop on Local and Metropolitan Area Networks*, May 2010.

[60] K. Jackson, L. Ramakrishnan, K. Muriki, S. Canon, S. Cholia, J. Shalf, H. Wasserman, and N. Wright. Performance Analysis of High Performance Computing Applications on the Amazon Web Services Cloud. In *CloudCom*, December 2010.

[61] H. Jin, S. Ibrahim, T. Bell, W. Gao, D. Huang, and S. Wu. Cloud Types and Services. In *Handbook of Cloud Computing*. 2010.

[62] G. Juve, E. Deelman, K. Vahi, G. Mehta, B. Berriman, B. Berman, and P. Maechling. Scientific Workflow Applications on Amazon EC2. In *E-Science*, December 2009.

[63] K. Keahey, R. Figueiredo, J. Fortes, T. Freeman, and M. Tsugawa. Science Clouds: Early Experiences in Cloud Computing for Scientific Applications. In *CCA*, October 2008.

[64] K. Keahey, T. Freeman, J. Lauret, and D. Olson. Virtual Workspaces for Scientific Applications. *Journal of Physics: Conference Series*, 2007.

[65] S. Khemmarat, R. Zhou, L. Gao, and M. Zink. Watching User Generated Videos with Prefetching. In *MMSys*, February 2011.

[66] H. Kllapi, E. Sitaridi, M. M. Tsangaris, and Y. Ioannidis. Schedule Optimization for Data Processing Flows on the Cloud. In *Proceedings of the 2011 ACM SIGMOD International Conference on Management of Data*, 2011.

[67] D. E. Knuth. *The Art of Computer Programming, volume 3: (2nd ed.) Sorting and Searching*. Addison Wesley Longman Publishing Co., Inc., Redwood City, CA, USA, 1998.

[68] S. Ko, S. Park, and H. Han. Design Analysis for Real-time Video Transcoding on Cloud Systems. In *Proceedings of the 28th Annual ACM Symposium on Applied Computing*, 2013.

[69] S. S. Krishnan and R. K. Sitaraman. Video Stream Quality Impacts Viewer Behavior: Inferring Causality using Quasi-Experimental Designs. In *IMC*, November 2012.

[70] D. K. Krishnappa, D. Bhat, and M. Zink. Dashing YouTube: An Analysis of Using DASH in YouTune Videa Service. In *IEEE Proceedings of LCN*, October 2013.

[71] D. K. Krishnappa, S. Khemmarat, L. Gao, and M. Zink. On the Feasibility of Prefetching and Caching for Online TV Services: A Measurement Study on Hulu. In *Passive and Active Measurement*, March 2011.

[72] S. Lederer, C. Müller, and C. Timmerer. Dynamic Adaptive Streaming over HTTP Dataset. In *MMSys*, February 2012.

[73] M. Levy and K. Bosteels. Music recommendation and the long tail. *Copyright Information*, page 55.

[74] A. Li, X. Yang, S. Kandula, and M. Zhang. CloudCmp: Shopping for a Cloud Made Easy. In *HotCloud*, June 2010.

[75] H. Li, L. Zhong, J. Liu, B. Li, and K. Xu. Cost-effective Partial Migration of VoD Services to Content Clouds. In *CLOUD*, 2011.

[76] J. Li, M. Humphrey, D. Agarwal, K. Jackson, C. van Ingen, and Y. Ryu. eScience in the Cloud: A MODIS Satellite Data Reprojection and Reduction Pipeline in the Windows Azure Platform. In *IPDPS*, April 2010.

[77] P. Li, B. Veeravalli, and A. Kassim. Design and Implementation of Parallel Video Encoding Strategies using Divisible Load Analysis. *Circuits and Systems for Video Technology, IEEE Transactions on*, 15(9):1098–1112, Sept 2005.

[78] Z. Li, Y. Huang, G. Liu, F. Wang, Z.-L. Zhang, and Y. Dai. ICloud Transcoder: Bridging the Format and Resolution Gap between Internet Videos and Mobile Devices. In *Proceedings of the 22nd international workshop on Network and Operating System Support for Digital Audio and Video*, 2012.

[79] LoggerPlugin. Youtube video logger chrome plugin, 2014.

[80] H. Ma, B. Seo, and R. Zimmermann. Dynamic Scheduling on Video Transcoding for MPEG DASH in the Cloud Environment. In *Proceedings of the 5th ACM Multimedia Systems Conference*, 2014.

[81] D. McLaughlin, D. Pepyne, V.Chandrasekar, B. Philips, J. Kurose, and M. Z. et al. Short-Wavelenth Technology and the Potential for Distributed Networks of Small Radar Systems. *Bulletin of the American Meteorological Society (BAMS)*, 90(12):1797–1817, December 2009.

[82] D. Nurmi, R. Wolski, C. Grzegorczyk, G. Obertelli, S. Soman, L. Youseff, and D. Zagorodnov. The Eucalyptus Open-Source Cloud-Computing System. In *CCGRID*, May 2009.

[83] E. Nygren, R. Sitaraman, and J. Sun. The Akamai Network: A platform for high-performance Internet applications. *ACM SIGOPS Operating Systems Review*, 44(3):2–19, 2010.

[84] X. Qiu, H. Li, C. Wu, Z. Li, and F. Lau. Cost-Minimizing Dynamic Migration of Content Distribution Services into Hybrid Clouds. In *INFOCOM*, 2012.

[85] X. Qiu, H. Li, C. Wu, Z. Li, and F. C. M. Lau. Dynamic Scaling of VoD Services into Hybrid Clouds with Cost Minimization and QoS Guarantee. In *Packet Video Workshop*, 2012.

[86] L. Ramakrishnan, K. R. Jackson, S. Canon, S. Cholia, and J. Shalf. Defining Future Platform Requirements for e-Science Clouds. In *SoCC*, October 2010.

[87] J. Ratkiewicz, S. Fortunato, A. Flammini, F. Menczer, and A. Vespignani. Characterizing and Modeling the Dynamics of Online Popularity. *Physical review letters*, 105(15), 2010.

[88] J. J. Rehr, J. P. Gardner, M. Prange, L. Svec, and F. Vila. Scientific Computing in the Cloud. *ArXiv e-prints*, Dec. 2009.

[89] E. Ruzanski, V. Chandrasekar, and Y. Wang. The CASA Nowcasting System. *Journal of Atmospheric and Oceanic Technology*, May 2011.

[90] E. Ruzanski, Y. Wang, and V. Chandrasekar. Development of a Real-time Dynamic and Adaptive Nowcasting System. In *Interactive Information and Processing Systems for Meteorology, Oceanography, and Hydrology*, January 2009.

[91] N. Sharma, S. Barker, D. Irwin, and P. Shenoy. Blink: Managing Server Clusters on Intermittent Power. In *ASPLOS*, March 2011.

[92] N. Sharma, D. K. Krishnappa, D. Irwin, M. Zink, and P. Shenoy. GreenCache: Augmenting off-the-grid Cellular Towers with Multimedia Caches. In *Proceedings of the 4th ACM Multimedia Systems Conference*, 2013.

[93] I. Shin and K. Koh. Hybrid Transcoding for QoS Adaptive Video-on-Demand Services. *Consumer Electronics, IEEE Transactions on*, 50(2):732–736, 2004.

[94] J. A. Stankovic, M. Spuri, K. Ramamritham, and G. C. Buttazzo. Introduction. In *Deadline Scheduling for Real-Time Systems*, pages 1–11. Springer, 1998.

[95] R. Steinmetz and K. Nahrstedt. *Multimedia Systems*. X. media. publishing. Springer-Verlag, 2004.

[96] T. Stockhammer. Dynamic Adaptive Streaming over HTTP –: Standards and Design Principles. In *MMSys*, February 2011.

[97] C. Tucker and J. Zhang. Long Tail or Steep Tail? A Field Investigation into How Online Popularity Information Affects the Distribution of Customer Choices. 2007.

[98] G. Wang and T. Ng. The Impact of Virtualization on Network Performance of Amazon EC2 Data Center. In *INFOCOM*, March 2010.

[99] L. Wang, J. Tao, M. Kunze, A. Castellanos, D. Kramer, and W. Karl. Scientific Cloud Computing: Early Definition and Experience. In *HPCC*, September 2008.

[100] Z. Wang, L. Sun, C. Wu, W. Zhu, and S. Yang. Joint Online Transcoding and Geo-distributed Delivery for Dynamic Adaptive Streaming. In *INFOCOM*, 2014.

[101] D. Wilks. *Statistical Methods in the Atmospheric Sciences*. Elsevier, 2nd edition, 2005.

[102] Y. Wu, C. Wu, B. Li, X. Qiu, and F. C. M. Lau. CloudMedia: When Cloud on Demand Meets Video on Demand. In *ICDCS*, 2011.

[103] L. Xiaowei, C. Yi, and X. Yuan. Towards an Automatic Parameter-Tuning Framework for Cost Optimization on Video Encoding Cloud. *International Journal of Digital Multimedia Broadcasting*, 2012(935724):11, Sept 2012.

[104] M. Yuen. GENI in the Cloud. Master's thesis, University of Victoria, 2010.

[105] R. Zhou, S. Khemmarat, and L. Gao. The Impact of YouTube Recommendation System on Video Views. In *IMC*, November 2010.

[106] R. Zhou, S. Khemmarat, L. Gao, and H. Wang. Boosting Video Popularity through Recommendation Systems. In *DBSocial*, June 2011.

[107] M. Zink, E. Lyons, D. Westbrook, J. Kurose, and D. Pepyne. Closed-loop Architecture for Distributed Collaborative Adaptive Sensing: Meteorogolical Command & Control. *IJSNET*, February 2010.

[108] M. Zink, K. Suh, Y. Gu, and J. Kurose. Watch Global, Cache Local: YouTube Network Traffic at a Campus Network: Measurements and Implications. In *MMCN*, 2008.

[109] M. Zink, K. Suh, Y. Gu, and J. Kurose. Characteristics of YouTube Network Traffic at a Campus Network Measurements, Models, and Implications. *Computer Networks*, 2009.

[110] M. Zink, K. Suh, Yu, and J. Kurose. Characteristics of YouTube Network Traffic at a Campus Network - Measurements, Models, and Implications. *Elsevier Computer Networks*, 2009.