GEO-TAGGED VIDEO MANAGEMENT: STORAGE, QUERIES AND STREAMING

HE MA

(B.Sc., Northeastern University, CHINA)

A THESIS SUBMITTED

FOR THE DEGREE OF DOCTOR OF PHILOSOPHY

DEPARTMENT OF COMPUTER SCIENCE

SCHOOL OF COMPUTING

NATIONAL UNIVERSITY OF SINGAPORE

2013

DECLARATION

I hereby declare that this thesis is my original work and it has been written by me in its entirety. I have duly acknowledged all the sources of information which have been used in the thesis.

This thesis has also not been submitted for any degree in any university previously.

Signature: He MA **Date:** 30 Dec. 2013

ACKNOWLEDGEMENTS

First and foremost I would like to express my sincerest gratitude to my supervisor, Prof. Roger Zimmermann, for his persistent guidance and support throughout the past years. Without his brilliant inspiration to my research problems, his kindly encouragement and help when I encountered difficulties, and his great patience while helping with writing and polishing all my papers, as well as this dissertation, this thesis would not have been completed.

I am grateful to my collaborators Dr. Sakire Arslan Ay, Dr. Seon Ho Kim, Dr. Beomjoo Seo, Dr. Yu-Ling Hsueh and Dr. Junchang Xin for their great contribution to my research works. Their valuable advice, enthusiastic guidance, and great support makes this thesis possible.

I also want to show my gratitude to the committee members of my dissertation, Prof. Mohan S. Kankanhalli and Prof. Stéphane Bressan for their patience in reading and helpful, insightful comments on this dissertation.

I thank all my group members and labmates in NUS: Tian Gan, Jia Hao, Ke Liang, Haiyang Ma, Prabhu Natarajan, Zhijie Shen, Guanfeng Wang, Xiangyu Wang, Yuhui Wang, Xiaohong Xiang, Yangyang Xiang, Karthik Yadati, Yifang Yin, Ying Zhang, for their support to my research works and accompany. I would like to thank my friends in Singapore. Without them, the life would have not been colorful and wonderful. A special thank also goes to Mrs. Adele Chen Zimmermann, who has provided me plenty of advice and encouragement when I encountered difficulties in my life.

Last but not the least, I would like to express my deepest appreciation to my parents, for their kindly understanding, unreserved support, and unselfish love.

Contents

Summary

Geographic Information System (GIS) applications now increasingly make use of geo-located multimedia data such as images and videos. Furthermore, the widespread use of smartphones (and increasingly tablets) and the rapid improvement of hardware has enabled the acquisition of high-definition user-generated videos that are annotated with geo-properties. The sensor meta-data (*e.g.,* GPS and digital compass values), which associate a continuous stream of location and viewing direction information with the collected videos, are considerably smaller in size than the visual content and are helpful in effectively and efficiently manage and search through large repositories of videos.

In this thesis, the properties of these meta-data are studied and utilized them to manage geo-tagged videos. The first part of my work centers on building an index structure for these meta-data so that the videos can be quickly accessed. A multi-level grid-based index structure is proposed and a number of related query types, including typical spatial queries and queries based on a bounded radius and viewing direction restrictions, are introduced. These two criteria are important in many video applications and we demonstrate the importance with real-world datasets. Moreover, experimental results on a large-scale synthetic dataset show that my approach can provide significant speed improvements of at least 30%, considering a mix of queries, compared with a multi-dimensional R-tree implementation. However, a major practical issue is the noisy nature of such sensor data. For example, due to sensor data inaccuracies the visual coverage described by the meta-data may not exactly match the actual video scene, which leads to imprecise search results and positional disagreements on map overlays. Obstructions between the camera and its captured objects make these situations worse. Therefore, robust error-tolerance is an essential feature of any geo-tagged video search application. To this end I introduce a modeling and indexing approach for uncertain geo-tagged videos as my second work. An uncertainty model for video frames and segments is constructed. Since

the frame-by-frame uncertainty model involves high computational complexity, we then propose an approximate modeling method based on a video segmentation algorithm which eliminates costly overlap calculations between the query region and individual frames. Finally, the performance of the proposed method is tested with both a real-world and a large-scale synthetic dataset. Experimental results show that the proposed method achieves high recall and good scalability and allows for the efficient querying of noisy sensor data. The proposed approach also returns confidence probabilities with the results which can then be beneficially used in upstream GIS applications. My third work is the design of a dynamic scheduling algorithm for video transcoding in the context of Dynamic Adaptive Streaming over HTTP (referred as DASH) in a cloud environment. In order to support live or near-live streaming of media content and to provide a satisfactory user experience, the overall video transcoding completion time should be minimized. We first model the estimation of the video transcoding time (referred to as VTT) with respect to the video duration based on statistics and probability theory. The scheduler keeps monitoring the speed of each processor by comparing the estimated VTT and measured VTT. The scheduler then distributes jobs to free processors when they are not urgent, but to the fastest processors if video viewing requests are pending. It can dynamically optimize the video transcoding mode when the number of processors is insufficient to support all video viewing requests. The experimental results show that the proposed scheduler can support near-live streaming while balancing the workload and providing smooth and seamless playback. Finally, the sensor meta-data associated with videos can be used in displaying the geo-properties of videos to end-users.

List of Tables

List of Figures

Chapter 1

Introduction

1.1 Background and Motivation

Recent developments in video recording technology have enabled user-generated video (UGV) production on a daily basis. For example, smartphones (and increasingly tablets), which are carried by users all the time, have become extremely popular for capturing and sharing online videos due to their convenience, enhanced quality of images and wireless connectivity. YouTube [YouTube 2013] has indicated that by the end of 2013, over six billion hours of videos were watched each month and 100 hours of video were uploaded every minute. 40% of global YouTube views come from mobile devices. According to another study from Cisco Systems, Inc. [Cisco 2014], the overall mobile data traffic reached 1.5 Exabytes per month at the end of 2013, 69.1% of which was by mobile video. It is forecast that mobile video traffic will grow at a Compound Annual Growth Rate (CAGR) of 61 percent between 2013 and 2018 and reach 15.9 Exabytes per month by 2018. The statistics of the mobile data traffic in 2013 and the forecast for the following five years are shown in Figure 1.1.

The multimedia community nowadays has increasing interest in GIS applications, *e.g.,* Flickr[1] and Panoramio[2] allow users to upload images and videos with attached geo-locations of the cameras. Furthermore, embedded sensors (*e.g.,* GPS and compass units) have been cost-efficiently deployed on mobile devices. Consequently, some very useful meta-data of videos, especially geographical properties of

[1] http://www.flickr.com/
[2] http://www.panoramio.com/

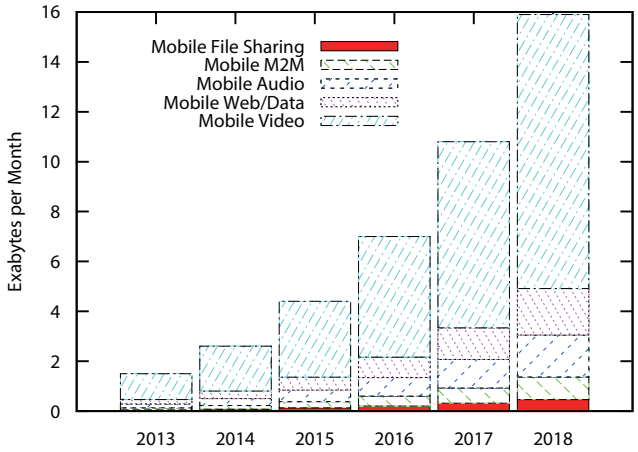


Figure 1.1: Statistics from Cisco Systems, Inc., of the mobile data traffic in 2013 and the prediction by 2018 [Cisco 2014].

video scenes, can easily be collected during video capturing. This association of video scenes and their geographic meta-data has led to interesting research topics in the multimedia community. For instance, the recorded sensor meta-data can be utilized to aid in modeling, indexing and searching of geo-tagged videos at the high semantic level preferred by humans. Figure 1.2 illustrates the overall architecture of our geo-tagged video management system. Mobile users can collect the associated sensor meta-data during video recording with special recording apps and then upload the videos as well as the meta-data to a remote hosting system through a wireless network. The remote hosting system (*e.g.,* the system can be hosted on a single server or in a cloud hosting environment), provides different services for managing geo-tagged videos and hosts a web interface for end-users to access videos by posing queries through a map application. When streaming to mobile clients, the Dynamic Adaptive Streaming over HTTP (DASH) standard, which supports adaptive streaming and overcomes the disadvantages of progressive download, is used for high quality streaming of media content based on the network conditions.

Some prior research approaches [Arslan Ay 2008, Liu 2005] have modeled the coverage region (*i.e.,* field of view) of video frames as a pie-shaped geometric area described by the sensor meta-data, such as the camera location, viewing direction and visible distance. This approach treats the visual content of a video as a series

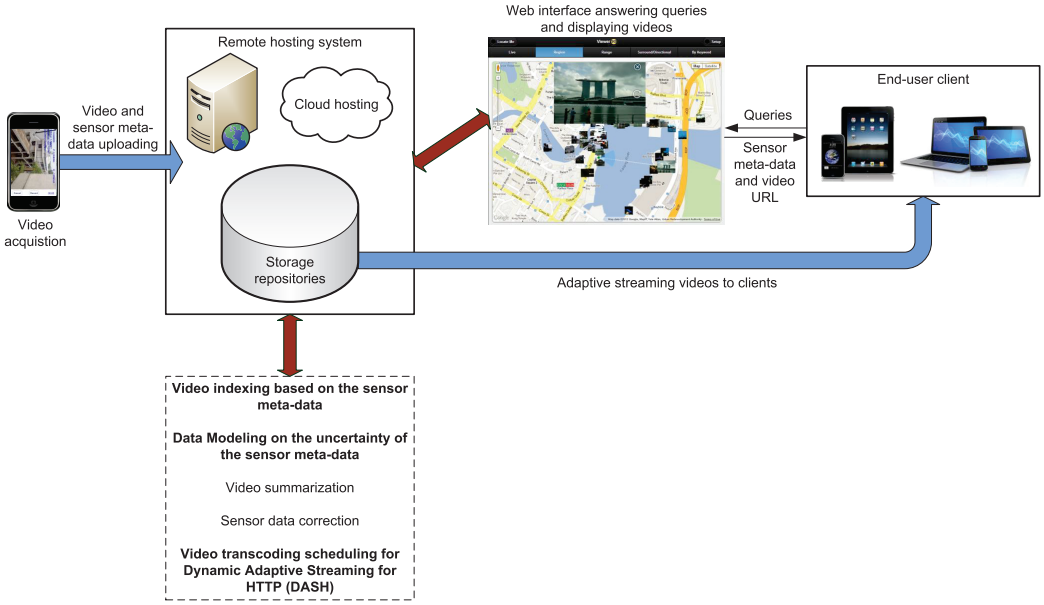


Figure 1.2: The overview of the geo-tagged video management system. The topics discussed in this thesis are shown in bold.

of spatial objects. Compared to visual content, the meta-data occupies much less space, which makes searching through large quantities of videos practical. Consequently, the challenging video search problem is transformed into a known spatial data selection problem. The objective is then to index the spatial objects and to search videos based on their geographic properties. Existing spatial data indexing methods always treat the spatial data as a point or a small volume object. However, the geo-coverage region of a video frame is large and hence these methods are not appropriate. Moreover, the overlap region among consecutive frames is also large, which drives indexing video segments instead of single frames become a challenging problem, since too much overlap might deteriorate the query performance due to duplicate calculation. To this end, we have implemented a multi-level grid-based index structure to support quick searching among large-scale geo-tagged videos, which is detailed in Chapter 3.

Utilizing the geo-properties of visual content can improve the effectiveness and efficiency of searching through geo-tagged videos. However, an important practical

aspect is often neglected in existing solutions: the noisy nature of sensor data. It has long been known (and many users have had first-hand experience) that sensors, especially on consumer-grade electronics, produce sometimes inaccurate and fluctuating values. The surrounding environment can exacerbate this problem. For example, tall buildings and narrow passageways in urban areas can lead to very difficult conditions for obtaining accurate GPS positions [Martin 2006]. Typically alignment errors, non-orthogonal errors and magnetic deviations can affect the digital compass heading accuracy. Moreover, obstructions (such as buildings, people or vehicles passing by) in front of a camera may in parts of a scene result in the recording of objects far away (*e.g.,* 1,500 m) while only close objects in other parts within the same frame. The latter effects influence the captured video scene, but not the measured sensor data. The above described issues lead to a mismatch between the viewable scenes of the visual content and the area represented by the sensor data. Rather than trying to completely avoid or correct such issues (which may be difficult or impossible), a well-known approach in the information management community is to design methods that can handle *uncertain data*. The uncertainty region of the FOV model is affected by multiple factors (*i.e.,* sensor accuracy, obstacles) and this makes the uncertain modeling of FOV complex. As stated in the previous paragraph, current research only deal with small objects but not large geo-coverage region. Considering the large uncertainty region of the FOV and the overlap among FOVs, the probability calculation is computational expensive. Therefore, modeling the uncertainty region of the FOV and hence video segment, with light-weight calculation is a challenging problem. In order to overcome the shortcomings of the existing works, modeling the uncertainty of the sensor meta-data are introduced and the details are presented in Chapter 4.

Building an index structure and handling uncertain data, which we have implemented in our system, is helpful in managing and searching for geo-tagged videos. Furthermore, the final goal of a geo-tagged video search application is to display

videos to end-users. During the video streaming, both resulting videos and meta-data are delivered to mobile clients for different purposes, *i.e.,* the meta-data are used to display the geographical information of each video frame on the map during video playback. The rapid development of smartphones and 3G/WiFi networks in recent years has enabled video streaming over wireless networks to account for an increasing portion of the global data traffic. However, the unstable conditions of wireless networks (*e.g.,* connection failures, bandwidth variations, etc.), may result in an unacceptable user experience and bandwidth waste with conventional media streaming protocols, such as the Real-time Transport Protocol (RTP) and the Real-Time Streaming Protocol (RTSP) [Ma 2011]. Dynamic Adaptive Streaming over HTTP (DASH) is known as a video delivery standard which enables high quality streaming of media content over HTTP. The visual content is encoded at a variety of different bitrates and the HTTP-client can automatically select the segment from the alternatives to download and play back based on the current network conditions. In our current implementation of a mobile DASH video recorder, the mobile device partitions the captured video file into a sequence of playable video segments where each segment contains a short playback interval of the content data. The task of transcoding media content to different qualities and bitrates is computationally expensive, especially in the context of large-scale video hosting systems. Therefore, it is preferably executed in a powerful cloud environment, rather than on a consumer-grade computer, due to the heavy workload. In order to support live or near-live streaming, the video processing latency needs to be minimized. To the best of our knowledge, there exists research on cloud-based video transcoders that are designed for processing batch tasks but none of them support live or near-live transcoding, especially when the video is under a viewing request before the required bitrate is ready for streaming and hence this video needs to be encoded urgently. To this end, we have designed a dynamic scheduling algorithm on video transcoding for DASH in a cloud environment, which is detailed in Chapter 5.

1.2 Overview

As illustrated in Figure 1.2, a geo-tagged video management system covers several different research topics. Some research has considered video uploading strategies based on the DASH standard [Seo 2012] or aiming at saving battery life of mobile devices that upload videos and the associated meta-data [Hao 2011]; while others have focused on the video and meta-data management from a remote hosting system. For example, Zhang *et al.* [Zhang 2013] proposed a multi-video summarization methodology based on the sensor meta-data. Wang *et al.* [Wang 2012] presented an automatic way to correct GPS data collected during video recording. In this thesis, three tasks of managing geo-tagged videos in a remote hosting system are presented. We have analyzed each particular problem and proposed a data model to provide a solution. Experimental results have shown the effectiveness and efficiency of the proposed methodologies. The contributions of each work are listed below:

Geo-tagged video index. We transform the video search problem into a spatial data selection problem. We then propose a multi-level grid-based index structure to help indexing and quickly search through geo-tagged videos. Moreover, a number of related query types, including typical spatial queries and queries with a radius or direction restriction, are introduced. This work has been published in the SIM^3 2012 [Ma 2012a] workshop and the GeoInformatica [Ma 2013] journal.

Uncertain geo-tagged video management. To address the mismatch between the viewable scenes of video content and the area represented by the sensor data, We have designed an uncertain data model for both individual video frames and video segments. A video segmentation method is also presented and the parsed segments are indexed with an extended R-tree. This work has been published in the GIS 2012 [Ma 2012b] conference.

Scheduling on video transcoding on the cloud. In order to support live or near-live streaming to mobile devices from a remote hosting system, we have designed a video transcoding scheduling methodology for DASH in a cloud environment. We first model the estimation of the video transcoding time (VTT) based on the video duration. Then, the scheduling algorithm is designed based on the VTT estimation. This work has been published in the MMSys 2014 [Ma 2014] conference.

1.3 Roadmap

The rest of this thesis is organized as follows. Chapter 2 introduces the preliminaries of my research, the dataset used for all the work and summarizes the related topics. Chapter 3 details the multi-level grid-based index structure. In Chapter 4, we demonstrate the uncertain model for both individual frames and video segments. Chapter 5 presents the scheduling strategies on video transcoding for DASH in the cloud environment. Finally, Chapter 6 concludes the thesis and proposes the potential future work.

Chapter 2

Related Work and Preliminaries

2.1 Related Work

My research on the geo-tagged video management covers three aspects: video storage and index based on meta-data, uncertain meta-data modeling, and scheduling on video transcoding for DASH in the cloud environment. Figure 2.1 illustrates the topics related to my research. In the following paragraphs, the research related to these three works are presented in detail.

2.1.1 Video Storage and Index

People may capture different places of interest (POI) within the same video but users may only be interested in the parts of the video that show a specific place. Therefore, parsing the video into segments to extract a specific place is essential for geo-relevant applications. Section 2.1.1.1 introduces the content-based methods on video segmentation and indexing. Section 2.1.1.2 introduces the research on the association of sensor information and the visual contents. Section 2.1.1.3 explores the related works on spatial data management.

2.1.1.1 Content-based Methods on Video Segmentation and Index

The aim of video segmentation is usually achieved using content-based methods. When a video is uploaded the server, it will be divided into elementary shots. A shot is defined as the consecutive frames from the start to the end of recording in a camera. It shows a continuous action in an image sequence.

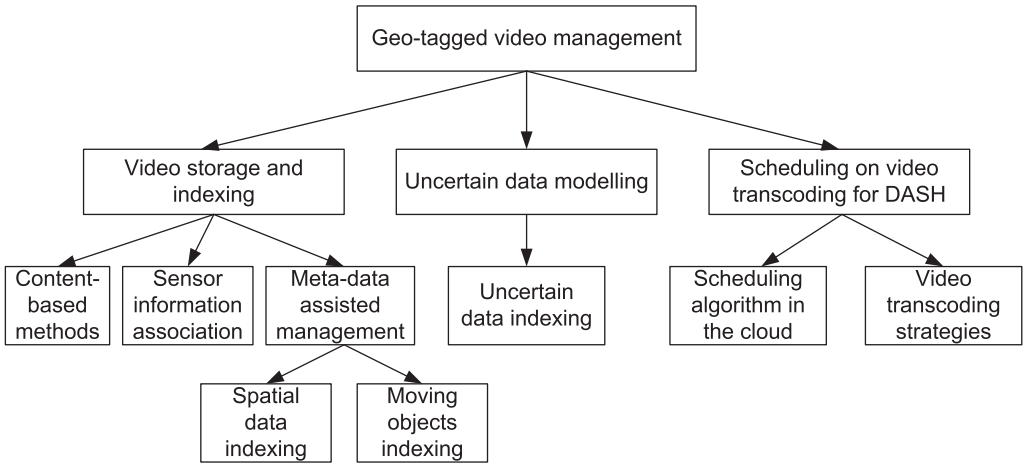


Figure 2.1: Topics related to my research.

The shot detection approaches are always used for video segmentation. Gao *et al.* [Gao 2005] described a technique for video shot boundary detection using rough fuzzy set. They classified twelve low-level features into five different types and used them to achieve high accuracy for shot boundary detection. Boreczky and Lynn [Boreczky 1998] proposed a video segmentation method using Hidden Markov Model (HMM) by considering both the visual and audio factors. Zhai and Shah [Zhai 2006] utilized Markov Chain Monte Carlo to determine the boundaries between video scenes. Cermekova *et al.* [Cernekova 2006] introduced a method on the detection of gradual transitions such as dissolves and wipes, which are the most difficult to be detected by using graph partitioning. Other research partition video into segments based on the spatio-temporal factors. Sundaram and Chang [Sundaram 2000] partitioned the video by using both video and audio features. Lezama *et al.* [Lezama 2011] proposed an efficient spatio-temporal video segmentation algorithm that incorporates long-range motion cues from both the past and the future frames, considering clusters of point tracks with coherent motion. Existing studies [Grundmann 2010, Xu 2012, Budvytis 2011] used different hierarchical graph-based models to achieve video segmentation targets. The current research on video indexing includes high dimensional indexing and semantic

indexing. The high dimensional indexing extracts features from the video segment and uses these features to process similarity comparison, while semantic indexing builds up the connection between the visual content and the semantics by video data mining, classification and annotation [Hu 2011]. The video clustering and indexing methods used are also based on the visual content. Ngo *et al.* [Ngo 2003] proposed a two-level hierarchical clustering structure to organize the content of sport videos: the top level is clustered by color feature while the bottom level is clustered by motion feature. The video indexing approaches always use the following key image processing algorithms: camera motion estimation and compensation, object segmentation and tracking techniques, and line detection. Akrami and Zargari [Akrami 2013] proposed a compressed domain video indexing method. It is based on the position of the blocks which are used for motion compensation in the coded video.

Summary: The content-based video indexing method is very helpful for understanding videos: content and semantics. The queries are always processed by feature similarity comparison between the input example and that in video database. The similarity comparison in high dimensional data is always time consuming, due to the heavy computational workload. Therefore, the content-based method is not appropriate for real-time application, especially geographical based video applications.

2.1.1.2 Associating Sensor Information with Videos

Associating geo-location and camera orientation information for video retrieval has become an active topic. Research [Zhu 2005, O'Connor 2008, Crandall 2009, Larson 2011] associating geographic information always help on video and image applications. Flickr[1] and Panoramio[2] enable users to upload their photos, associating the cameras' geo-location, and share with others. Hwang *et al.* [Hwang 2003] and Kim *et al.* [Kim 2003] proposed a mapping between the 3D world and the videos by linking the objects to the video frames in which they appear. Their work used

[1] http://www.flickr.com/groups/geotagging
[2] http://www.panoramio.com

GPS location and camera orientation to build links between video frames and world objects. Jaffe *et al.* [Jaffe 2006] proposed an automatic organization of digital photographs with geographic coordinates. They grouped the photos into hierarchies of location and time-based events. Luo *et al.* [Luo 2010] used GPS and images to estimate the camera's pose during capturing. Epshtein *et al.* [Epshtein 2007] introduced a hierarchical photo organization using geo-relevance. This method takes into account not only where the photos were taken but also the camera direction and the field of view. Liu *et al.* [Liu 2005] presented a sensor enhanced video annotation system (referred to as SEVA) which enables searching videos for the appearance of particular objects. SEVA serves as a good example to show how a sensor-rich environment can support interesting applications. However, it does not propose a broadly applicable approach to geo-spatially annotate videos for effective video search. Other than GPS and compass, sensors such as accelerometer, gyro, inertial sensor can also be used in video management. Pettersen *et al.* [Pettersen 2014] tracked the soccer players in Norway soccer league with ZXY system and processed queries based on player, region in the soccer filed and *etc.* The ZXY system includes sensors such as signal-based positioning sensors, accelerometers, gyros, compass, heart-rate sensor. Rowlands *et al.* [Rowlands 2012] used inertial sensors to detect event in sports and to build video indexing.

Navarrete and Blat [Navarrete 2002] utilized geographic information to segment and index video. Kim *et al.* [Kim 2010] proposed a vector-based approximation model to efficiently index and search videos based on the FOV (short for camera field-of-view) model[3]. It mapped an FOV to two individual points in two 2D subspaces using a space transformation. This model works well on supporting the geo-spatial video query features, such as point query with direction and bounded distance between the query point and camera position. However, it does not in-

[3]The FOV model is used to represent the camera viewable scene [Arslan Ay 2008]. The FOV coverage is with a pie-slice shape in 2D space and modeled with four parameters: camera location, camera viewing direction vector, camera viewable angle, and camera maximum visible distance. The detail on the FOV model is presented in Section 2.2.1.

vestigate query optimization issues. Vector model works effectively for basic query types, such as point and range query, however does not support the k-NVS query. Moreover, there was no consideration in scalability.

Summary: These research build the connection between the visual content and their corresponding geo-properties. In our geo-tagged video search applications, the individual frames and video segments are hence treated as spatial data during storage and indexing.

2.1.1.3 Meta-data Assisted Management

In this section, we explore the existing methodologies on spatial data indexing and moving objects indexing.

Spatial data indexing. There exist two categories of spatial data indexing methods: *data-driven structures* and *space-driven structures* [Rigaux 2001]. The R-tree family (including R-tree [Guttman 1984], R^+-tree [Roussopoulos 1987], R^*-tree [Beckmann 1990] and their variations) belongs to the first fold and is the most widely used multidimensional spatial data indexing structure. The space occupied by the objects are propagated up the hierarchy, with the identities of the objects being implicit to the presentation. Guttman [Guttman 1984] proposed the R-tree, which is a dynamic tree data structure, as an extension of the ubiquitous B-tree in multi-dimensional space, for spatial data indexing. Each node in the R-tree is represented as a bounding rectangle. To access a node of the indexed collection, one typically follows a path from the root down to one or several leaves, testing each bounding rectangle at each level for either containment or overlap. R^+-tree allows partitions to split rectangles then zero overlap among intermediate node entries can be achieved. Both R-tree and R^+-tree are nondeterministic in allocating the entries on the nodes while R^*-tree is on the contrary. R^*-tree utilizes the idea of 'forced-reinsert' by deleting some rectangles from the overflowing node, and reinserting them. Recently, Beckmann and Seeger [Beckmann 2009] presented the Revised

R*-tree (RR*-tree) by redesigning the algorithm ChooseSubtree, so that irrelevant entries, which are not able to host the new object, are pruned. The variations of the R-tree family is broadly used among different application areas (*e.g.,* spatio-temporal databases, multimedia, warehousing and data mining). Nanopoulos *et al.* [Nanopoulos 2006] explored different variations of R-tree for different applications in their book. For example, the Hilbert-R-tree [Kamel 1994, Wang 2013] used for spatial data indexing is another kind of R-tree that acts as a B^+-tree when objects are inserted. The position of the inserted rectangle is chosen based on the Hilbert value of the center of the rectangle. GeoTree [Kim 2014] is used to index the FOV model by building and rotating the minimum bounding rectangle (MBR) of a series of consecutive FOVs and indexed the MBRs in R-tree. However, their work is based on the assumption that the camera's orientation does not change frequently. Besides, there also exist other structures that belong to *space-driven structures*, *e.g.,* M-tree [Ciaccia 1997], X-tree [Berchtold 2001], and *etc.*

Summary: These methods are designed mainly for supporting efficient query processing when the construction and the maintenance of the data structure is computationally expensive. Moreover, although these structures can support for special query types, such as queries with direction restriction or bounded radius, they are not that efficient.

The second fold includes methods such as the grid file [Nievergelt 1984], quadtree [Finkel 1974], k-d tree [Bentley 1975], octree [Meagher 1982] and Voronoi diagram [Okabe 2000]. These structures partition the metric space in different regions following specific roles. The grid-based method partitions the space from which the data is drawn into rectangular cells by overlaying it with a grid. The grid can be with either equal size or the line of the grid be drawn at arbitrary positions that are dependent on the underlying data. For example, the grid file uses the fixed cell size while quadtree and k-d tree merge the spatially adjacent empty grid cells into larger ones. The original design of quadtree or k-d tree becomes

impractical when the data volume becomes very large. Thus, the variations (*e.g.,* k-d-B tree [Robinson 1981], Local Split Decision tree (LSD-tree) [Henrich 1989], hB-tree [Lomet 1990, Evangelidis 1997]), referred as *bucket methods* [Samet 2006], were developed for accessing large scale of data by treating each disk page containing objects as a leaf node of the tree. The Voronoi diagram divides the space into a number of regions with respect to a finite set of points (referred as representative point), and each region contains one representative point. Every point within a Voronoi region is closer to its corresponding representative point than others. The index structure using Vonoroi diagram keeps safety region for each object so as to quickly process kNN query. The k-d tree are recently widely used in GPU-based parallel processing [Santos 2012, Liu 2012, Gieseke 2014]. Researches also use either the grid structure [Chon 2003], the skip quadtree [Eppstein 2005], Voronoi diagram [Nutanong 2008], or DP-tree [Peng 2012] to process multiple types of queries.

Summary: These data structures consider spatial objects as points or small rectangles, and none of them are appropriate to index our FOV model. The reason is that the FOV model has a large coverage region and consecutive FOVs have large overlap region. This results in the difficulty in partitioning the space while guaranteeing each FOV only belongs to one cell without cell overlapping.

Moving objects indexing. Besides indexing multi-dimensional spatial objects, indexing moving objects is another active and important topic on spatial data management. Ilarri *et al.* [Ilarri 2010] presented a detailed survey on indexing moving objects. Indexing moving objects can be divided into three main categories: indexing historical, current and future locations of spatial objects. Several methods have been proposed to index historical locations of spatial objects and expected trajectories [Faria 2002, Hadjieleftheriou 2002, Pfoser 2000, Agarwal 2003, Hadjieleftheriou 2006]. Specifications and framework for efficient indexing in spatio-temporal databases can be found in [Theodoridis 2002] and [Faria 2002] respectively. In particular, the work in [Pfoser 2000] proposes a B-tree based scheme,

TB-tree, that strictly preserves trajectories, *i.e.,* leaf nodes in the index contain segments belonging to one trajectory. More works have focused on indexing techniques that can facilitate queries on current and future positions of points. Some research are only focus on indexing the current positions of moving objects, for example, hashing [Song 2001], the LUGrid [Xiong 2006], the RP-tree [Lin 2006], the R^R-tree [Biveinis 2007]. The HTPR*-tree [Fang 2012] utilized the parameterized R-tree structure and considered the historical property of the moving objects. Samet *et al.* [Samet 2013] introduced the moving objects indexing methods by using the loose quadtree. This work deals with objects with large geographic coverage instead of point objects. Other structures consider both current and future positions of the moving objects. The time-parametric tree (TPR-tree) [Šaltenis 2000] is an R-tree based index where the location of a moving point is represented by a reference position and a corresponding velocity vector. The following three algorithms convert the spatial and temporal data into only one dimension and therefore build an B^+-tree based indexing structure: Jensen *et al.* [Jensen 2004] use B^x-tree to efficiently index moving objects, which is constituted of a B^+-tree with space-filling curves, *i.e.,* Peano curve (Z-curve) and Hilbert curve (H-curve); Chen *et al.* [Chen 2008] introduce the structure and basic query algorithm of the ST^2B-tree. It introduces a self-tuning framework for tuning the performance of the B^x-tree while dealing with data skew in space and data change with time. Yiu *et al.* [Yiu 2008] proposed the B^{dual}-tree that converted moving objects into a 2d-dimensional dual space, which contains d location dimensions and d velocity dimensions. Each domain is then fulfilled with Hilbert curve. Finally, the BB^x-index [Lin 2005] and the R^{PPF}-tree [Pelanis 2006] were introduced to index the historical, current, and future positions of moving objects.

Summary: The ideas in dealing with moving objects are useful for building up index of video segments based on the consideration of temporal domain. However, they only consider the spatiotemporal property of the moving objects but nothing

related to the visibility of the video content, which is the main task of the geo-tagged video management.

2.1.2 Uncertain Data Modelling

Since the meta-data are used to help indexing and searching for geo-tagged videos, the uncertainty of the sensor data affects the search results. There exist two categories of techniques for indexing uncertain data with arbitrary probability density functions (PDF). The first type is based on probabilistically constrained regions (PCRs) [Chen 2007, Tao 2007, Cheng 2004b, Angiulli 2012]. It uses a so-called "x-bound" to restrict the probability of a region so that candidates can be pruned when processing probabilistic constrained queries. In particular, Cheng *et al.* [Cheng 2004b] cluster the data points with similar degrees of uncertainty together. Tao *et al.* [Tao 2007] develope U-tree to minimize the query (both fuzzy and non-fuzzy queries) overhead. The UP-index [Angiulli 2012] is a pivot-based index structure supporting distance based range query, which prunes large percentage of candidates with little time and speeds up the range query computation. The second type is based on geometry-based index structures [Cheng 2004a, peter Kriegel 2006, Singh 2007]. More specifically, the uncertain region of multidimensional uncertain objects is grouped with an R-tree, where each data unit is the minimum bounding rectangle (MBR) of a PDF. One popular model for uncertainty is that, at any point in time, the location of the object is within a certain distance, d, of its last reported position. The study in [Cheng 2004a] utilized this model to evaluate the uncertainty of moving objects. Li *et al.* [Li 2007] use R-tree as query filtering schemes, and designed an optimized Gaussian mixture hierarchy (OGMH) to index objects for both certain and uncertain queries. Böhm *et al.* [Böhm 2006] propose a Gauss-Tree to probabilistic feature vectors, which is a R-tree based structure but stores the parameter space of Gaussian PDF instead of the spatial objects. Besides using the basic R-tree structure, Kim *et al.* [Kim 2007]

attach a secondary index for fast access to the leaf nodes. Zhang *et al.* [Zhang 2010] proposed UI-tree, which is a R-tree based inverted index structure, to support various queries (*e.g.,* range queries, similarity joins, continuous or discrete queries over multidimensional uncertain objects). They also proposed the U-Quadtree Structure [Zhang 2012] that utilize quadtree to index the cells in the space and B^+-tree as a secondary index to entries within the same cell. Emrich *et al.* [Emrich 2012] used a diamond shape and the first-order Markov Chain model to approximately present the trajectory between two sampling points. The uncertain spatio-temporal data are then indexed with UST-tree, which stores the MBR approximation, the diamond approximation, the linear approximation functions, and the object ID in the R-tree-based structure.

Summary: To the best of our knowledge, existing works only consider the uncertainty of the location but none of them is related to any visual contents, which are inappropriate to solve the problem on the mismatch between the visual contents and the geographical regions represented by the sensor meta-data. The uncertainty region of the FOV model is quite complex, so that it is necessary to design the uncertain model for FOV and hence for video segment.

2.1.3 Video Transcoding Strategies in the Cloud Environment

2.1.3.1 Video Transcoding Services in the Cloud

There exist several cloud services that offer video transcoding. Amazon released their Elastic Transcoder [Amazon 2013] in January of 2013. It executes transcoding jobs using Amazon's Elastic Compute Cloud (Amazon EC2)[4] and stores the video content in Amazon's Simple Storage Service (Amazon S3)[5]. Amazon's Elastic Transcoder manages all aspects of the transcoding process transparently and automatically. It also enables customers to process multiple files in parallel and organize their transcoding workflow using a feature called transcoding pipelines.

[4] http://aws.amazon.com/ec2/
[5] http://aws.amazon.com/s3/

Zencoder [Zencoder 2010] provides video transcoding services as well. In addition to providing typical video transcoding services in the cloud, it also supports live cloud video transcoding. EncoderCloud [EncoderCloud] also provides the same web-based "pay-as-you-go" service and helps to build applications on top of other service providers (*e.g.,* Amazon EC2 and RackSpaceCloud [Rackspace]), but offers a different pricing policy – charging by the volume of the total amount of source video transfered in and encoded video transfered out.

Summary: These services provide the capability of video transcoding in the cloud, but the transcoding scheduling mechanism is transparent to end-users. A batch of jobs are processed in the elastic cloud but no user interaction is considered, which is not practical for online video hosting service.

2.1.3.2 Scheduling Strategies

There exist a few widely used scheduling algorithms and many operating systems use extended or combinations of these algorithms.

- **First In First Out (FIFO):** FIFO is the simplest scheduling algorithm, which simply queues the all the waiting jobs and processes them in the order that they arrive in the queue.

- **Shortest Job First (SJF):** SJF enables the scheduler to assign the job, whose estimated processing time is the shortest among all the waiting jobs, to be processed in the first place.

- **Round-Robin (RR):** RR assigns a fixed processing time for each job in the queue. If the processor cannot finish the job in the current cycle, it gives up and allocates the resource to another job and re-processes the current job when the next cycle comes.

- **Fixed Priority (FP):** The priority of each job is pre-defined, and the scheduler will choose to process the job with highest priority first. When a coming

job's priority is higher than the current processing job, the current processing job gets interrupted. The processing will resume when the job with higher priority is finished.

- **Multilevel Queue (MQ):** MQ is hybrid scheduling algorithm by dividing jobs into groups and each group has different scheduling strategies.

For example, Windows NT/XP/Vista uses a multilevel feedback queue, a combination of fixed priority preemptive scheduling, round-robin, and first in first out. In this system, processes can dynamically increase or decrease in priority depending on if it has been serviced already, or if it has been waiting extensively. Every priority level is represented by its own queue, with round-robin scheduling amongst the high priority processes and FIFO among the lower ones. In this sense, response time is short for most processes, and short but critical system processes get completed very quickly. Since processes can only use one time unit of the round robin in the highest priority queue, starvation can be a problem for longer high priority processes.

Cloud computing provides tremendous computing resources for applications but there is no universal "best" scheduling algorithm. Instead, they are usually optimized for certain applications. One study [Li 2012] introduces a parameter-tuning framework by combining the bitrate and encoding speed as encoding cost and provides a cost optimization method for video cloud transcoding. Li *et al.* [Li 2005] proposed a parallel video encoding strategy considering a load balance factor. By considering the granularity of the load partitions and the associated overheads caused, they utilized the divisible load theory paradigm to distribute the video frames among processors. Zaharia *et al.* [Zaharia 2009] presented a fair job scheduling for Hadoop to minimize the job response time. Kllapi *et al.* [Kllapi 2011] presented an optimization of scheduling dataflows on these three aspects: minimizing completion time, minimizing the monetary cost given a deadline, and finding the trade-off between completion time and monetary cost.

Summary: These approaches investigate neither the specific properties of DASH[6] (*e.g.,* correlation between segments, alternative bitrates of uploaded segments) nor the interaction between the service hosts and the end-users. In order to fully utilizing these computing resources in the cloud, an appropriate scheduling algorithm for the specific application is essential. To the best of our knowledge, there exists no work on scheduling video DASH transcoding for cloud environments, especially while untranscoded segments already have pending video watching requests.

2.2 Preliminaries

2.2.1 Modeling of Camera Viewable Scene

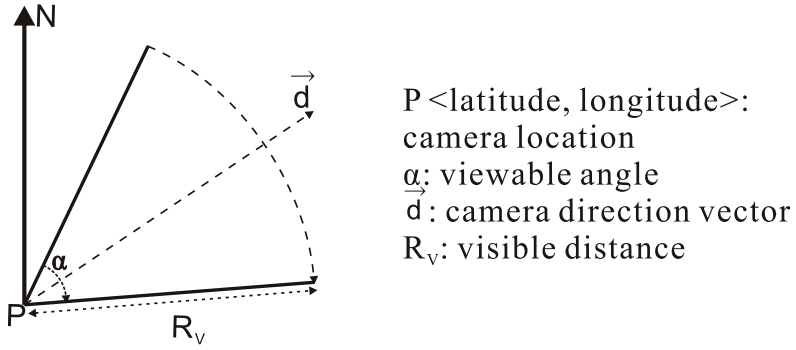


Figure 2.2: Illustration of the field-of-view (FOV) model in 2D space.

The *camera viewable scene* is what a camera in geo-space captures. This region is referred to as *camera field-of-view* (FOV in short) with the shape of a pie-slice [Arslan Ay 2008]. The FOV coverage in 2D space can be defined with four parameters: the camera location P, the camera viewing direction vector $\overrightarrow{d}$, viewable angle α, and maximum visible distance R_V (see Figure 2.2). These geo-spatial meta-data can be obtained from the embedded sensors during the video recording. The location P of camera is the $<$ *latitude*, *longitude* $>$ coordinate reading from a positioning device (*e.g.,* GPS and/or Cricket coordinates [Priyantha 2000]). In

[6]Dynamic Adaptive Streaming over HTTP (DASH) can provide seamless, bitrate adaptive streaming to clients. It works by breaking the content into a sequence of small HTTP-based file segments, and each segment containing a short interval of playback time.

our current implementation, the value of P is collected from a GPS reader embedded in the smartphones since we only collect outdoor data. The camera viewing direction vector $\overrightarrow{d}$ is acquired from a digital compass, which also exists in most of smartphones. We use θ to represent its value with respect to the North. The camera viewable angle (α) is calculated based on the camera and lens properties for the current zoom level [Graham 1965]. The visible distance R_V is the maximum distance at which a large object within the camera's FOV can be recognized. Then, the camera viewable scene at time t is denoted by the tuple $FOV\left(P\left\langle lat, lng\right\rangle, \theta, \alpha, R_V, t\right)$. The detailed data acquisition and stream synchronization method is stated in Section 2.2.3.1.

2.2.2 DASH Standard

2.2.2.1 HTTP Streaming Fundamentals

With the development of content delivery networks (CDN), Hypertext Transfer Protocol (HTTP) streaming has emerged as the *de-facto* streaming standard, replacing the conventional streaming with the Real-Time Transport Protocol (RTP) and Real-Time Streaming Protocol (RTSP). Existing streaming platforms such as Microsoft's Smooth Streaming (MSS) [Microsoft Corporation 2012], Apple's HTTP Live Streaming (HLS) [Pantos 2012], and Adobe's HTTP Dynamic Streaming (HDS) [Adobe System Inc], all use HTTP streaming as their underlying delivery method. The commonalities [Seo 2012] of these techniques are:

- They split an original encoded video file into small pieces of self-contained media segments,
- They separate the media description into a single playlist file,
- They deliver segments over HTTP.

Among each other, these techniques differ in the following way:

- MSS is a compact and efficient method for the real-time delivery of MP4 files from Microsoft's IIS web server, using a fragmented, MP4-inspired ISO Base Media File Format (ISOBMFF),
- HLS uses an MPEG-2 Transport Stream (TS) as its delivery container format and utilizes a higher segment duration than MSS,
- HDS is based on Adobe's MP4 fragment format (F4F) and its corresponding XML-based proprietary manifest file (F4M).

Once an HTTP client sends a request and establishes a connection between the server and itself, a progressive media download is activated until the streaming is terminated [Stockhammer 2011]. Disadvantages of a progressive download include:

- Unstable conditions of the network, especially a wireless connection for mobile clients, may cause bandwidth waste due to reconnection or rebuffering events,
- It does not support live streaming (*e.g.,* concert or football match),
- It does not support adaptive bitrate streaming.

2.2.2.2 The DASH Standard

Published in April 2012, DASH [DAS 2012] addresses the above weaknesses of simple, progressive HTTP streaming. Figure 2.3 illustrates a simple streaming scenario between an HTTP server and a DASH client. The visual content is then encoded at a variety of different bitrates and the HTTP-client can automatically select the segment from the alternatives to download and play back based on current network conditions. In our current implementation of the mobile DASH video recorder, the mobile device breaks the captured video file into a sequence of playable video segments and each segment contains a short interval of playback time of the content. The client selects the segment with the highest possible bit rate that can be downloaded in time for smooth and seamless playback, without causing rebuffering events.

DASH standardizes the two most important components: the Media Presentation Description (MPD) and the segment formats. MPD is a document that contains metadata (*e.g.*, a manifest of the available content, its various alternatives, their URL addresses, and other characteristics) required by a DASH client to construct appropriate HTTP-URLs to access Segments and to provide the streaming service to the user. The Segment formats specify the formats of the entity body of the request response when issuing a HTTP GET request or a partial HTTP GET with the indicated byte range through HTTP/1.1. In order to support use with DASH, a delivery format should have the property that decoding and playback of any portion of the media can be achieved using a subset of the media which is only a constant amount larger than the portion of the media to be played. To implement this functionality, each media segment is assigned a unique URL, an index, and explicit or implicit start time and duration. Each media segment contains at least one stream access point, which is a random access or switch-to point in the media stream where decoding can start using only data from that point forward. Both ISOBMMF and MPEG-2 TS are supported in DASH.

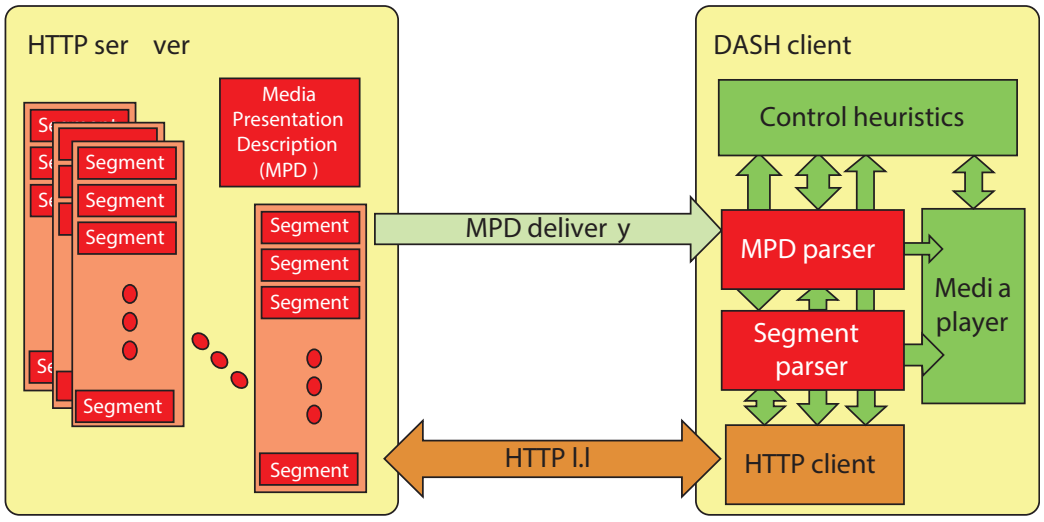


Figure 2.3: Scope of the MPEG-DASH standard (red blocks in the figure) [Sodagar 2011].

Table 2.1: The statistics of Dataset1 and Dataset2.

Parameters	Values
Dataset1	
# of videos	1,200
# of frames	138,774
Overall duration	38.55 hours
Longest video	18.45 minutes
Shortest video	3 seconds
Collecting manners	walk or on bus
Dataset2	
# of videos	247
# of frames	12,555
Overall duration	209.25 minutes
Longest video	10 minutes
Shortest video	1 seconds
Collecting manners	walk

2.2.3 Datasets

All the experiments are processed on four datasets: two small sets of real-world videos and sensor meta-data, one large synthetic dataset, and one testing video uploading set. We used the first real-world dataset to show the functionality of the proposed methodologies and the synthetic dataset to demonstrate the scalability for large-scale applications. The last one is used to test the performance of the scheduler by generating different video streams.

2.2.3.1 Real-world Dataset

We collected 1200 geo-tagged videos within Singapore (***Dataset1***), and 247 videos in Chicago (***Dataset2***), recording the event of the NATO Summit 2012 using smartphones (Android Devices and iPhone). The videos are collected in Singapore by people either walking in the open space or taking a bus. Conversely, all the videos collect in Chicago are by people walking in the street. The statistics on these two datasets are listed in Table 2.1.

Data Acquisition. Our group have made great effort on collecting the geo-tagged videos with associated sensor data. Figure 2.4 shows the first generation of data collection tool. The videos are recorded by the HD camera and the meta-data are collected as the same time using GPS and digital compass. All these three streams are directly written to the disk on the laptop. With the development of the integrated hardware on mobile devices, we implement the second generation of data collection tool on smartphones and tablets (shown in Figure 2.5). All videos and the meta-data are directly store in the mobile devices, which is much easier for users to carry than the first generation. One important observation is that the different sampling rates among different devices (*i.e.,* the video clip contains about 25 or 30 frames per second, the digital compass can collect 40 samples per second, while the GPS device can only get one location per second). Therefore, we match the temporally closest location of the camera to the frame based on timecodes and estimate the location at other timestamps through a positional interpolation technique.

Figure 2.4: The first generation of the geo-tagged video collection tool.

(a) Main menu. (b) Recording menu.

Figure 2.5: The second generation of the geo-tagged video collection tool.

2.2.3.2 Synthetic Dataset

Due to the difficulty of collecting large set of real-world videos associating with geographical meta-data (*i.e.,* camera location and viewing direction), a synthetic dataset (***Dataset3***) of moving cameras was used to test the performance of the grid-based index structure with large-scale data. The dataset for moving camera trajectories was generated with positions inside a $75km \times 75km$ region in the geo-space using the Georeferenced Synthetic Meta-data Generator (GSMG) [Arslan Ay 2010]. The interface of the GSMG is shown in Figure 2.6. Users can set up all the parameters through the GUI and the statistics of the generated meta-data are displayed on the right.

The generated synthetic meta-data exhibit equivalent characteristics to the real world data. The camera's viewable angle is $60°$ and the maximum visible distance is $250m$ [Arslan Ay 2008]. In the experiments, we chose 100 randomly-distributed center points within the area and generate $5,500$ moving cameras around these center points. Hence each one of the cameras is traced for $1,000$ seconds, with snapshot of one frame per second, due to the sampling rate of GPS and the compass. (Note that the digital compass can achieve 30 or 40 readings per second but GPS can only get one sample per second. Thus, we only collect one snapshot per second.) Therefore we have a dataset with about 5.4 million video frames. The center points are randomly distributed in the experiment region, which are used as the initial positions of the camera location. Subsequently, the cameras start to move inside

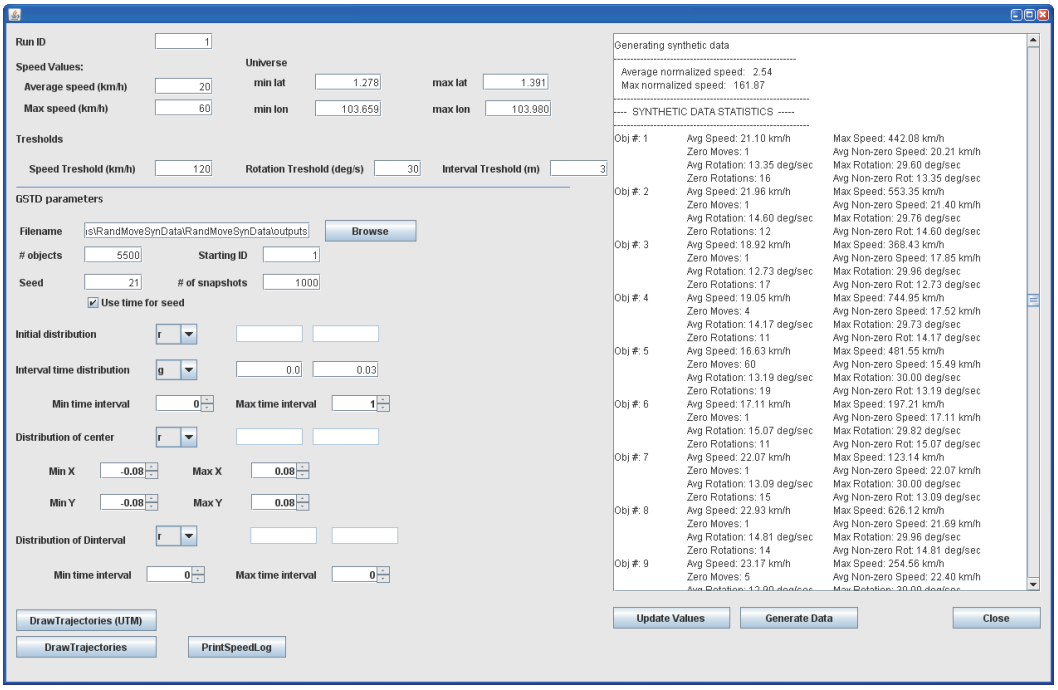


Figure 2.6: The GUI of Georeferenced Synthetic Meta-data Generator.

the region under a maximum speed limit, as well as a viewing direction rotation limit. The speed of the cameras, and the position of center points, affect the final distribution of the frames. Faster movement causes the frames distributed uniformly throughout the region in contrast to slower movement. To simulate real-world case, we set the maximum speed of moving cameras as $60km/h$, with the average speed as $20km/h$. Besides the speed limit, we also set the camera's maximum rotation limit as 30° per second, which guarantees that the camera rotates smoothly and not jump from one direction to another, the same as what people do when they are capturing videos. With restriction to these limitations, unexpected data (*e.g.,* the object's speed is larger than the speed threshold, viewing direction rotates over the rotation limit and etc.) are removed from the dataset. The parameters used are summarized in Table 2.2 and the sampling trajectories of 100 moving cameras are shown in Figure 2.7.

Table 2.2: Parameters used when generating Dataset3.

Parameter	Value
# of Center points	100
Speed Limit	60km/h
Average Speed	20km/h
Rotation Limit	30°/s
# of Cameras	5500
# of Snapshots	1000
# of FOVs	5405051
viewable angle of FOV (α)	60°
visible distance of FOV (R_V)	250m

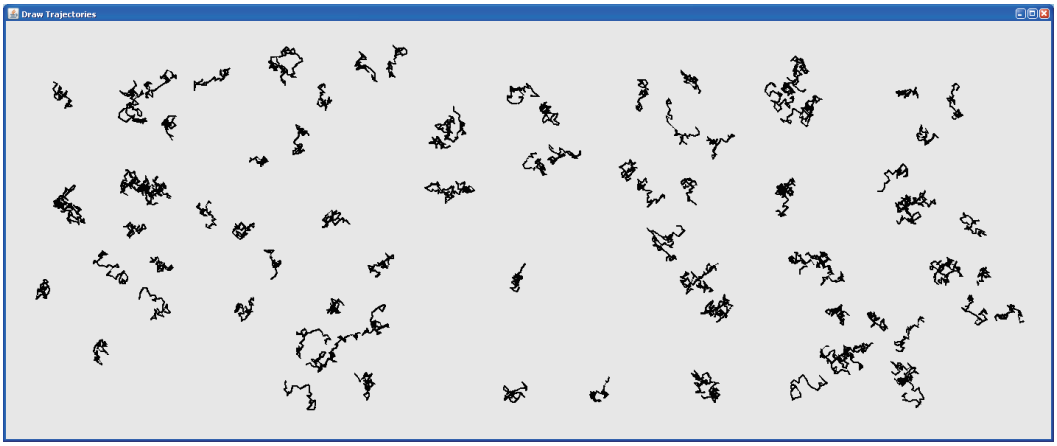


Figure 2.7: The sampling trajectories generated for experiments.

2.2.3.3 DASH Dataset

To test the performance of the scheduler, two DASH datasets are applied: a synthetic dataset (***Dataset4***) with generated video uploading traces and a real-world dataset with video uploading trace (***Dataset5***).

Dataset4 contains 400 videos (collected from Android devices), which consists of 20,000 video segments. The number of segments in each individual video varies from 10 to 197, which means that the video lengths varies from around 30 seconds to 16 minutes.

Dataset5 includes nine uploaders submitting a total of 259 videos, which consist of 4,899 video segments. The length of these video segments varies from 0.1 to

Table 2.3: The statistics of Dataset4 and Dataset5.

Parameters	Values
Dataset4	
# of videos	400
# of segments	20,000
Overall duration	23.94 hours
Longest video	16.4 minutes
Shortest video	30 seconds
Longest segment	6.5 seconds
Shortest segment	0.2 second
Dataset5	
# of videos	259
# of segments	4,899
Overall duration	6.8 hours
Longest video	16.35 minutes
Shortest video	1 second
Longest segment	10 seconds
Shortest segment	0.1 second

10 seconds and the overall duration of the segments is around 6.8 hours. Most of the segments are five seconds long, while about 2% of the segments have a longer duration (*e.g.,* 10 seconds), and 4% have a shorter duration. The statistics on Dataset4 and Dataset5 is listed in Table 2.3.

2.2.4 Notations

All the notations used in this thesis are listed in Table 2.4.

Table 2.4: Notations used in this thesis.

Symbols	**Meanings**
FOV model	
$P < lat, lng >$	Camera location with geo-coordinates
α	Camera's viewable
$\overrightarrow{d}$	Camera's direction vector

Continued on next page

Table 2.4 – continued from previous page

Symbols	Meanings
θ	The angle between $\overrightarrow{d}$ and the North (0°)
R_V	Camera's visible distance
t	The time stamp of a video frame
Grid-based index structure	
$C_{\ell 1}(m,n)$	The mth row and the n column of the first-level grid
$C_{\ell 2}(f,g)$	The fth row and the g column of the second-level grid
$C_{\ell 3}(\beta_1,\beta_2)$	The viewing direction of FOV between β_1 and β_2 of the third-level grid
δ	The grid size (in metres)
s	The number of intervals divided in each first-level cell
M	The row number of first-level cells exist in the space
N	The column number of first-level cells exist in the space
x	The value of degree intervals in the third-level cell
ε	The value of the error margin when processing directional queries
Video segment presentation	
$FOV_{v_j}(i)$	The ith frame from the set of FOV objects for video v_j
$\mathbb{FOV}$	The set of all FOVs
$VS_{v_j}(S_{fr},E_{fr})$	The video segment with the starting frame S_{fr} and the ending frame E_{fr}
q	The query point
q_r	The query rectangle
ϕ	The overlap threshold between the q_r and $C_{\ell 1}(m,n)$
MIN_R	The minimum bounded radius from the camera location to the query input

Continued on next page

Table 2.4 – continued from previous page

Symbols	Meanings
MAX_R	The maximum bounded radius from the camera location to the query input
The uncertain model of FOV	
β_ε	The angle offset between the camera viewing direction and the line of |Pq|
θ_ε	The value of compass error
d	The distance between the camera and the targeted object
σ	The Gaussian parameter used in measuring the uncertainty of obstacles
d_ε	The GPS error range
$\mu, \rho, \sigma_x, \sigma_y$	The Gaussian parameter used in measuring the uncertainty of camera location
β_b	The boarder value of the direction domain
$dist_b$	The boarder value of the distance domain
τ	The probability threshold
δ_b	The micro-block size
The scheduler	
VTT	The video transcoding time
VTJ	The video transcoding job
$dur(V_{B_k})$	The duration of video segment V_{Bk}
T_{err}	The normalized error between the calculated VTT and measured VTT
G_k, G_θ	The coefficient of the Gamma distribution
L_s	The startup latency of the VTJ
N_{QS}	The number of quality switches
N_{RE}	The number of rebuffering events

Continued on next page

Table 2.4 – continued from previous page

Symbols	Meanings
T_{mr}	The mean rebuffering time
MOS	Mean opinion score
LBF	Load balance factor
N_{pro}	The number of processors in the cloud environment
$T_{pro(i)}$	The overall processing time of processor i
$Job_{DL}^{B_i}$	The deadline of the VTJ on V_{B_i}
M_{up}	The video uploader
VTM	The video transcoding manner
λ	The mean inter-arrival time
L_{va}	The video available latency
Bit_r	The targeted video bitrates
N_{wr}	The request arrival rate

Chapter 3

Multi-level Grid-based Index

3.1 Introduction

In the presence of a huge size video depository such as YouTube[1], effectively and efficiently searching such a repository for meaningful results is still a challenging problem. Current video search techniques that annotate videos based on the visual content are struggling to achieve satisfactory results in online user-generated videos (UGVs), particularly in accuracy and scalability. In this work, we utilized the camera's FOV to convert the visual content into a series of spatial objects. Compared to visual content, the meta-data occupies much less space, which makes searching among large scale of videos practical.

For a practical implementation of search engine with a large amount of geo-tagged videos and their associated geospatial meta-data, there remain some other critical issues to be resolved. For example, the performance of searching sensor meta-data should efficiently handle large video depositories (such as YouTube). The widely used index structure, *i.e.,* R-tree [Guttman 1984] (and/or its variance [Roussopoulos 1987, Beckmann 1990]) has been the chosen structure to index geometric figures. However, its performance deteriorates as the number of figures indexed increases greatly. Assuming all videos in a huge depository are represented using sensor meta-data, *i.e.,* streams of geospatial objects, a R-tree may suffer significantly to provide enough search performance due to its increased heights.

Furthermore, from the semantic perspective, in searching videos through their

[1] http://www.youtube.com

geographic coverage, distance and direction are two important criteria that can help to improve query functionality. For example, people are searching for a video that records a large building (*e.g.,* the Esplanade). Some would like to see the panoramic view while others prefer to discover a specific part of the building. For example, Figure 3.1 shows a group of images that are taken of the Esplanade at different distances away. In such application, searching videos based on distance helps to accelerate the query processing and retrieve meaningful results to end-users. However, due to the large geographical coverage region of the FOV, indexing FOVs with a R-tree or its variations can only provide overlap calculation but not prune any unnecessary search on-the-fly if the query is with distance restriction. Moreover, searching for videos captured from a specific direction is helpful in applications such as event reviews and video summaries. Another possible application can be to automatically extract panoramic images of a building from a video and use these images to construct the 3D model of the building. With the viewing direction information of the camera, we can select the smallest number of images, which are with complete coverage but least redundant details, to finish the target. The R-tree can support this query functionality by adding one more dimension, but its performance will deteriorate due to the increasing searching dimension, as well as the additional coverage computation.

The above drawbacks with the R-tree based structures on searching large-scale geo-tagged videos raise the question that it is essential to develop a study of high performance index structure which can effectively harness the captured geospatial meta-data. An important observation is that the geo-space is bounded while the number of videos is almost un-bounded. Based on this observation, we propose a new multi-level grid-based index structure for geo-tagged video search by fully utilizing their geographic properties. Specifically, this index structure is created to allow efficient access of FOVs based on their distance to the query location and the cameras viewing direction. Based on these criteria, we introduce a number of

Figure 3.1: The importance of querying with distance restriction. The images are taken by people at different distances away from the Esplanade. The images in the first row show the whole theater while those in the second row show a specific part of the Esplanade.

related query types which support better human perception of images in resulting videos, including typical spatial queries (*i.e.,* point queries, range queries, and kNN queries) and the queries with bounded radius or direction restriction.

Among these, one of the unique query types proposed in this work is a *Nearest Video Segments query* (k-NVS). This query retrieves the k closest video segments that show a given query point. k-NVS query can significantly enhance human perception and decision in identifying requested video images, especially when search results return a large number of videos in a highly populated area. Moreover, the query can additionally specify a bounded radius range to get the closest video segments that show the query point from a distance within a given radius range. Alternatively, the query may specify a certain viewing direction to specifically retrieve the k closest segments that show the query point from that direction, which is critical in human perception of objects. An example application that can utilize k-NVS is automatically building panoramic (360 degree) view of a point-of-interest (*i.e.,* a query point). The application needs to search for the nearest videos that look at

the query point from different viewing directions and that are within a certain distance from it. Similarly, k-NVS can serve as a useful feature for video browsing applications. For example, on a map-based interface the videos that show important landmarks from the users viewing point can be quickly retrieved as the user navigates by issuing continuous k-NVS queries.

In the remaining sections of this work, we describe our geo-tagged video indexing and searching approach, and report on an extensive experimental study with synthetic dataset. The results we have obtained illustrate that the three-level grid index structure supports new geospatial video query features. It efficiently scales to large datasets and significantly speeds up the query processing (compared to the R-tree) for finding the related video segments, especially for queries with direction. The rest of this work is organized as follows. Section 3.2 details the proposed data structure. Section 3.3 introduces the new query types and details the query processing algorithms. Section 3.4 reports the results on the performance evaluations of the proposed algorithms. Finally, Section 3.5 summarizes the work.

3.2 Grid Based Indexing of Camera Viewable Scenes

We present our design of the memory-based grid structure for indexing the coverage area of each camera viewable scene. The proposed structure constructs a three-level index, where the first level indexes the video FOVs according to location, the second level indexes them based on the distance to the overlapping cell, and the third level builds an index based on FOV viewing direction. That is, the FOV is indexed by any first level cell if its coverage region overlaps with the cell. If the FOV overlaps with the first-level cell, the distance between the camera location P and the center of the first-level cell is then indexed in the second-level cell by which subcell P locates. The proposed three-level index structure is illustrated in Figure 3.2. The collections of cells at the first, second, and third level are denoted by $C_{\ell 1}$, $C_{\ell 2}$, and $C_{\ell 3}$, respectively. Note that, each level of the index structure stores only the ID

numbers of the FOVs for the efficient search of the video scenes. The actual FOV meta-data (*i.e.*, P, θ, α, R_V and t values) are stored in a MySQL database where the meta-data for a particular FOV can be efficiently retrieved through its ID number. Figure 3.3 illustrates the index construction with an example of a short video file. In Figure 3.3 (c), only the index entries for the example video file are listed.

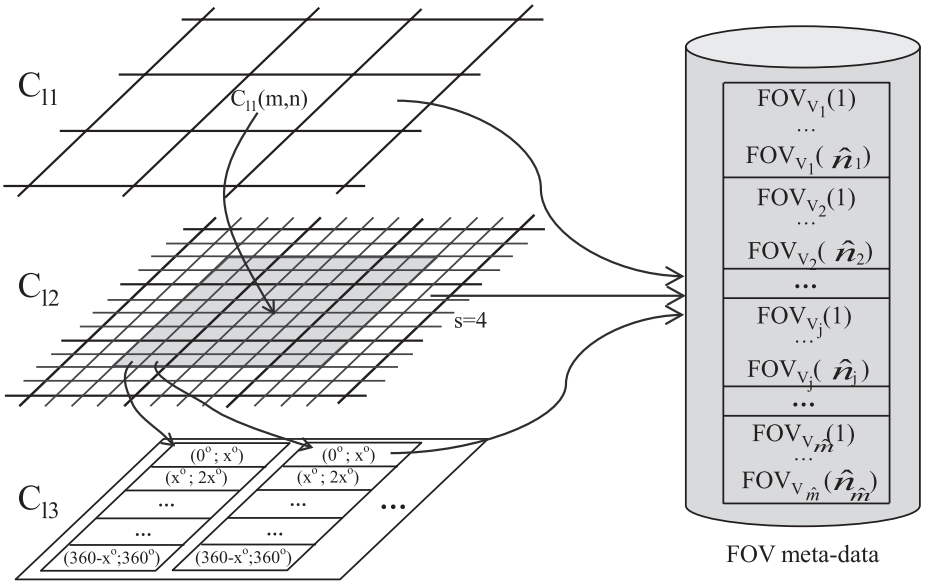
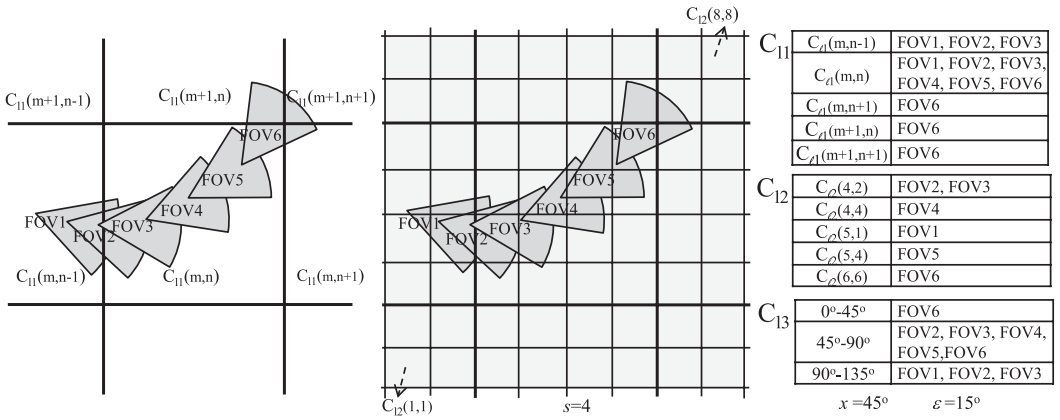


Figure 3.2: The three-level grid data structure.

The first level organizes the embedding geo-space, which covers all the regions that geo-tagged videos are recording, as a uniform coarse grid. The space is partitioned into a regular grid of $M \times N$ cells, where each grid cell is an $\delta \times \delta$ square area, and δ is a system parameter that defines the cell size of the grid. A specific cell in the first-level grid index is denoted by $C_{\ell 1}(row, column)$ (assuming the cells are ordered from the bottom left corner of the space). The 2D geographical coverages of the FOVs are indexed in this coarse grid structure. Specifically, FOVs are mapped to the grid cells that overlap with their coverage areas and each grid cell maintains the IDs of the overlapping FOVs. In Figure 3.3 (c), the set of FOVs that overlap with the first-level cells are listed in the first table.

The second-level grid index organizes the overlapping FOVs at each first-level cell



(a) First-level grid. (b) Second-level grid. (c) Index tables.

Figure 3.3: Illustration of grid-based index construction.

based on the distance between the FOV camera locations and the center of the cell. To construct the second-level grid, each $C_{\ell 1}$ cell is further divided into $s \times s$ subcells of size $\left(\frac{\delta}{s} \times \frac{\delta}{s}\right)$, where each subcell is denoted by $C_{\ell 2}(f, g)$ (see Figure 3.2). s is a system parameter and defines how fine the second-level grid index is. For each first level grid cell $C_{\ell 1}(m, n)$, we maintain the range of the second-level subcells, covering the region in and around $C_{\ell 1}(m, n)$ and containing all the FOVs that overlap with the cell $C_{\ell 1}(m, n)$. In Figure 3.2, the shaded region at $C_{\ell 2}$ shows the range of $C_{\ell 2}$ subcells corresponding to the first-level cell $C_{\ell 1}(m, n)$. Note that the FOVs whose camera locations are at most R_V away from cell $C_{\ell 1}(m, n)$, will also be included in that range. In the example shown in Figure 3.3 (b), the second-level range for $C_{\ell 1}(m, n)$ includes all subcells $C_{\ell 2}(1, 1)$ through $C_{\ell 2}(8, 8)$. While the first-level cells hold the list of the FOVs whose viewable scene areas overlap with the cell, the second-level subcells hold the list of FOVs whose camera locations are within those subcells. For example in Figure 3.3 (c), the second table lists the non-empty second-level subcells and the set of FOV IDs assigned to them. In order to retrieve the FOVs closest to a particular query point in the cell $C_{\ell 1}(m, n)$, first, the second-level cell $C_{\ell 2}(f, g)$ where the query point resides is obtained, and then the FOV IDs in and around subcell $C_{\ell 2}(f, g)$ are retrieved. The second-level index enables the efficient retrieval of closest FOVs in the execution of queries with bounded radius restriction,

e.g., a k-NVS (k Nearest Video Segments) query.

The first- and second-level grid cells hold the location and distance information only, therefore cannot fully utilize the collected sensor meta-data, such as direction. Direction can be an important cue in retrieving the most relevant video results when the videos showing the query location from a certain viewpoint are of higher interest. To support the directional queries we construct a third-level in the index structure that organizes the FOVs based on the viewing direction. The 360° angle is divided into x° intervals in clockwise direction, starting from the North (0°). We assume an error margin of $\pm\varepsilon$° around the FOV orientation angle θ°. Each FOV is assigned to one or two of the view angle intervals with which its orientation angle margin (θ°$\pm\varepsilon$°) overlaps. The value of ε can be customized based on the application. In Figure 3.3 (c), the third table lists the third-level index entries for the example video for x=45° and ε=15°.

For a video collection with about 2.95 million FOVs, the index size for the three-level index structure is measured as 1.9GB. As the dataset size gets larger the index size grows linearly. For example, for datasets with 3.9 million and 5.4 million FOVs, the index size is measured as 2.5GB and 3.3 GB, respectively. In our experiments in Section 3.4, we report the results for a dataset of 5.4 million FOVs. Next we will describe the query processing for various query types.

3.3 Query Processing

We represent the coverage of a video clip as a series of FOVs where each FOV corresponds to a spatial object. Therefore the problem of video search is transformed into finding the spatial objects in the database that satisfy the query conditions. In searching video meta-data, unlike a conventional spatial query (*i.e.,* typical point queries, range queries and kNN queries), the query may enforce additional application specific parameters. For example, it may search with a range restriction for the distance of the camera location from the query point, which is interpreted as

the *query with bounded radius*. Or the query may ask only for the videos that show the query location from a certain viewpoint, then it may restrict the FOV direction to a certain angle range around the specified viewing direction, which is interpreted as the *query with direction*. In this section we introduce several new spatial query types for searching camera viewable scenes. We will formulate these query types in Section 3.3.1. All the queries work at the FOV level. In Section 3.3.2 we will provide the details about the query processing and present the algorithms of the proposed queries.

3.3.1 Query Definitions

Let $FOV_{v_j} = \{FOV_{v_j}(i), i = 1, 2, ..., \widehat{n_j}\}$ be the set of FOV objects for video v_j and let $\mathbb{FOV}$=$\{FOV_{v_j}, j = 1, 2, ..., \widehat{m}\}$ be the set of all FOVs for a collection of $\widehat{m}$ videos. Given $\mathbb{FOV}$, a query q returns a set of video segments $\{VS_{v_j}(S_{fr}, E_{fr})\}$, where $VS_{v_j}(S_{fr}, E_{fr})$=$\{FOV_{v_j}(i),\ S_{fr} \leq i \leq E_{fr}\}$ is a segment of video v_j which includes all the FOVs between $FOV_{v_j}(S_{fr})$ and $FOV_{v_j}(E_{fr})$, where i stands for the ith frame, and $<S_{fr},\ E_{fr}>$ denotes the starting and ending frame of a video segment respectively.

Definition 1 *Point Query with Bounded Radius* (PQ-R):

Given a query point q in geo-space and a radius range from MIN_R to MAX_R, the PQ-R query retrieves all video segments that overlap with q and whose camera locations are at least MIN_R and at most MAX_R away from q, *i.e.,*

PQ-R(q, MIN_R, MAX_R) :

$q \times \mathbb{FOV} \rightarrow \{VS_{v_j}(S_{fr}, E_{fr}),$ where $\forall j\ \forall i\ S_{fr} \leq i \leq E_{fr}$ such that $FOV_{v_j}(i) \cap q \neq \varnothing$,

and $MIN_R \leq dist(P(FOV_{v_j}(i)), q) \leq MAX_R\}$,

where P returns the camera location of an FOV and function *dist* used here calculates the distance between P and q.

Definition 2 *Point Query with Direction*(PQ-D):

Given a query point q in geo-space and viewing direction β, the PQ-D query retrieves all FOVs that overlap with q and that were taken when the camera was pointing towards β with respect to the North. The PQ-D query exploits the camera's bearing to retrieve the video frames that show the query point from a particular viewing direction. Since slight variations in the viewing direction does not significantly alter the human perception, using only a precise direction value β may not be practical in video search. Therefore a small angle margin ε around the query view direction is introduced, and the query searches for the video segments whose directions are between $\beta - \varepsilon$ and $\beta + \varepsilon$.

PQ-D(q, β):

$$q \times \mathbb{FOV} \rightarrow \big\{VS_{v_j}(S_{fr}, E_{fr}),\ \text{where } \forall j\ \forall i\ S_{fr} \leq i \leq E_{fr} \text{ such that } FOV_{v_j}(i) \cap q \neq \varnothing,$$
$$\text{and } \beta - \varepsilon \leq D(FOV_{v_j}(i)) \leq \beta + \varepsilon\big\},$$

where D returns an FOV's camera direction angle with respect to North.

Definition 3 *Range Query with Bounded Radius* (RQ-R):

Given a rectangular region q_r in geo-space and a radius range from MIN_R to MAX_R, the RQ-R query retrieves all video segments that overlap with q_r and whose camera locations are at least MIN_R and at most MAX_R away from the border of q_r. RQ-R definition is very similar to PQ-R query, therefore we omit further details here.

Definition 4 *Range Query with Direction* (RQ-D):

Given a rectangular region q_r in geo-space and a viewing direction β, the RQ-D query retrieves all video segments that overlap with region q_r and that show it with direction interval between $\beta - \varepsilon$ and $\beta + \varepsilon$. We also omit the details for RQ-D query, as the definition is similar to PQ-D query.

Definition 5 *k-Nearest Video Segments Query* (k-NVS):

Given a query point q in geo-space, the k-NVS query retrieves the closest k video segments that show the query point q. The returned video segments are ordered from closest to the farthest based on their distance to q.

k-NVS(q,k):

$$q \times \mathbb{FOV} \rightarrow \{(VS_{v_j}(S_{fr_1}, E_{fr_1}), .., VS_{v_j}(S_{fr_k}, E_{fr_k}))$$
$$\text{where } \forall S_{fr_t}, E_{fr_t}\ (t = 1, .., k),$$
$$\text{and } \forall j \forall i\ S_{fr_t} \leq i \leq E_{fr_t}, \text{ such that } FOV_{v_j}(i) \cap q \neq \varnothing$$
$$dist(VS_{v_j}(S_{fr_t}, E_{fr_t}), q) \leq dist(VS_{v_j}(S_{fr_{t+1}}, E_{fr_{t+1}}), q)\Big\},$$

The function *dist* used here calculates the minimum distance between the camera locations of a video segment and the query point.

Definition 6 *k-Nearest Video Segments Query with bounded Radius* (k-NVS-R):
The k-NVS-R query is similar to the k-NVS and PQ-R queries. Given a query point q in geo-space and a radius range from MIN_R to MAX_R, the k-NVS-R query retrieves the closest k video segments that show the query point q from a distance between MIN_R to MAX_R. Similar to the k-NVS query, the returned video segments are ordered from the closest to the farthest based on their distance to q.

Definition 7 *k-Nearest Video Segments Query with Direction* (k-NVS-D):
The k-NVS-D query is also similar to the k-NVS and PQ-D queries. Given a query point q in geo-space and a viewing direction β, the k-NVS-D query retrieves the closest k video segments that show the query point q with the direction β.

3.3.2 Algorithm Design

The query processing is performed in two major steps. In the first step, the FOVs (*i.e.,* the video frames) that satisfy the query conditions in the set $\mathbb{FOV}$ are retrieved.

The returned FOVs are grouped according to the video files that they belong to. And in the second step, the groups of adjacent FOVs from the same videos are post processed to retrieve as the video segments in the query results. We argue that, the length of the resulting video segments should be larger than a certain threshold length for visual usefulness. For some query types, such as RQ-R and RQ-D queries, the number of consecutive FOVs that match the query requirements is usually large enough to form a reasonable length video segment, therefore this post processing step is straightforward. However, for more restricted queries such as k-NVS query, often the formed video segments may contain only a few FOVs. Therefore this post processing step may add additional video frames to the video segments according to the requirements of the search application.

Next we will further elaborate on these two major steps of the query processing. In Section 3.3.2.1, we will describe the retrieval of the FOVs that match the query requirements for each of the proposed query types. In Section 3.3.2.2, we will describe a simple approach for the post processing of the retrieved FOVs to form the resulting video segments.

3.3.2.1 Query Processing: Retrieval of Matching FOVs

In this section, we will present the algorithms for processing the proposed query types on our three-level grid structure. We will describe these query processing procedures under three groups: Point query (PQ-R and PQ-D), Range query (RQ-R and RQ-D) and k-NVS query (k-NVS and k-NVS-D). Within each query group, we will further elaborate on the direction and bounded radius queries.

We retrieve the FOVs that match the query requirements in two steps: a *filter step* followed by a *refinement step*. First, in the filter step, we search the three-level index structure starting from the first level and then moving down to the second and third level, if needed. The set of FOVs resulting set from the filter step are referred as the *candidate set*. In the refinement step, an exhaustive method is applied to

check whether an FOV actually satisfies the query conditions.

Point Query The video segments, that show a certain object of interest at a specific location, can be retrieved through the point query. When the object size is small, it would be preferred to retrieve the close-up views of the object, with a reasonable size for better visual perception. The PQ-R query searches the video frames with a certain radius range restriction for the distance of the camera locations from the query point, according to the required level of details in the video. Additionally, the camera viewing direction when the query object appears in the video can be an important factor for the image perception of the observer. For example, an object's images from a certain viewing direction (*e.g.,* the frontal view, when the object is viewed from the North) can be of higher importance. The PQ-D query can exploit the collected camera directions for querying video segments when the camera is pointing towards the requested direction (*e.g.,* North).

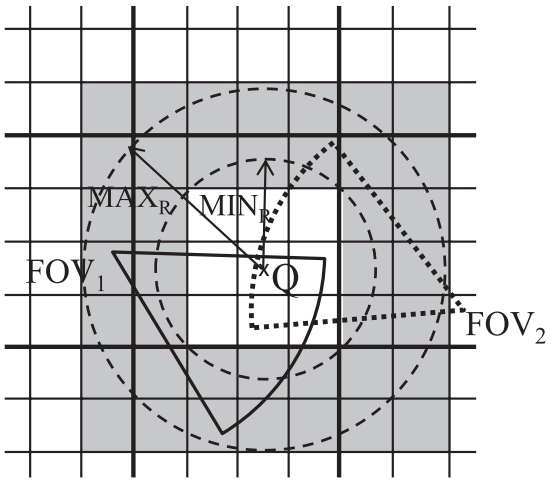


Figure 3.4: Illustration of the function *applyRadius*.

Algorithm 1 formalizes the query execution for point queries PQ-R and PQ-D. When processing the point query, we first calculate the first-level cell ID $C_{\ell 1}$ where the query point is located. For a typical point query (PQ), the candidate FOVs would include all FOVs indexed at the cell $C_{\ell 1}$. For the *Point Query with bounded*

Algorithm 1: Point query with bounded radius (PQ-R) and direction (PQ-D).

Input: query type: *q_type* (PQ-R, or PQ-D), query point: $q\langle lat, lng\rangle$,
(for PQ-R) min and max radius: MIN_R, MAX_R,
(for PQ-D) viewing direction : β
Output: vector *segments* $\langle v_j, VS(S_{fr}, E_{fr})\rangle$

```
C_ℓ1 = getCellID(q);                                   // First-level cell
/* Point Query with bounded Radius                                    */
if q_type is PQ-R then
    C_ℓ2 = getSubCellID(q);                            // Second-level cell
    subCellsInR = applyRadius(q, C_ℓ2, MIN_R, MAX_R);
    candidateFOVs = fetchData(subCellsInR);
/* Point Query with Direction                                         */
if q_type is PQ-D then
    C_ℓ3 = getDirCellID(C_ℓ1,β,ε);                     // Third-level cell
    candidateFOVs = fetchData(C_ℓ3);
res = refinementStep(candidateFOVs);
segments = getVideoSeg(res,q_type);
return segments
```

Radius (PQ-R), we additionally apply the distance condition given by the radius range (MIN_R, MAX_R). The function *applyRadius* reduces the search area for the candidate FOVs in the second-level index by eliminating the subcells outside of the radius range (see Algorithm 2). In function *applyRadius*, we first retrieve the $C_{\ell 2}$ subcell where query point q is located. Then we find out all the second-level subcells around $C_{\ell 2}$, which are within distance range MIN_R to MAX_R from the query point. For example, in Figure 3.4, according to the minimum (MIN_R) and maximum (MAX_R) distance conditions, only the FOVs locating between the two dot circles will be returned. Since this function works on subcell level, it takes all the subcells that overlap with the region between two circles (*i.e.,* the shadow region) into account. In this example, both video frames $FOV1$ and $FOV2$ overlap with q. However, since the location of $FOV2$ is outside of the shadow region, it won't be returned by the function *applyRadius*, and therefore $FOV2$ will not be included in the candidate set. For the *Point Query with Direction* (PQ-D), we check the third–level index cell to find cells that cover the query angle range given by (β-ε, β+ε). We return the FOVs indexed in the cells $\{C_{\ell 3}(h_1), ..., C_{\ell 3}(h_2)\}$ where β-ε falls

into the angle range of $C_{\ell 3}(h_1)$ and $\beta+\varepsilon$ falls into the angle range of $C_{\ell 3}(h_2)$. As an example, let us assume that the 360° viewing angle range is divided into $x = 45°$ intervals in the third-level index. When $\beta = 0°$ (*i.e.,* North) and $\varepsilon = 15°$, we would retrieve the FOVs in the third-level cells $C_{\ell 3}(7)$ and $C_{\ell 3}(0)$ as the candidate FOVs.

Algorithm 2: applyRadius()

Input: query point: $q\langle lat, lng\rangle$,
Minimum and maximum bounded radius: MIN_R, MAX_R,
first–level cell: $C_{\ell 1}$, second–level cell: $C_{\ell 2}$
Output: set of second-level cells: *checkRadius*
distClose = compMinDist(q, $C_{\ell 2}$);
while *distClose* $\leq MAX_R$ **do**
    `/* Find out the minimum distance between the` q `and` $C_{\ell 2}$ `*/`
    distFar = compMaxDist(q, $C_{\ell 2}$);
    if *distFar* $\geq MIN_R$ **then**
        checkRadius.add(getCellsAtDist(q,*distClose*));
    distClose += $GRIDSIZE/s$;
return *checkRadius*

After the candidate FOVs are retrieved, we run the refinement step (through the function *refinementStep*) to get the actually matching FOVs (See Algorithm. 3). For each FOV in the candidate set we check whether the FOV overlaps with q. In the refinement step of PQ-R query, we also check whether the distance between the camera location and the query point is within the radius range (MIN_R, MAX_R). While for PQ-D query, we check whether the viewing direction of the camera falls into the angle range (β-ε, β+ε). These FOVs, along with their video file IDs (v_j) are stored in the vector *res*.

The last step in the point query processing is the generating of resulting video segments from the retrieved FOVs. The function *getSegments* organizes the group of consecutive FOVs from the same video as video segments $VS_{v_j}(S_{fr}, E_{fr})$. The details of the *getSegments* function is explained in Section 3.3.2.2.

Range Query When the search application asks for the videos that show a large region in geo-space, rather than a point location, it may issue a range query. The

Algorithm 3: refinementStep()

```
Input: query type: q_type (PQ-R, or PQ-D)
FOV candidate set: vector candidateFOVs,
(for PQ-R) min and max radius: MIN_R, MAX_R
Output: vector res ⟨v_j, FOV.id⟩
for all the FOVs in the candidateFOVs do
    if q_type is PQ-R then
        distP2P = dist(q, FOV.P);   /* distance between two points */
        if distP2P ≥ MIN_R AND distP2P ≤ MAX_R then
            if pointInFOV(q, FOV) then
                res.push( ⟨FOV.v_j, FOV.id⟩ );
    if q_type is PQ-D then
        if FOV.θ ≥ β − ε AND FOV.θ ≤ β + ε then
            if pointInFOV(q, FOV) then
                res.push( ⟨FOV.v_j, FOV.id⟩ );
```

queried region is estimated with a bounding rectangle. Similar to the PQ-R query, the closeness to the query region, therefore the level of details in the video, can be customized through the RQ-R query. Additionally, the RQ-D query retrieves videos of the query region from different view points.

In the range query processing, a naive approach is to access only to the first-level index to get the candidate FOVs. Since the first-level grid cells are larger, each FOV appears only in a few $C_{\ell 1}$ cells. When the overlap area between the $C_{\ell 1}$ cell and the query rectangle is large, using the first-level index is more efficient, since the duplicate FOV IDs in the candidate set is minimized. On the other hand, if the query rectangle overlaps with a small percentage of the $C_{\ell 1}$ cell, the retrieved candidate set will have many false positives due to FOVs covering parts of the $C_{\ell 1}$ cell but not the query region. Therefore, in our range query processing algorithm, we use a hybrid approach where we try to cover the query region with a mixture of $C_{\ell 1}$ and $C_{\ell 2}$ cells. We try to minimize the uncovered regions in cells (*i.e.,* minimizing the false positives) and at the same time, we also minimize the duplicate FOV IDs in the candidate set, by using as many $C_{\ell 1}$ cells as possible. The goal is to reduce

Algorithm 4: Range query with bounded radius (RQ-R) and direction (RQ-D).

Input: query type: q_type (RQ-R, or RQ-D)
query rectangle: $q_r\langle lat1, lng1; lat2, lng2\rangle$,
(for RQ-R) min and max radius: MIN_R, MAX_R,
(for RQ-D) viewing direction : β
Output: vector *segments* $\langle v_j, VS(S_{fr}, E_{fr})\rangle$
$C_{\ell 1}$ = getCellID(q_r)
`/* Range Query with bounded Radius */`
if *q_type is RQ-R* **then**
  cellsInR = applyRadius(q_r, $C_{\ell 1}$, MIN_R, MAX_R);
  for *each cell* $C_{\ell 1}(m,n)$ *in cellsInR* **do**
    overlapArea = compOverlap($C_{\ell 1}(m,n)$,q_r); `// Compute the overlap area`
    if *overlapArea* $\geq \phi$ **then**
      candidateFOVs.append(fetchData($C_{\ell 1}(m,n)$));
    else
      subCellsInR = applyRadius(*overlapArea*,$C_{\ell 2}$,MIN_R,MAX_R);
      candidateFOVs.append(fetchData(*subCellsInR*));
`/* Range Query with Direction */`
if *q_type is RQ-D* **then**
  for *each cell* $C_{\ell 1}(m,n)$ *in cellsInR* **do**
    overlapArea = compOverlap($C_{\ell 1}(m,n)$,q_r);
    if *overlapArea* $\geq \phi$ **then**
      $C_{\ell 3}$ = getDirCellID($C_{\ell 1}(m,n)$,β,ε);
    else
      subCells = applyRadius(*overlapArea*,$C_{\ell 2}$,0,R_V);
      $C_{\ell 3}$ = getDirCellID(*subCells*,β,ε);
    candidateFOVs.append(fetchData($C_{\ell 3}$));
res = refinementStep(*candidateFOVs*);
segments = getVideoSeg(*res*,*q_type*);
return *segments*

the size of the candidate set, so that the time required to process and sort the FOVs in the refinement step is minimized.

Algorithm 4 formalizes the query execution for the range queries RQ-R and RQ-D. In Algorithm 4 we first find out which cells will be accessed from the first-level and second-level indexes. Among the $C_{\ell 1}$ cells that overlap with q_r, we choose the cells whose overlap areas are larger than a certain threshold value ϕ (*e.g.,* the overlap

area is 40% of that of the $C_{\ell 1}$ cell). If the overlap area is less than ϕ, we cover the overlap region with the $C_{\ell 2}$ subcells. Recall that the second-level subcells hold the list of the FOVs whose camera locations are within those subcells. Therefore, to retrieve the candidate FOVs from a $C_{\ell 2}(m, n)$ subcell, we need to search for the neighboring subcells around it, and find out the FOVs in those subcells which overlap with the $C_{\ell 2}(m, n)$. After finding out the cells and subcells that we would retrieve the candidate FOVs from, the rest of the query processing is similar to PQ-R and PQ-D queries.

k-NVS Query Typical kNN queries consider only the distance between a query point and the objects in the database. In our geo-tagged video search system, we consider not only the distance between the query point and camera location in the database, but also the visibility of the query point from the camera location. Here, we propose the *k-Nearest Video Segments* query as, "For a given point q in geo-space, find the k nearest video segments that overlap with q." Taking Figure 3.5 as an example, the camera locations of the video segment V_1 are closer to the query point q than those of V_2. Due to the camera's location and viewing direction, the FOVs of V_1 cannot cover q while the FOVs of V_2 can. In typical kNN queries, V_1 will be selected before V_2 because V_1 is closer to q. However, in the k-NVS query, V_2 will be selected as the nearest neighbor instead of V_1 because of the visibility. The k-NVS query can be utilized in various video search applications to continuously retrieve the most related videos that show a frequently updated query point. Additional radius range and viewing direction requirements can be added to the query through the k-NVS-R and k-NVS-D queries.

Algorithm 5 formulates the k-NVS query processing. We first retrieve the $C_{\ell 2}$ cell where the query point is located and, similar to the PQ-R query, we find out the neighboring subcells around $C_{\ell 2}$ from which the FOVs can see q. For the k-NVS-R query, the search range around the $C_{\ell 2}$ cell is (MIN_R, MAX_R) whereas for the k-NVS and k-NVS-D queries search range is (0, R_V). For the k-NVS queries, we

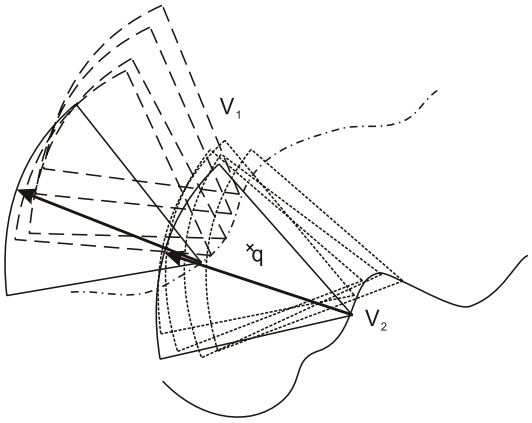


Figure 3.5: Illustration of k-NVS query. Although camera locations of V_1 are closer to the query point q than those of V_2, V_2 is selected as the nearest video segment, since V_1 cannot capture q.

need to return only the closest k video segments. Therefore, in order to find the candidate FOVs, we gradually search the neighboring subcells in the search range, starting with the closest subcells. As shown in Algorithm 5, we first retrieve the candidate FOVs in the subcells closest to $C_{\ell 2}$ (within distance 0 or MIN_R). And at each round we increase the search distance by δ/s and retrieve the FOVs in the next group of cells within the enlarged distance (δ/s is the size of a second-level subcell). We apply the refinement step on these candidate FOVs and store them in priority queue, in which the FOVs are sorted based on their distance to q in ascending order. The refinement steps for the k-NVS-R and k-NVS-D queries are similar to PQ-R and PQ-D queries. After each round of candidate retrieval, the candidate FOVs are organized as videos segments, *i.e.,* the consecutive FOVs from the same video file are grouped together. The search for candidate FOVs ends either when the number of video segments reaches k or when there are no more subcells that need to be checked. The output of the algorithm is the list of the retrieved video segments, ordered from closest to the farthest.

3.3.2.2 Query Processing: Returning Video Segments

As explained in Section 3.3.2.1, in query processing after retrieving the FOVs that satisfy the query requirements, the groups of adjacent FOVs from the same videos

Algorithm 5: k-Nearest Video Segments Queries: k-NVS, k-NVS-R, and k-NVS-D

Input: query point: $q\langle lat, lng\rangle$, number of output video segments: k,
(for k-NVS-R) min and max radius: MIN_R, MAX_R,
(for k-NVS-D) viewing direction : β,
Output: vector *segments* $\langle VS_{v_j}(s_t, e_t)\rangle$
$C_{\ell 1}$ = getCellID(q), $C_{\ell 2}$ = $getSubCellID(q)$;
priority_queue $sortedFOVs$ $\langle$ $FOVID$, distance to q $\rangle$ = $\emptyset$;
if *q_type is k-NVS-R* **then**
    $subCellsInR$=applyRadius(q,$C_{\ell 2}$,MIN_R,MAX_R);
else
    $subCellsInR$ = applyRadius(q, $C_{\ell 2}$, 0, R_V);
i=0; distClose=δ/s;
while *not enough FOVs AND nextSubCells= getNeighbors(q,subCellsInR,i++) is not empty* **do**
    $candidateFOVs$ = fetchData($nextSubCells$);
    for *all the FOVs in the candidateFOVs* **do**
        $distP2P$ = dist(q, FOV);
        if *q_type is k-NVS* **then**
            if *pointInFOV(q, FOV)* **then**
                $sortedFOVs$.push($\langle FOVID, distP2P\rangle$);
        if *q_type is k-NVS-R* **then**
            if $distP2P \geq MIN_R$ *AND* $distP2P \leq MAX_R$ *AND pointInFOV(q, FOV)* **then**
                $sortedFOVs$.push($< FOVID, distP2P >$);
        if *q_type is k-NVS-D* **then**
            if $FOV.\theta \geq \beta - \varepsilon$ *AND* $FOV.\theta \geq \beta + \varepsilon$ **then**
                $sortedFOVs$.push($< FOVID, distP2P >$);
    while $sortedFOVs.top() \leq distClose$ *AND* $numsegments < k$ **do**
        $topFOV$ = $sortedFOVs$.pop();
        if *isNewSegment(topFOV,res)* **then**
            $numsegments$++;
        res.push($topFOV$);
    i++; $distClose += \delta/s$;
$segments$ = getVideoSeg(res,q_type);
return $segments$

are returned as the resulting video segments. The length of the returned segments may vary extensively for different query types. For example for the range query, when the query region expands to a large area, the number of consecutive FOVs

that overlap with the query region is usually large. However for more selective queries, such as k-NVS query, the length of an individual segment can be as short as a few seconds. In Table 3.1, we report the number of FOVs returned from the k-NVS query and the number of video segments that these FOVs form for different values of k. The average segment length for k=20 is around 3 seconds, with a maximum segment length of 20 seconds. As the k value increases, the segment lengths also get longer. Practically, for visual clarity, the length of the resulting video segments should be larger than a certain threshold length. Depending on the requirements of the video search application, the query processing unit should customize the creation of the returned segments.

Table 3.1: Statistics for k-NVS queries with different k values.

k	# of FOVs	# of Segments	Segment Max Length
20	109,847	35,391	20
50	212,746	56,664	50
100	291,957	72,179	71
150	318,504	77,096	89
200	326,110	78,523	89
300	327,541	78,796	89

In our current implementation, for the point and range queries, the returned FOVs are post-processed to find out the consecutive FOVs that form the segments. If two separate segments of the same video file are only a few seconds apart, they are merged and returned as a single segment. For the k-NVS query, the video segments are formed simultaneously as the closest FOVs are retrieved. For each video segment the video ID, the starting and ending FOV IDs and the segments distance to the query point are maintained, *i.e.*, $\langle v_j, S_{fr_t}, E_{fr_t}, dist \rangle$. When the next closest FOV is retrieved, if it is adjacent to one of the existing segments it is merged with it, otherwise a new segment is formed. The S_{fr_t}, E_{fr_t}, and $dist$ values are updated accordingly. For example, in our current configuration of the experiments, we set that the returned segments should be at least 20 seconds long. Therefore the short

segments are expanded to 20 seconds. The segment's starting and ending offsets are adjusted so that the *dist* value for the segment is minimized.

3.4 Experimental Evaluation

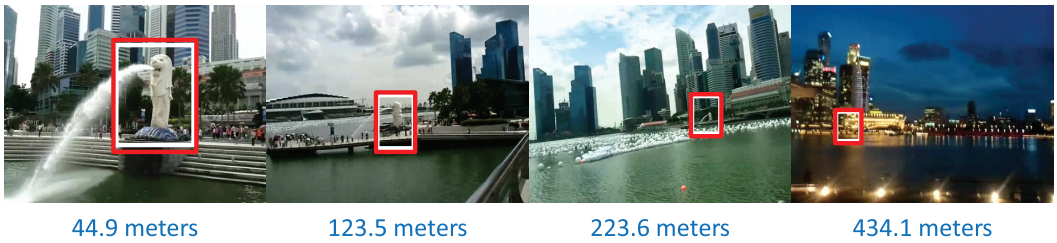
In this section, we elaborate on the experiments carried out on ***Dataset1*** and ***Dataset3***.

3.4.1 Experiments with Dataset1

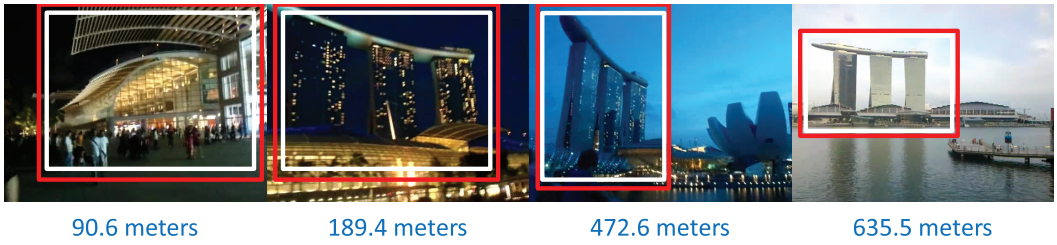
To show the impact of the bounded radius and direction restriction on the video searching results, we presented two representative query results on two landmarks: Q1 targets at the Merlion (a small statue) while Q2 targets the Marina Bay Sands (a tall and wide building).

3.4.1.1 Importance of Bounded Radius

Figure 3.6 shows the sampling frames from the resulting videos that answering Q1 and Q2 with various distances from the query locations. All the frames shown in Figure 3.6(a) actually capture Q1 in the scene, but only the first one from a closed position less than 50 meters away from Q1 displays a clear view. Due to the large distance, although Q1 appears in the other three frames, it is meaningless to end-users who want to see the the Merlion. The situation is quite different when processing Q2 (shown in Figure 3.6(b)). The frames captured from closed positions only show parts of the building, while users can have a panoramic view of the whole building from far away. Consequently, typical spatial queries on video search might sometimes not satisfy users' demands, and it is helpful to have spatial queries with bounded radius. For example, displaying small objects from a closed position, while displaying large buildings from a far-away location can help to show meaningful videos to users.



(a) Sampling frames from Q1.



(b) Sampling frames from Q2.

Figure 3.6: Sampling frames from the video searching results with various distances. The numbers in blue show the distance between the camera location and the queried building.

3.4.1.2 Importance of Direction

The sampling frames in Figure 3.7 are extracted from the same video and all of them capture the Merlion from different directions. The first two frames record Q1 from the best place while the last one records it from the back, which is not desired by users. Therefore, direction restriction is also an important factor for video searching on displaying best results.

Figure 3.7: Sampling frames from the video searching results with various directions on Q1.

3.4.2 Experiments with Dataset3

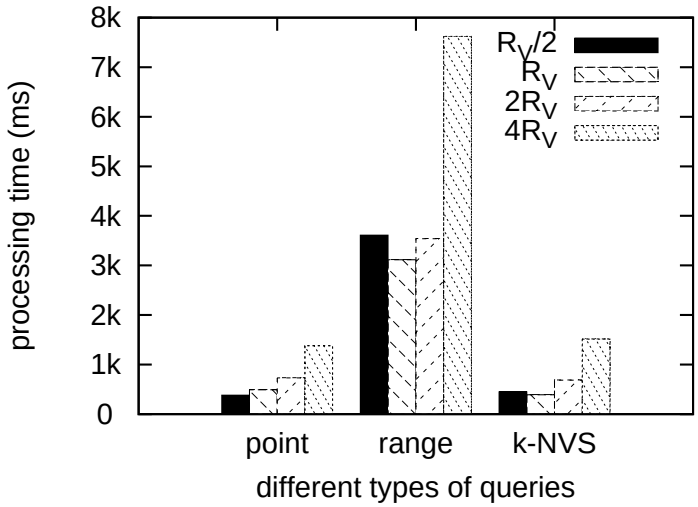
In this section, we use ***Dataset3*** to test the performance of the index structure on a large-scale dataset.

3.4.2.1 Experimental Settings

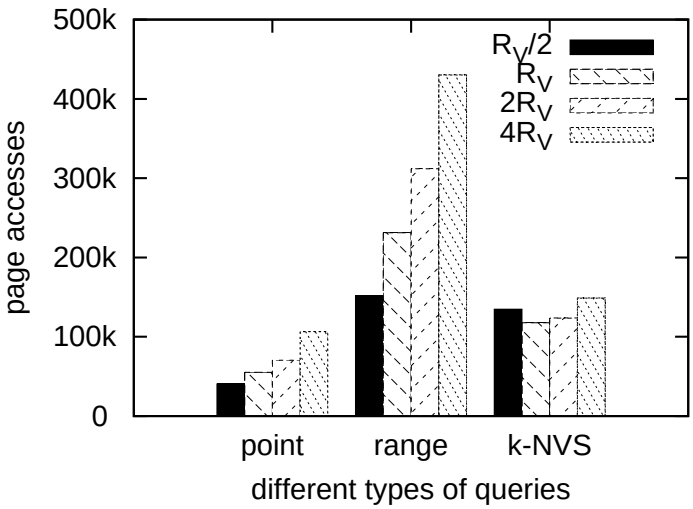
For all the experiments we constructed a local MySQL database and stored all the FOV meta-data, as well as the index structure tables. All the experiments were conducted on a server with two quad core Intel(R) Xeon(R) $X5450$ 3.0GHz CPUs and 16GB memory running under Linux RedHat 2.6.18. All the comparisons used in the experiments are based the geo-coordinates (latitude and longitude). The experiment results reported here show the cumulative number of FOVs returned for $10,000$ randomly generated queries within the experiment region. In our experiments, we mainly measure the Processing Time (PT for short) and the number of Page Access (PA for short). The PT includes the total amount of time for searching for the candidate set through the index structure in the filter step and the time for using the exhaustive method to process overlap calculation in the refinement step. We assume that even if the index structure was in memory, when we access to it, we count the PA as it is on disk. Additionally, we set the page cache size as one page large. Therefore, when there is no page hit in the cache, the PA will be increased by one. This also helps to analyze the performance if the index structure is disk-based instead of memory-based. In our experiments, we try to fully utilize the space inside each one page by storing as many nodes as possible for both the grid-based approach and R-tree. The page has empty space only when there exists no exact match between the page space and the node size.

In the next two experiments, we process the typical queries without any distance or direction restriction as preliminary experiments to set the basic parameters: the value of the grid size δ and the overlap threshold ϕ. In the following experiments, if not specifying, the default value of k is 20, and query rectangle size is $250m \times 250m$.

When generating the moving objects, the maximum viewable distance (R_V) of the camera is set as $250m$. As shown in Figure 3.8, grid with size equalling to $R_V/2$ or R_V performs better than larger sizes. The performance of grid-based index structure with size of R_V is better in some cases while worse in others compared to that of $R_V/2$. Since our structure is mainly designed for k-NVS query, we set δ equalling to R_V.



(a) Processing time.

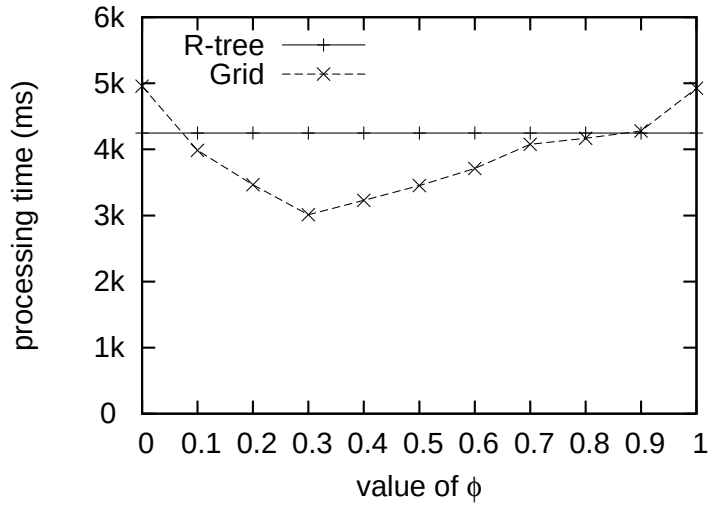


(b) Page accesses.

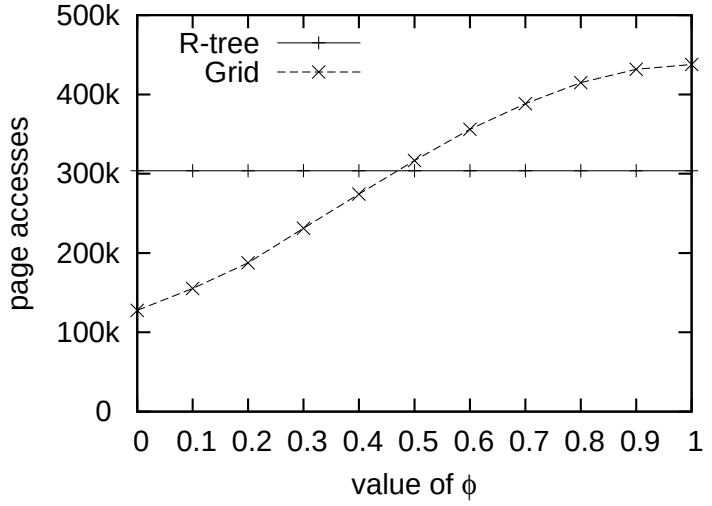
Figure 3.8: Effect of grid size.

As well as the grid size, the overlap threshold ϕ for range query also affects the

performance of the grid-based index structure. As presented in section 3.2, both the first-($C_{\ell 1}$) and second-level($C_{\ell 2}$) indices are loaded into memory. To set the value of ϕ, we ran a series of typical range queries without distance and direction conditions. As shown in Figure 3.9, the PA of grid-based index structure is smaller than that of R-tree when ϕ is smaller than 40%. Moreover, the grid approach is faster than R-tree for most of the cases, and we achieve the fastest performance at value of 30%. Consequently, ϕ is chosen as 30%. The parameters used in the experiment are summarized in Table 3.2.



(a) Processing time.



(b) Page accesses.

Figure 3.9: Effect of the overlap threshold ϕ on range query performance.

Table 3.2: Experiment Parameters and Their Values

Parameter	Value
Page Size	4,096
Cache Size	4,096
Non-Leaf Node Size(R-tree)	64
Leaf Node Size (R-tree)	36
Non-Leaf Node Size (R-tree with viewing direction as a dimension)	80
Leaf Node Size (R-tree with viewing direction as a dimension)	52
$C_{\ell 1}$ Node Size	68
$C_{\ell 2}$ Node Size	36
$C_{\ell 3}$ Node Size	4
FOV Meta-data Size	32

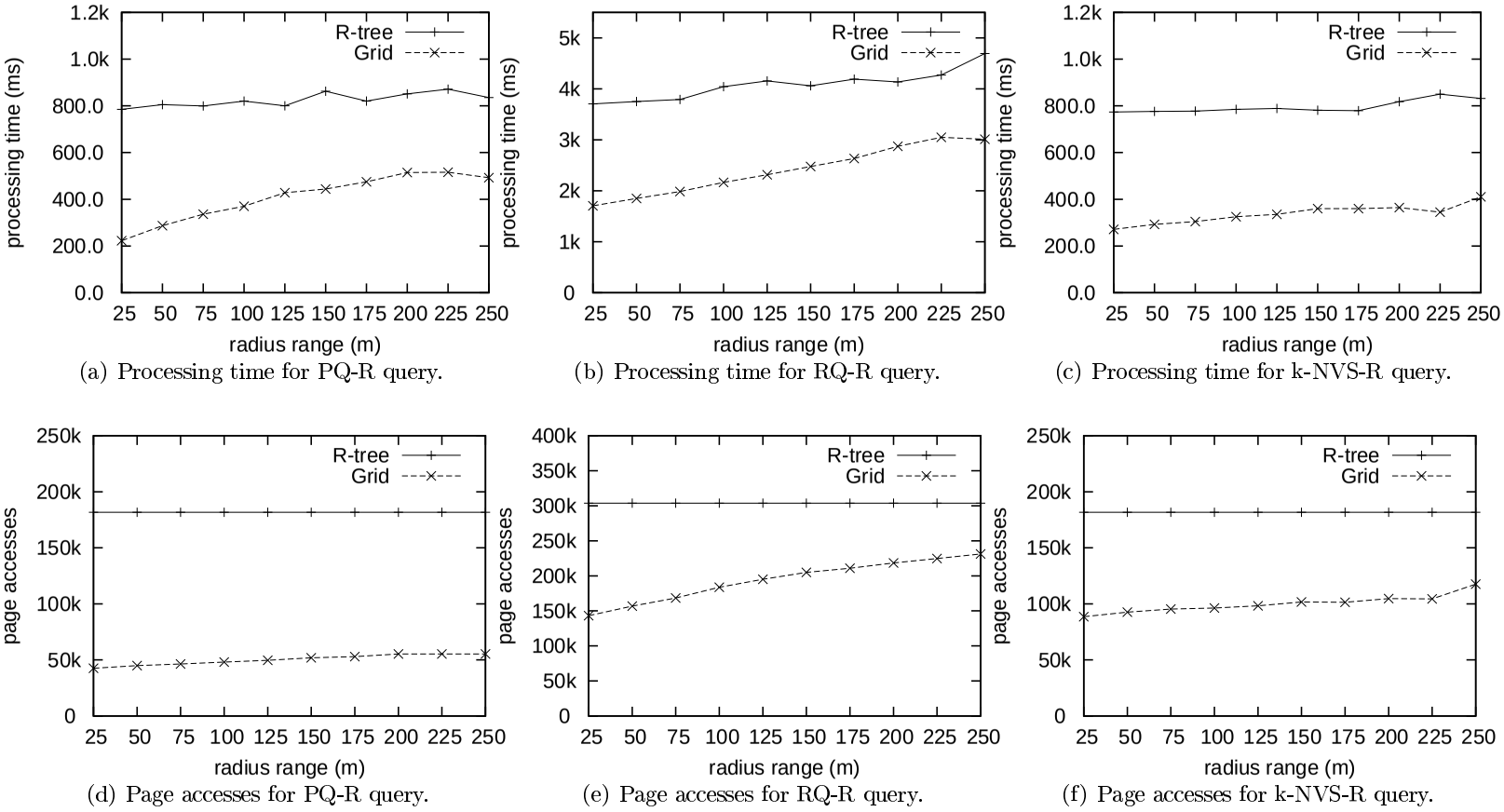
3.4.2.2 Comparison

The R-tree is one of the basic index structures for spatial data which is widely used. In our experiments, we insert the Minimum Bounding Rectangle (MBR for short) of all FOVs into R-tree and process all types of queries based on the R-tree [Green 2010] implemented by Melinda Green for comparison. To the best of our knowledge, this implementation achieves the best performance compared to others. We use Equation 3.1 to calculate the MBR of an FOV with geo-coordinates. The parameter σ_x and σ_y denotes the factor of converting distance to geo-coordinate difference in the x-axis or y-axis directions, respectively. The query procedure is to search for all the FOVs whose MBRs overlap with the query input in the filter step and hence to use the exhaustive method to calculate the actual overlap in the refinement step. Consequently, some of the parameters (*e.g.*, value of k for k-NVS query, distance condition, etc.) have no effect on the PA for R-tree.

$$\begin{aligned} MBR.left &= min(lng, lng \pm R_V \times \sin(\theta \pm \alpha/2)/\sigma_x) \\ MBR.right &= max(lng, lng \pm R_V \times \sin(\theta \pm \alpha/2)/\sigma_x) \\ MBR.bottom &= min(lat, lat \pm R_V \times \cos(\theta \pm \alpha/2)/\sigma_y) \\ MBR.ceil &= max(lat, lat \pm R_V \times \cos(\theta \pm \alpha/2)/\sigma_y) \end{aligned} \tag{3.1}$$

Effect of distance condition We study the effect of the distance condition by varying the radius range from $25m$ to $250m$. For each one of the radius range, we start from the minimum distance condition MIN_R equalling to $0m$ until the maximum distance condition MAX_R reaching $250m$. For example, when the radius range is equal to $25m$, the value of the tuple $\langle MIN_R, MAX_R \rangle$ can be one of the followings: $\{\langle 0, 25\rangle, \langle 25, 50\rangle, \langle 50, 75\rangle, ... \langle 200, 225\rangle, \langle 225, 250\rangle\}$. While the radius range is equal to $225m$, the value of $\langle MIN_R, MAX_R \rangle$ can only be either $\langle 0, 225\rangle$ or $\langle 25, 250\rangle$. The results shown in Figure 3.10 are the averages of the different radius ranges. Since the R_V of an FOV is set as $250m$ when generating the synthetic data, the last point with radius range of $250m$ is the result of processing queries without distance condition. Figures 3.10 (a), (b) and (c) illustrate the PT of PQ-R query, RQ-R query and k-NVS-R query respectively, while Figures 3.10 (d), (e) and (f) illustrate the PA for each type of query. In general, the performance of our grid-based index structure works better than R-tree on both the PT and the PA. Figures 3.10 (a) and (b) show that the PT for radius range of $250m$ is a little shorter than that of $225m$. The reason is that all the subcells in the second-level index $C_{\ell 2}$ needs to be checked for large radius range and this costs extra PT compared to queries without distance condition. As shown in Figures 3.10 (d), (e) and (f), the PA using the R-tree remains the same because the R-tree finds out all the FOVs whose MBRs overlap with the query in the filter step, regardless of the radius range. It cannot prune unnecessary search based on the radius range. The PA of the grid-based structure grows as the radius range becomes larger but is still smaller than that of the R-tree even when the radius range reaches the largest number.

Effect of direction condition We proceed to evaluate the efficiency of our grid-based index structure with directional queries. In this experiment, the query datasets are the same as those used in queries without direction condition, except the additional viewing direction constraint. As presented in Table 3.2, the angle margin ε in this experiment is $15°$. The $2D$ and $3D$ R-trees used in Figure 3.11

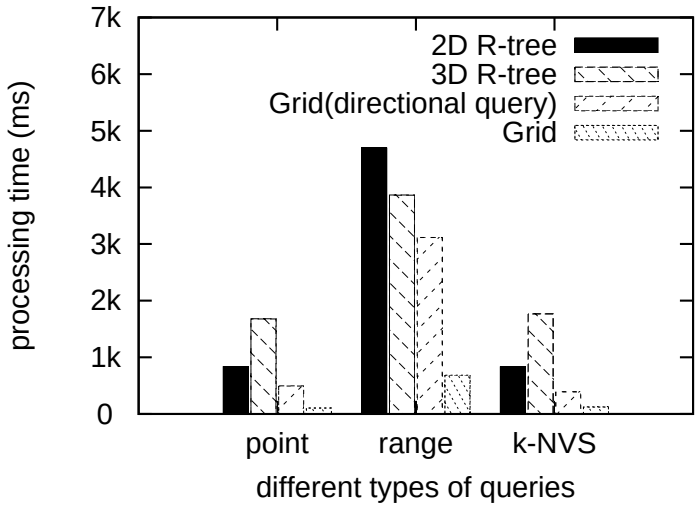


(a) Processing time for PQ-R query.
(b) Processing time for RQ-R query.
(c) Processing time for k-NVS-R query.
(d) Page accesses for PQ-R query.
(e) Page accesses for RQ-R query.
(f) Page accesses for k-NVS-R query.

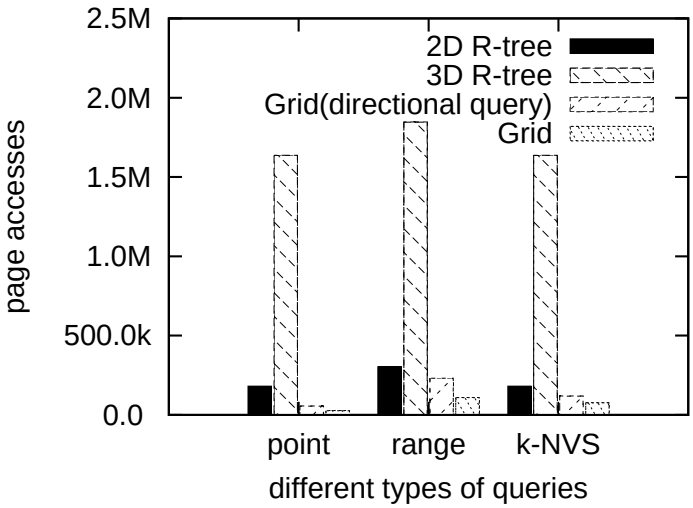
Figure 3.10: Effect of distance condition.

denote the R-tree for processing queries without direction condition and queries with direction condition, respectively. Overall, the PT of the grid-based structure is smaller than that of the R-tree processing the same query types. More specially, Figure 3.11 (a) shows that, in the processing of PQ-D query and k-NVS-D query, the PT of the R-tree is almost two times of that without direction condition. The reason is that searching one more dimension in the R-tree slows down the performance of the R-tree. The situation is different for RQ-D query because of less number of candidates obtained from the filter step so that the refinement step costs less time. On the contrary, the grid-based approach directly accesses the third-level ($C_{\ell 3}$) cell to narrow down the search for a small amount of meta-data within a short time. Figure 3.11 (b) shows that the PA in R-tree for processing queries with direction is over eight times larger than the typical ones while the grid-based approach shrinks to about half. The reason is that the node size of the 3D R-tree is larger than that of the 2D R-tree, which results in accessing more pages in both the filter and refinement step. Because most of the PA is to memory pages, the difference in the PT which is not that large as the PA. Comparing queries with and without direction condition between R-tree and the grid-based approach, our algorithm significantly improves the performance for directional queries.

Effect of query rectangle size We next study the effect of the query rectangle size to range query. The query rectangle size varies from $125m$ to $500m$, which is from half to two times of the grid size δ. Larger area contains more number of videos and thus leads to longer processing time and more number of accesses. As expected, the result in Figure 3.12 shows that the PT and the PA increases with the query rectangle size. From Figure 3.12(a), the value of the PT increases slower using the grid-based approach, which means that our approach performs even faster for large query area than R-tree. However, Figure 3.12(b) shows that as the query area grows, the difference in number of the PA between these two methods gets larger when the query rectangle size increases from $125m$ to $250m$, and then becomes smaller after



(a) Processing time.

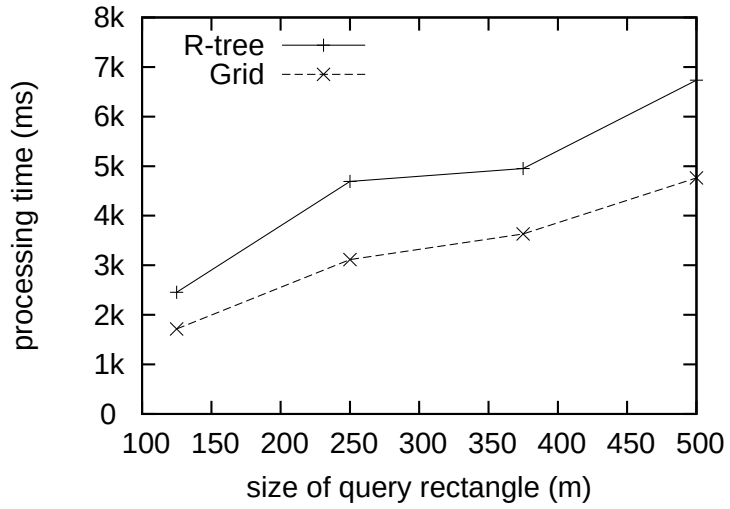


(b) Page accesses.

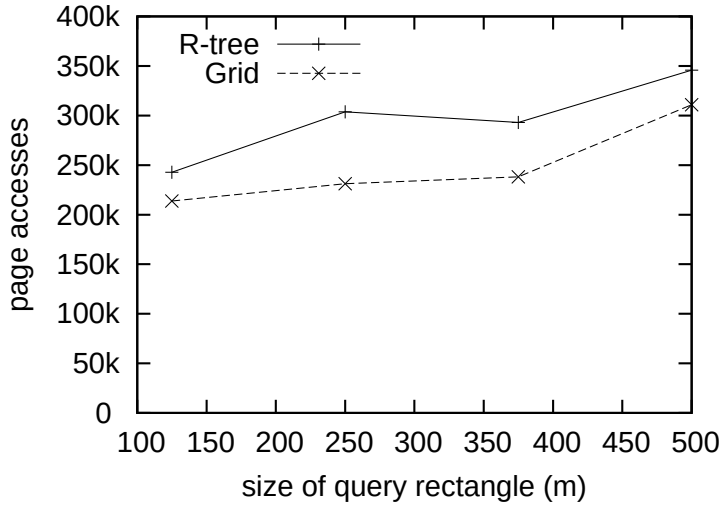
Figure 3.11: Effect of direction condition.

the query rectangle size is larger than $250m$, which the maximum visible distance of the camera. Therefore, when users are interested in what happened at special places or small regions, *e.g.,* an area with size $500m \times 500m$, our grid-based approach outperforms better than the R-tree.

Effect of k value To test the performance of the grid-based approach with different values of k for k-NVS query, we calculate the PT and the PA using the same query points. The results in this experiment are discrete FOVs (not video segments).



(a) Processing time.

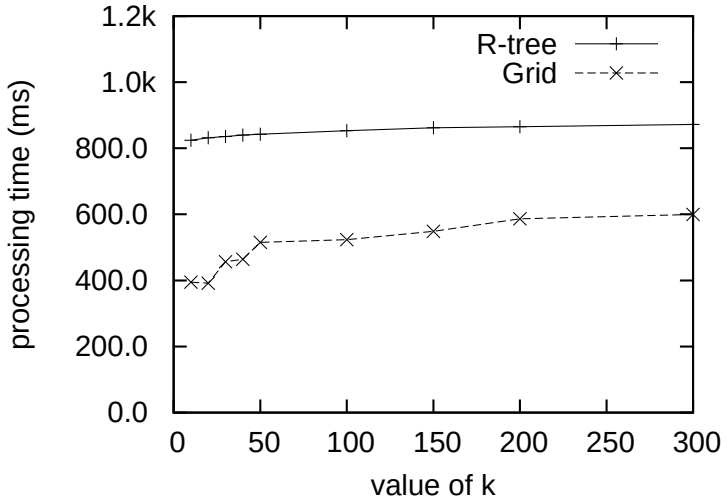


(b) Page accesses.

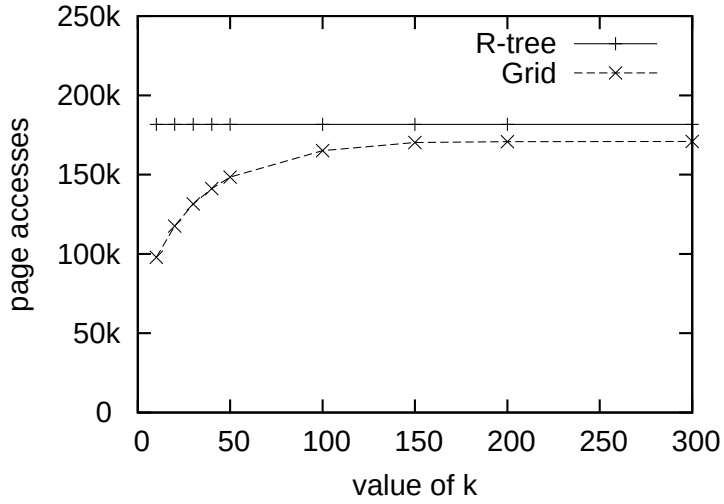
Figure 3.12: Effect of rectangle size on range query performance.

Figure 3.13(a) shows the comparison in the PT and Figure 3.13(b) shows the comparison in the PA. As k increases, the PT increases for the grid-based index at the beginning and keeps nearly unchanged when k is larger than 200, which is closed to the maximum number of FOVs found in typical point query. The PT for R-tree is almost the same with different k values because all the results are found and sorted once. When k is larger than 150, the PA for the grid-based approach is almost the same since the searching radius is enlarged to the maximum according to the

design of the structure. From the gap in Figures 3.13(a) and (b) between the R-tree and our approach, we can infer that even if the dataset is large and k is big, the grid-based index structure performs better than the R-tree.



(a) Processing time.



(b) Page accesses.

Figure 3.13: Effect of value of k on k-NVS query performance.

3.5 Summary

In this work we proposed a novel multi-level grid-based index structure and a number of related query types that facilitate application access to such augmented,

large-scale video repositories. Experiments on a real-world dataset show the importance of the queries with bounded radius and viewing direction restrictions. The experimental results with a large-scale synthetic dataset show that this structure can significantly speed up the query processing, especially for directional queries, compared to the typical spatial data index structure R-tree. The grid-based approach successfully supports new geospatial video query types such as queries with bounded radius or queries with direction restriction. We also demonstrate how to form the resulting video segments from the video frames retrieved.

The limitation of the this work is the small size of the real-world dataset, although the dataset size keeps growing. In the future, we will keep collecting data. Moreover, the current work only uses GPS and digital compass to build the FOV model and ignores the changes in camera parameters (*e.g.,* the zoom level, various viewable angle). Therefore, a more reasonable model including all the embedded sensors in smartphones is a potential research direction. Furthermore, the specific query types can be utilized to construct the 3D model of a building from video, and the index structure can be migrated onto Cloud for parallel processing and video hosting service with large-scale database.

Chapter 4

Uncertain Data Management

4.1 Introduction

GIS applications now increasingly make use of geo-located multimedia data such as images and videos. Furthermore, the wide-spread availablity of smartphones allows the acquisition of user-generated videos that are annotated with geo-properties. As stated in previous sections, these sensor data can be utilized to manage geo-tagged videos, such as building index, searching among large-scale dataset. However, a major practical issue is the noisy nature of such sensor data. It has long been known (and many users have had first-hand experience) that sensors, especially on consumer-grade electronics, produce sometimes inaccurate and fluctuating values. The surrounding environment can exacerbate the problem. For example, tall buildings and narrow passageways in urban areas can lead to very difficult conditions for obtaining accurate GPS positions. Figure 4.1 illustrates the situation that the GPS data collected during video recording might be inaccurate. When we collecting the location information using GPS devices, the raw data include an error range indicating that the actual location might be within the range away from the GPS reading. Besides, typically alignment errors, non-orthogonal errors and magnetic deviations can affect the digital compass heading accuracy. Due to sensor data inaccuracies the visual coverage described by the meta-data may not exactly match the actual video scene, which leads to imprecise search results and positional disagreements on map overlays.

Moreover, obstructions (*i.e.,* buildings, people or vehicles passing by) in front

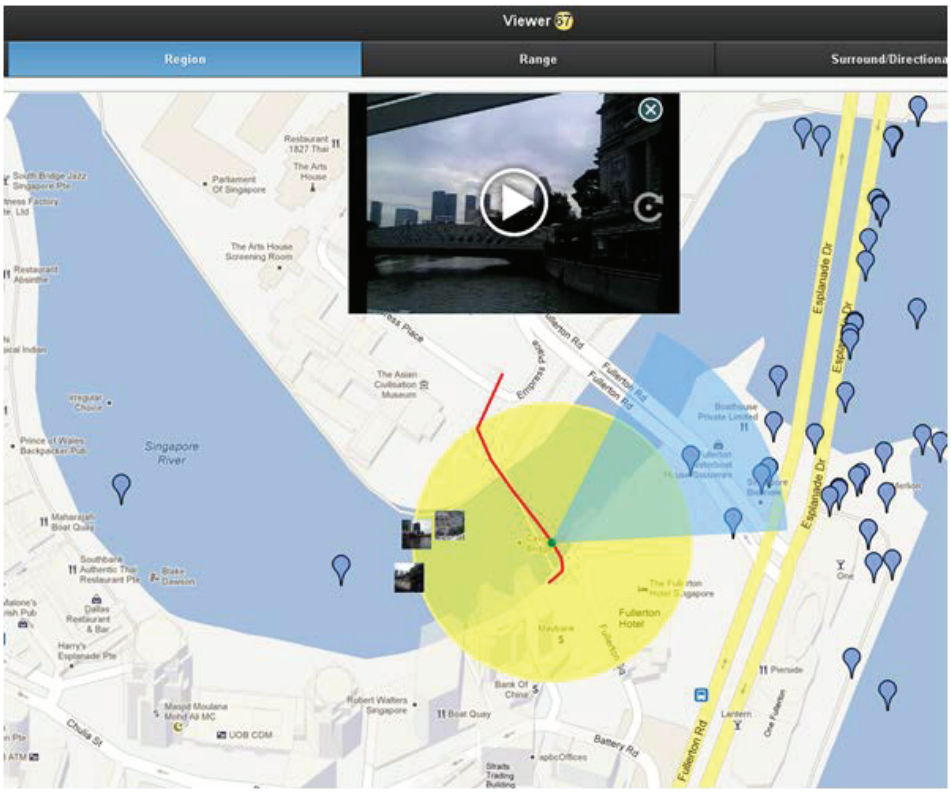


Figure 4.1: An snapshot from our GeoVid website that shows the collected GPS data are sometimes inaccurate. The red line is the camera trajectory. The yellow circle shows the GPS error range, which indicates that the actual camera location can be any point within the circled area. The pie-slice shape shows the coverage region of the FOV at that time the snapshot is taken.

of a camera may in parts of scene result in the recording of objects far away while only close ones in other parts within the same frame. For example, when people are recording videos in the urban area, some buildings are blocked by others due to the 3D visibility [Shen 2011]. Figure 4.2 shows a few sampling frames extracted from the videos we collected. The targeted building is blocked or partially blocked by other buildings, a small cabinet, and a person passing by, respectively. This effects influence the captured video scene, but not the measured sensor data.

The above described issues lead to a mismatch between the viewable scenes of video contents and the area represented by the sensor data. In general, there exist two typical ways to deal with these inaccurate data. The first one is to correct these inaccurate data. In order to correct the inaccurate sensor data, a set of ground-truth

Figure 4.2: Sampling frames extracted from videos collected show that obstructions can also cause a mismatch between the viewable scenes of video contents and the area represented by the sensor data.

data is necessary, which sometimes is difficult to obtain. Furthermore, even though the ground-truth data is available, the subsequential steps, which might involve the content-based comparison or camera motion detection, are still open and challenging problems. Rather than trying to completely avoid or correct such issues (which may be difficult or impossible), a well-known approach in the information management community is to design methods that can handle *uncertain data*.

In this work we propose a novel approach (HUGVid) which has the objective of modeling the uncertainty of a camera's viewable scene in the presence of noisy, imprecise sensor data. An approximate method is introduced for an efficient, but still effective, estimation of the uncertain data model. Based on the approximate model and geographical properties of a video, a video segmentation method is introduced which helps to describe the probability of a region being captured by a video segment. Finally we introduce efficient methods to perform probabilistic queries. The HUGVid architecture and data flow are presented in Figure 4.3. The contribution of our work can be summarized as follows:

- **Uncertain data model for individual geo-tagged video frames.** We model the spatial coverage of an individual video frame with multi-sensor meta-data and an approximate method for effectively calculating the probability of any place being captured by this model is introduced.

- **Video segmentation and uncertain data model for segments.** We

introduce a video segmentation algorithm with the aim that all frames within the parsed video segment capture a common region. This region is recorded by all frames with a non-zero probability. At the same time, we extend the uncertain data model to video segments.

- **Video indexing with extended R-tree.** For effective and efficient search, an R-tree based method is proposed to index the parsed video segments with geo-properties. Additionally, the statistical information of each video segment is attached to a standard R-tree as a secondary index.

- **Distance estimation.** When videos are uploaded, a centroid point is used to represent all the cameras' locations. We then utilize this point to approximately calculate the distance between the query and the video segment.

The above novel components of HUGVid allow the efficient processing of probabilistic queries over naturally noisy sensor data. Upstream GIS applications can utilize the results and prioritize their processing by concentrating on the most promising candidate video segments first.

We validate our design through extensive experiments with both real and synthetic datasets. The results show that the proposed approach produces high precision, and the approximate distances computed match well with the actual values. The result of a user study shows that our method also satisfies the human perspective. Moreover, the performance on the synthetic dataset indicates that our approach performs efficiently and effectively on large-scale datasets.

The rest of the work is organized as follows. Section 4.2 introduces the uncertain data model for a video viewable scene. Section 4.3 then presents the uncertain data model for video segments and demonstrates the geo-tagged video indexing method. Section 4.4 details the query processing. Section 4.5 reports the results on the evaluations of the proposed method. Section 4.6 summarizes the work.

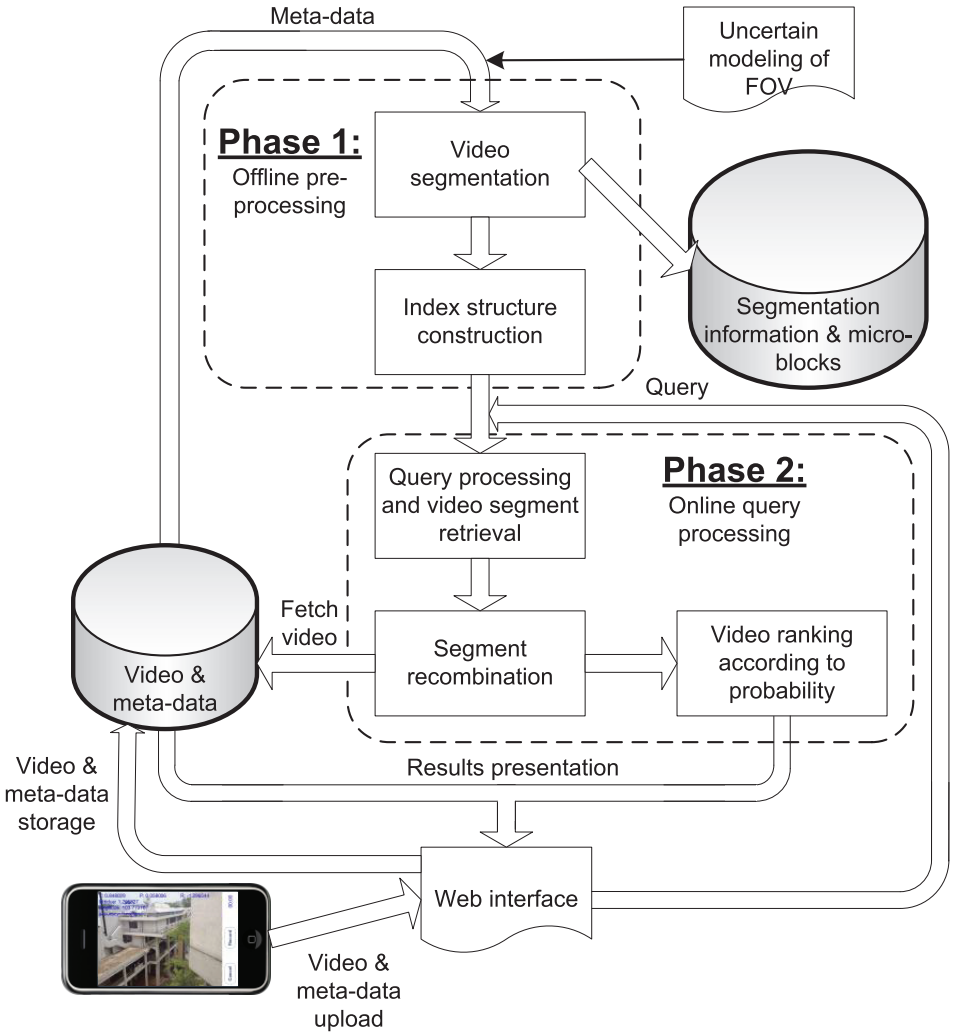


Figure 4.3: Architecture with uncertain data model and the corresponding tasks for processing queries.

4.2 Uncertain Data Modeling of FOV

There is an increasing awareness of the naturally occurring uncertainty of some spatial data, for example in the GIS and other communities. The uncertainty falls into two cases: sensor errors (Section 4.2.1.1 and 4.2.1.3) and obstacles (Section 4.2.1.2). In our geo-tagged video search application, the mismatch between video scenes and the associated geographical coverage significantly affects the final results. For example, when users desire to search for videos that capture the Marina Bay Sands (MBS) complex in Singapore, the system may find the videos that can theoretically

capture the MBS according to geometric calculations. However, if an object is located between the camera and MBS and blocks the building, or if the meta-data are inaccurate, the actual video may not capture the hotel complex. Therefore, an uncertain model for the camera's viewable scene is essential to measure the probability that a location actually appears in video scenes. When taking videos using smartphones, the GPS devices are collecting the location information as well as the accuracy. We manually checked the GPS accuracies and compass readings and found that most errors are within a certain range. Based on the FOV model presented in Section 2.2.1, the extended uncertain viewable scene model is introduced in Section 4.2.1, and finally Section 4.2.2 presents the simplified, approximate model.

4.2.1 Uncertain Data Model for FOV

As illustrated in Figure 4.4, the solid pie-slice shape is formed from the sensor meta-data while the dashed shape might be the actual video scene. For a single FOV, it is commonly assumed that an object has a higher probability of being captured by a camera if it appears close to the camera's location and in the center of the FOV. If not otherwise specified, we refer to the *probability* as the probability of an object being captured by an FOV or a video segment. Next, we describe the uncertainty of meta-data that affects the probability based on the FOV model in three independent dimensions: camera viewing direction, visible distance and camera location. Finally we will combine them together.

4.2.1.1 Uncertainty of Viewing Direction

Due to the inaccuracy of the digital compass, the measured viewing direction does sometimes not exactly align which the camera direction. The digital compass based orientation measurement is sensitive to motion and magnetic disturbances. In practice, the camera orientation can at best reach an accuracy of plus/minus few degrees with respect to the compass reading [1]. A study [Erifiu 2012] showed that the com-

[1] ACM Multimedia Grand Challenge 2010.

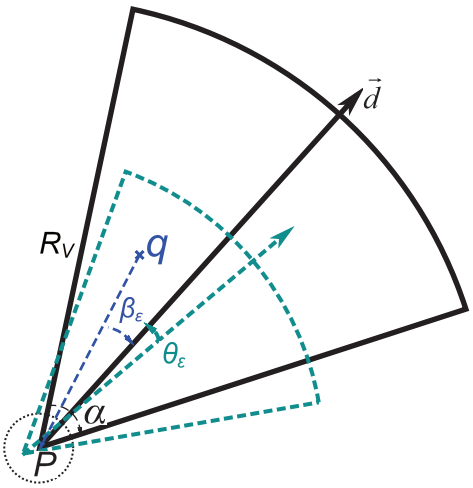


Figure 4.4: Illustration of the uncertainty of a field-of-view (FOV). The solid pie-slice shape is drawn from the sensor meta-data, while the dashed shape might be the actual FOV.

pass reading on an HTC G1 phone can achieve an overall five-degree drift to the actual orientation without magnetic interference. When the measurement is taken place in cars, near radio or speakers, near large metal structures, the compass accuracy degrades to 20-50 degrees, or even worse. In practice, we used different types of mobile devices (*e.g.,* iPhone, Android phones with different brands) to capture videos and collect sensor meta-data. Therefore, the inaccuracy of the digital compass might vary on different devices and different environment as well. Considering the above literatures, we manually checked the captured video and the associated compass data, and inferred that the distribution of the compass error satisfies a Gaussian function with a maximum error value. To calculate the ground-truth, we manually check the geo-coordinates of a few landmarks, and estimate the pixel location of the landmark in the frames. We then select the frames whose associated GPS accuracy falls within 10 meters. These GPS data of these frames are treated as the correct ones and used to calculate the camera viewing direction. Based on this observation, we assume that the maximum compass error is θ_ε ($\theta_\varepsilon < \alpha$). Thus, the camera can definitely capture the directions ranging from $(\theta - \frac{\alpha}{2} + \theta_\varepsilon)$ to $(\theta + \frac{\alpha}{2} - \theta_\varepsilon)$, while probably record other directions. As shown in Figure 4.4, the probability that

a given location q (*i.e.,* the query input) is covered by an FOV is presented in Equation 4.1:

$$Prob_q^{\beta_\varepsilon} = f(\beta_\varepsilon) = \begin{cases} 1 & (\beta_\varepsilon \leq |\frac{\alpha}{2} - \theta_\varepsilon|) \\ e^{\left(\frac{K_0 \cdot |\beta_\varepsilon - \frac{\alpha}{2} + \theta_\varepsilon|^2}{|\frac{\alpha}{2} + \theta_\varepsilon|^2}\right)} & (\beta_\varepsilon > |\frac{\alpha}{2} - \theta_\varepsilon|) \end{cases} \tag{4.1}$$

where β_ε denotes the angle offset between $|Pq|$ and $\vec{d}$, and K_0 is a constant ensuring that it is a small probability event that q is covered by the FOV when β_ε is larger than $\frac{\alpha}{2} + \theta_\varepsilon$.

4.2.1.2 Uncertainty of Obstacles

Most current smartphones lack an optical zoom and it is difficult to extract the focal length information. This leads to difficulties when measuring the distance between a camera and the objects it records. Moreover, due to obstructions, it is possible that some objects are hidden by others, or objects at different distances appear in the same scene. This situation has no effects on the sensor data but affects the final search results. Taking the scenes extracted from a video as an example (shown in Figure 4.5), in this case the camera neither moves nor rotates much during video recording. Due to the bus passing by, the Esplanade building appearing at time 00:01:20 is hidden for a few seconds, even though it is much larger than the passing by bus. Our conjecture is that when an object is closer to the camera, the probability of it being blocked by other objects is lower. Thus, the probability of the object captured by the camera is higher. A Gaussian function is therefore one of the possible ways to represent the probability of objects not obstructed by others (Equation 4.2):

$$Prob_q^d = g(d) = e^{\left(\frac{K_1 \cdot |d|^2}{|\sigma|^2}\right)} \quad (d \leq R_V) \tag{4.2}$$

where d is the distance between the camera and the object, and K_1 and σ are the parameters to satisfy that it is a small probability event that any object outside of

R_V is captured by the camera.

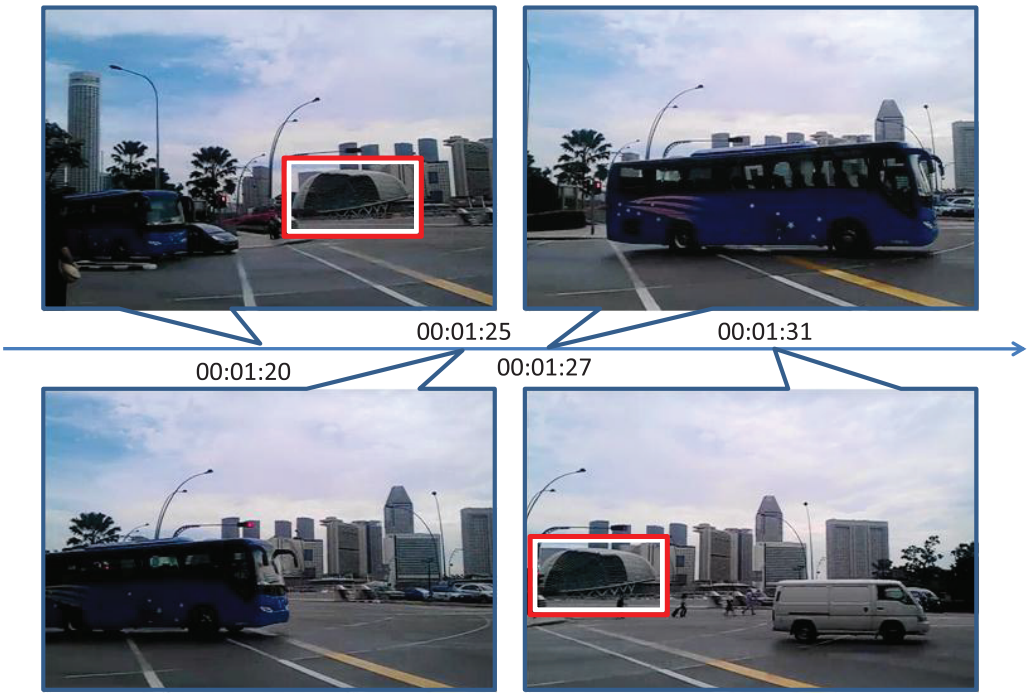


Figure 4.5: Demonstration of the uncertainty of the visible distance. The Esplanade building (highlighted) has been hidden by a bus for a few seconds.

Hence, we obtain the uncertainty model for the FOV whose camera location is at a specific position (*e.g.*, with geo-coordinates $P(x_0, y_0)$) by combining the above two uncertainty models. Equation 4.3 shows the probability that $q(x, y)$ is captured by the FOV when the camera is located at $P(x_0, y_0)$. All the parameters used here can be obtained from sensors or the configuration settings of the cameras.

$$
\begin{aligned}
Prob_q^{P(x_0,y_0)} &= Prob_q^{\beta_\varepsilon} \cdot Prob_q^d \\
&= \begin{cases}
Prob_q^d & (\beta_\varepsilon \leq (\frac{\alpha}{2} - \theta_\varepsilon)) \\
e^{\left(\frac{K_0 \cdot (\frac{\pi}{2} - \gamma - \theta)^2}{(\frac{\alpha}{2} + \theta_\varepsilon)^2}\right)} \cdot Prob_q^d & (x > x_0 \wedge \beta_\varepsilon > (\frac{\alpha}{2} - \theta_\varepsilon)) \\
e^{\left(\frac{K_0 \cdot (-\frac{\pi}{2} - \gamma - \theta)^2}{(\frac{\alpha}{2} + \theta_\varepsilon)^2}\right)} \cdot Prob_q^d & (x < x_0 \wedge \beta_\varepsilon > (\frac{\alpha}{2} - \theta_\varepsilon))
\end{cases}
\end{aligned}
\tag{4.3}
$$

Here $Prob_q^d = e^{\left(\frac{K_1 \cdot ((x-x_0)^2 + (y-y_0)^2)}{\sigma^2}\right)}$, $\gamma = \arctan(\frac{y-y_0}{x-x_0})$, and $\beta_\varepsilon = |\gamma - \theta|$.

4.2.1.3 Uncertainty of Camera Location

When measuring a location with a GPS device, the position reading is accompanied by an error range d_ε. Therefore, the actual position of the camera is located within a circle around the GPS reading with a radius of d_ε (as shown in Figure 4.4). Due to various reasons (*i.e.,* a tunnel traversal), the GPS locations are sometimes missing for a while. In this case, we estimate the object location by applying positional interpolation techniques [Arslan Ay 2008]. A number of research [Sistla 1998, He 2005, Rife 2012, Okuda 1993] has proposed that the object location follows a Gaussian distribution inside the uncertainty region, as shown in Equation 4.4:

$$Prob_q^L = h(x_0, y_0) = \frac{e^{-\frac{1}{2(1-\rho^2)} \cdot [\frac{(x_0-\mu_x)^2}{\sigma_x^2} - \frac{2\rho(x_0-\mu_x)(y_0-\mu_y)}{\sigma_x \sigma_y} + \frac{(y_0-\mu_y)^2}{\sigma_y^2}]}}{2\pi\sigma_x\sigma_y\sqrt{1-\rho^2}} \tag{4.4}$$

where (μ_x, μ_y) denotes the GPS reading, $Prob_q^L$ denotes the probability that the camera is at location (x_0, y_0), and σ_x, σ_y and ρ are parameters of the probability density function. Finally, the probability of a location to be covered by an FOV is the accumulation of different possible camera positions (see Equation 4.5). The distribution of the probability is illustrated in Figure 4.6 with a series of parameters.

$$Prob_q = \sum_{d=0}^{d_\varepsilon} Prob_q^L \cdot Prob_q^{P(x_0,y_0)} \tag{4.5}$$

4.2.2 Approximate Uncertain Data Model

As observed from Figure 4.6, the coverage region of the probability in 2D space is an irregular shape (the solid shape in Figure 4.7). Given such an irregular shape, it is computationally expensive to calculate exact probabilistic values, especially for a large set of FOVs. Therefore, we have developed an approximate method to estimate the probability, which can be carried out with a few calculations. As presented in Sections 4.2.1.1 and 4.2.1.2, the probability is independent of the direction and the

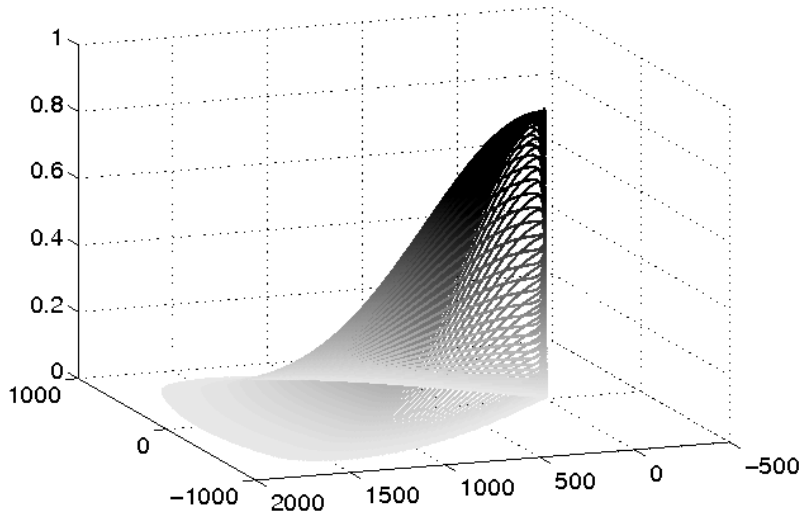


Figure 4.6: The probabilistic distribution of an area being captured by an FOV at a specific position.

distance domains. For example, if the distance d between the camera and an object is larger than a certain value (*e.g.,* 240 m), the probability affected by distance is smaller than a value, *e.g.,* 50%. Hence, according to Equation 4.3, the probability must be smaller than 50% no matter what value β is. The same situation occurs for the direction domain. Consequently, we can find a border value for each of the two domains, denoted β_b and $dist_b$, respectively, below which the probability is less than a threshold τ. These two border values form a pie-slice shaped region, ensuring that the probability is smaller than τ outside this region. We then use this area to estimate the probability instead of the irregular shape. In Figure 4.7, P is the camera location according to the GPS reading while P' is a reference point for calculating β_b and $dist_b$. The geo-coordinates of P' can be computed by using R_V, d_ε, α and the location of P. The irregular shapes within the solid lines are the actual probabilistic regions while the pie-slice shapes within the dashed lines are the approximate regions. The exact probability falls on the solid line, *i.e.,* the probability is 50% on the inner solid lines. The regions between the solid lines and the dashed lines are so-called “false positive" regions. For example, the probability

of the region between the inner solid shape and the inner dashed lines is actually less than 50% while it is considered greater than or equal to 50% in the approximate model.

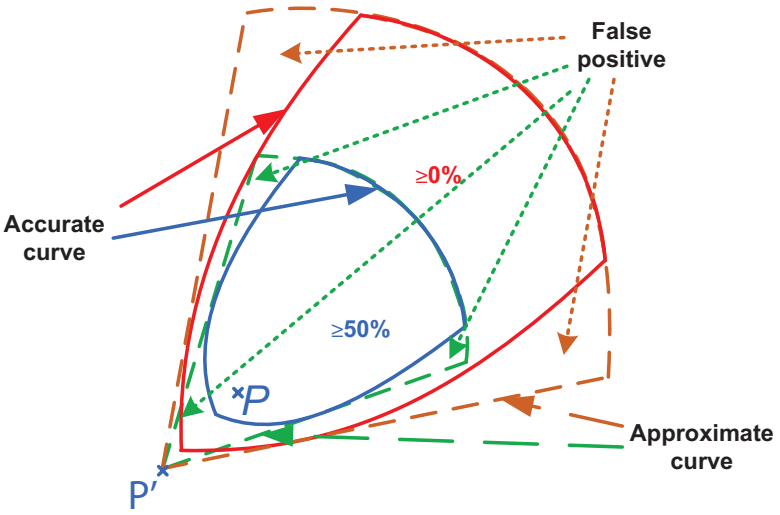


Figure 4.7: Illustration of the approximate uncertain FOV model in 2D space.

4.3 Offline Uncertain Data Modeling of Video Segments

The approximate model described in Section 4.2 applies to a single FOV, indicating that it works for geo-tagged images. Building on this, we represent the coverage of a video clip as a spatial object through a series of FOVs. Therefore the next step is to extend our probability model from a single FOV to a sequence, representing a video segment. This is performed by parsing the video into segments according to the associated geo-properties of the FOVs. Additionally, to support large-scale geo-tagged video search applications, we index these spatial objects using an extended R-tree such that they can be effectively and efficiently retrieved. In Section 4.3.1, the uncertain data model of a video segment is introduced and a video segmentation algorithm is presented. Section 4.3.2 provides the details of constructing the index structure.

4.3.1 Uncertain Data Model for Video Segments

Based on the introduced uncertain data model for a single FOV, we now develop the uncertain model of a video segment. One of its benefits is that it can retrieve meaningful results even though there exist sharp jitters among sequential sensor data, which is an essential feature for a robust system. Treating an entire video segment as a spatial object is semantically more meaningful compared to dealing with individual frames. However, treating the whole video as an object may not be a good choice since some long videos or videos captured at high speed may cover a large region. Moreover, processing the entire video results in redundant calculations and increases processing time. Our goal is hence to parse the video into segments and model the probabilistic distribution of each such segment.

Broadly speaking a video segment is represented by its coverage region. When recording videos, people generally point the camera at interesting places even if they are moving along a trajectory. The common, overlapping region among multiple FOVs can hence be considered a Point of Interest (POI). For this reason we aim to parse the video such that the FOVs within each parsed segment overlap with each other. This way the likelyhood is high that there exists at least one local POI captured by each video segment. A search will return a video segment without any further processing if users want to search for videos showing a local POI. To parse the video in this manner, the overlapping area between FOVs needs to be calculated. Once a geo-tagged video is uploaded to the server, the video parsing process is activated. It scans the meta-data forward from the beginning and calculates the overlap among the FOVs until a subsequent FOV is identified which does not intersect with the current overlap region. Then the video is partitioned before this FOV and a new segment is started. Figure 4.8 shows a video clip captured by a camera moving along a trajectory. The video is parsed into three segments illustrated in different line styles. The latticed regions depict the overlap areas, indicating local POIs within the parsed video segment.

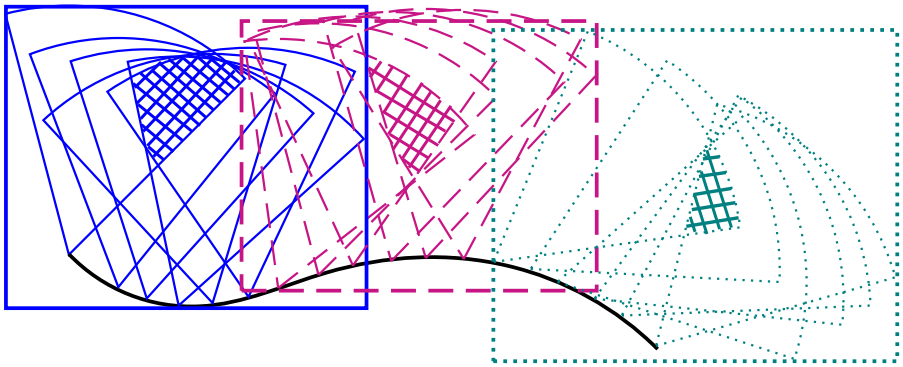


Figure 4.8: Illustration of the video segmentation algorithm. The video is divided into three segments and the latticed regions illustrate the common overlap within each segment.

Next, we introduce the overlap calculation and design the uncertain data model of a video segment based on the overlap calculation. We define the probability that an object is captured by a video segment as follows:

Definition 1. The probability of a position being captured by a video segment is the probability of it being captured by all frames within this segment (formalized in Equation 4.6):

$$Prob(P(x,y), VS(0,n-1)) = 1 - \prod_{i=0}^{n-1} (Prob_{cap(i)}\overline{(x,y)}) \tag{4.6}$$

Here VS denotes the video segment, and $Prob_{cap(i)}(x,y)$ is the probability that an object at location $P(x,y)$ is captured by the i^{th} FOV of the segment. With this definition the probability is high when the query region is close to P and along the camera's viewing direction $\vec{d}$, and when most of the FOVs within the segment point towards P. We subsequently use this probability to help rank the results.

It is computationally complex to find the overlap regions between FOVs with a pie-sliced shape. The Monte Carlo method [Metropolis 1949] is one approach for estimating the area of a shape by generating random sampling points. In our geo-tagged video search application, the maximum visible distance R_V is usually hundreds or thousands of meters. Thus, it is impractical to perform the overlap

calculation with dense random sampling points. Instead, we divide the MBR of the segment into micro-blocks of size $\delta_b \times \delta_b$ ($\delta_b \ll R_V$). The probability within each micro-block can be treated as constant since δ_b is much smaller than R_V. We select the center point of each block as the sampling point during the Monte Carlo procedure. All the other points inside a micro-block are treated equivalently to its center point, *i.e.,* the whole micro-block is considered as the overlapping area among FOVs if the center point of this micro-block is covered by all the FOVs. Note that a smaller δ achieves higher accuracy with this method. The current settings used in our experiments work for most cases. The only possible difficulty is an overlap at the border of an FOV, which usually has little or no effect on the final results and may be unimportant to users. The tradeoff is that retrieving more accurate results will cost more processing time and storage space. Since the probability of each micro-block is only slightly different from its adjacent micro-blocks, it is not necessary to maintain the exact value for each one. To simplify the calculation, we quantize the probabilistic range into several levels, which decrease as the probabilistic level falls. Any location out of the approximate FOV model is assigned zero.

Next, we present the video segmentation algorithm based on the overlap among micro-blocks. When parsing a video clip, the first FOV is selected as a reference. Initially the probabilities of all micro-blocks within this reference FOV are calculated and all blocks are stored in the overlap set *overlapBlocks*. When processing the next FOV, blocks that fall within both FOVs are retained in *overlapBlocks*, while others are moved to the set $non-overlapBlocks$. The probability of each block is updated using Equation 4.6. A video segment is completed once *overlapBlocks* is empty. Note that if the latest inserted FOV has no overlap with the ones ahead, an undo procedure is carried out to guarantee that there exists overlap within a video segment. The detailed process is shown in Algorithm 6. The overlap region is usually very important to the user. Although this parsing algorithm is based on a heuristic and may result in missed global POIs, it still mines the local ones. To

overcome this limitation, a post-processing procedure is carried out when retrieving the results during querying.

Algorithm 6: VideoSegmentation()

```
Input: a video clip V_i{FOV_i0, FOV_i1, ..., FOV_im}
Output: video segments VS{V_i0, V_i1, ..., V_in} (m ≥ n)
overlapBlocks{} ← ∅, non−overlapBlocks{} ← ∅
/* FOV_i0 stands for the 0th FOV in Video V_i                    */
for all blocks within FOV_i0 do
    /* B_j stands for the jth block in space                     */
    overlapBlocks.append(B_j);
videoCount = 0;
for f=1 to m do
    for all blocks B_t within FOV_if do
        if B_j==overlapBlocks.find(B_t) then
            /* B_j is called a hit block                         */
            Prob(B_j).update();
        else
            non−overlapBlocks.append(B_t);
        move all non-hit blocks from overlapBlocks{} to non−overlapBlocks{};
        if overlapBlocks{} is not empty then
            VS_(videoCount).append(F_if);
        else
            calculate MBR of the segment;
            videoCount+=1;
            undo for the last inserted FOV;
            VS_(videoCount).append(F_if);
            reset all sets;
            for all blocks B_t within FOV_if do
                overlapBlocks.append(B_j);
return VS{V_i0, V_i1, ..., V_i(videoCount)};
```

4.3.2 Video Segment Index Structure

Since we represent the coverage of a video segment as a spatial object we can utilize a popular spatial index structure such as the R-tree to help effectively and efficiently search for requested videos. As described in Algorithm 6, the MBR of a video segment is obtained once the segment is partitioned from the original video. In our geo-tagged video search application, we construct an extended R-tree that regards each parsed video segment as an entry. The difference from a standard R-tree is that

the original video ID, the anchor of the starting frame and the segment length are attached to the leaf nodes. In order to quickly process kNN queries, we also store the camera location information in the leaf nodes. However, it is usually impractical and unnecessary to store all the camera locations within a segment. We instead use a centroid point to represent all the camera positions. The geo-coordinates of this centroid represent the average of all the camera locations in both latitude and longitude and we store the micro-block information in which the centroid is located. This information is utilized to estimate the average distance between the query and camera locations and is treated as the distance between the query and a video segment. Additionally, the corresponding probability map of each segment is attached to the leaf nodes as a secondary index.

The structure of the probability map is quite similar to an image. Thus, an image compression method might be helpful to save storage space. A discrete cosine transform (DCT) is widely used in signal and image processing, especially for lossy data compression. In our application, there is a tradeoff between storing the probability map directly and compressing it using DCT. We experimented with compressing the maps to a quarter of their original size and found that the probability level error rose to at least 10%. Furthermore, a DCT calculation needs to be carried out for each of candidate maps, which is inappropriate for an online query due to the increasing processing time. Consequently, we decided to store the probability map directly on the disk.

4.4 Online Query Processing

Given the offline pre-processing calculations described in Section 4.3, the online query procedure is performed in two major steps. First, the result video segments (referred to as *candidates*) are retrieved by searching through the R-tree. In the second step, the final segments are reconstructed by combining candidates to preserve video continuity. Note that our approach can retrieve videos without overlap

calculations between the query region and the individual video frames. Also, the probability is updated after recombination. Importantly, when presenting the results the video segments are sorted according to their probability.

4.4.1 Searching for Segment Candidates

Utilizing the R-tree index we process both typical spatial queries (including point, range and kNN queries) and queries with a probabilistic threshold to search for videos in large-scale datasets. Since we use an MBR to represent the video coverage, all the queries are processed with overlap calculations between query locations and MBRs. Generally, for all query types, we need to search from the root to the leaf nodes of the R-tree to find the video segments whose MBRs overlap with the query. Once the candidates are obtained, the corresponding probability maps are fetched.

The basic query procedure is described in the above paragraph, while the differences among various types of queries are as follows:

- **Point Query:** Once the leaf nodes containing candidates are retrieved from the R-tree, the corresponding probability maps attached to the leaf nodes are loaded into memory. Then the micro-block is identified in which the query point is located and the probability of that micro-block is obtained.

- **Range Query:** The difference to the point query is that the query rectangle may cover more than one micro-block. Hence, the maximum probability among all overlapping micro-blocks is selected as the result.

- **kNN Query:** The distance between the query and a video segment is defined as the average distance between the query and all its frames. When processing a kNN query, we approximately calculate the distance between the query point and the centroid point of the micro-block whose information is stored in the leaf nodes of the R-tree. Thus, we treat this as the distance between the query and the video segment and use it to rank the results.

- **Query with Probabilistic Threshold:** The basic processing of a query with a probabilistic threshold is the same as with the above typical queries. The difference lies in filtering out the candidates whose probabilities are smaller than the threshold and then reconstruct the candidates.

4.4.2 Candidate Recombination

As presented in Section 4.3.1, the video segmentation algorithm may result in missed POIs. In that case, frames that capture the same POI may be parsed in two consecutive segments, which is not what users expect. Considering the example in Figure 4.8, the last two FOVs of the first segment and the first three FOVs of the second segment cover a common region. When the query is located in these regions, the segments indexed by the R-tree may not be the optimal representation of the results. Consequently, when the results include segments that are contiguous in the original video, a recombination operation needs to be carried out as detailed in Algorithm 7. Consecutive segments in the results are re-combined into new segments. The procedure terminates when there are no more such segments. During this process, the probability is also updated using Equation 4.7. The videos are then ranked in descending order according to their probabilities. Alternatively, for kNN queries, the videos are ranked according to their distance from nearest to farthest without recombination.

$$Prob(V_{ij}) = 1 - (1 - Prob(V_i)) \cdot (1 - Prob(V_j)) \tag{4.7}$$

Here V_i and V_j are two consecutive segments and V_{ij} denotes the combination of these two segments.

Algorithm 7: Recombination()

Input: Candidate set $vSet\{V_0, V_1, ..., V_n\}$
Output: Results $rSet\{V_{r0}, V_{r1}, ..., V_{rm}\}$ $(m \leq n)$
for *all segments* V_i *in* $vSet\{\}$ **do**
    if V_i.*findAjacent()* == V_j **then**
        V_{ij} = combine(V_i, V_j);
        $vSet$.remove(V_i);
        $vSet$.remove(V_j);
        $vSet$.append(V_{ij});
        prob(V_{ij}).update();
    else
        $vSet$.remove(V_i);
        $rSet$.append(V_i);
return $rSet\{V_{r0}, V_{r1}, ..., V_{rm}\}$;

4.5 Experimental Evaluation

We performed our experiments on from Dataset1 to Dataset3. We used Dataset1 to test the functionality of HUGVid and Dataset3 to demonstrate its scalability for large-scale applications. For all the experiments we constructed a local MySQL database in which we stored the FOV meta-data. We inserted the MBRs of all the parsed video segments and the relevant statistical information into our extended R-tree and processed all types of queries based on the R-tree [Green 2010] implemention by Melinda Green. We found this implementation to be very mature and achieve excellent performance. Additionally, we treated each FOV as an object and indexed it using the same structure to form a *baseline method* (BM) for comparison.

Table 4.1: The parameters used in the construction of the uncertain data model.

K_0	α	θ_ε	K_1	σ_b	d_ε	δ	R_V
-23.237	60°	10°	-0.137	300m	10m	20m	2km

In the following experiments, if not otherwise specified, the micro-block size δ was set to 20 meters, which is small compared to the camera's maximum visible distance R_V. R_V may vary due to different devices and resolutions. For example, many smartphones now can record 1080p HD video. According to existing meth-

ods [Arslan Ay 2008], the maximum visible distance of, for example, an iPhone 4S can be estimated as 1470 meters since its CMOS sensor size is 1/3.2 inches and its focal length is 35 mm. To make our approach robust with videos captured from different devices, we set the value of R_V as 2 km. The angle α may be calculated with the image sensor size and the camera focal length of the lens [Graham 1965]. However, it is difficult to obtain the focal length of the camera on a mobile device and hence to calculate the precise value of α. Due to videos being captured with different smartphones, α might vary among different devices. Therefore, we use a large, practical value for α, which is set to 60°. The values of θ_ε and d_ε are obtained from real-world data. We manually checked GPS accuracies in our data and found that over 86% of GPS errors fall within 10 m (the GPS error range can be collected from GPS raw data), and that over 90% of the compass reading errors are less than 10°. Other parameters, *i.e.,* K_0 and K_1 are set to satisfy a small probability event when constructing the uncertain data model (presented in Section 4.2.1). The detailed parameters are summarized in Table 4.1.

4.5.1 Experiments with Dataset1 and Dataset2

Since the range and kNN queries are representative among the query types, we used the collected data to process these queries and demonstrate the functionality of HUGVid. We selected five landmark places in Singapore (the Marina Bay Sands, the Merlion, the Esplanade, the Singapore Flyer, and the One Marina Boulevard) and two in Chicago (Chicago City Hall and Bulter Field) for range queries, which we refer to as *Q1* to *Q7*. Within each query region, one representative point is selected for the kNN queries. Figure 4.9 shows two samplings of frames in the Marina Bay Sands region. As shown in Figure 4.9(a), the query area is the solid rectangle shown on Google Maps and the 3D image extracted from Google Earth illustrates what the videos are assumed to capture. The ten surrounding images are sampling snapshots from the result video segments with their respective probabilities, pinned

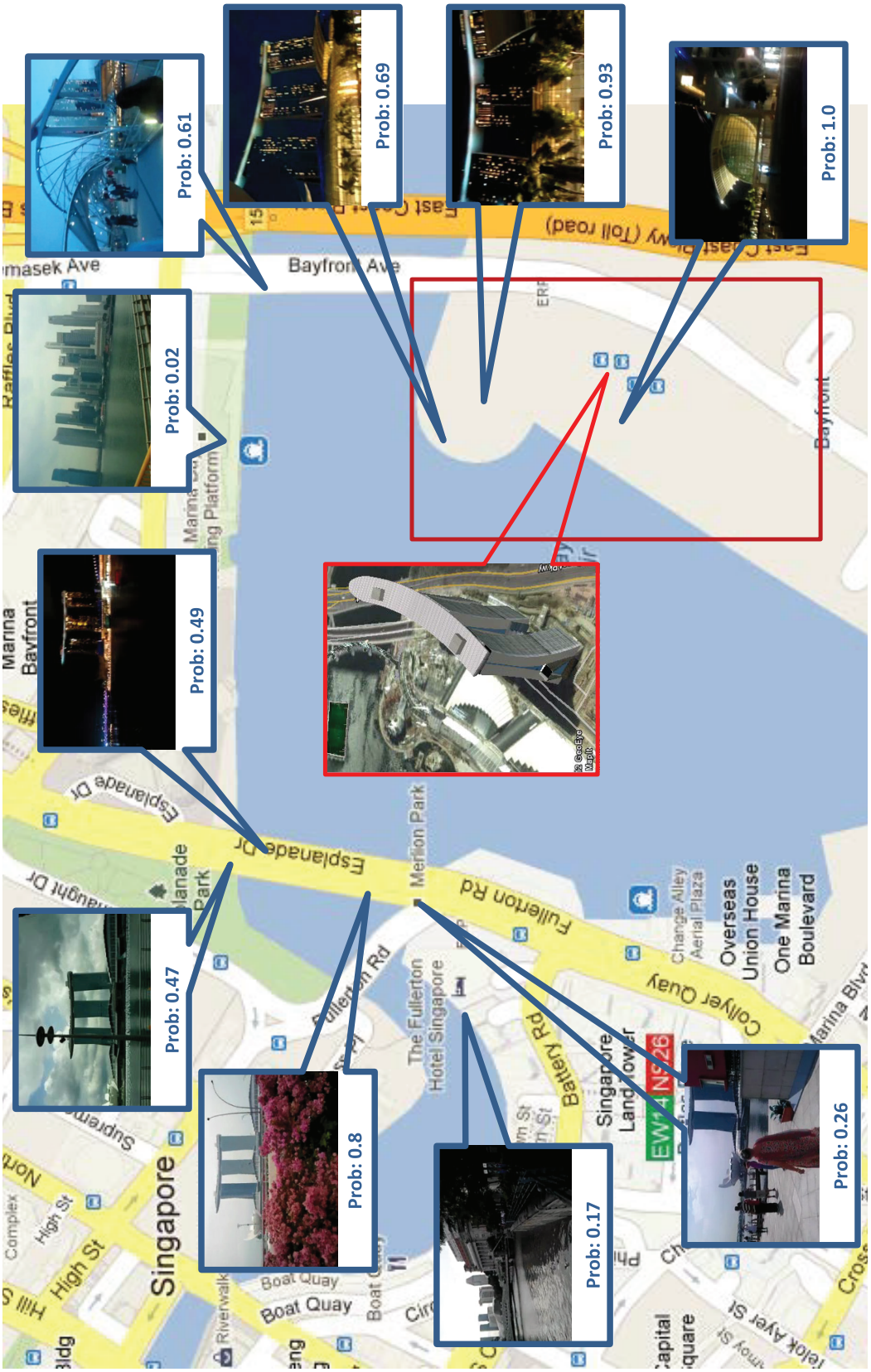
to their camera locations. The figure nicely illustrates how frames with higher probabilities capture the target landmark well and at close range while the ones with lower probabilities only capture parts or none of the target, or at a far distance. Figure 4.9(b) shows a sampling of frames of *Q2*. Observed from the sampling frames, only the results with probability larger than 0.6 can display a clear view of the Merlion. The frames with the probability value of 0.53 and 0.47 are captured from a relatively closed position but due to obstruction from the other buildings, the Merlion disappears in the frame. The frames with the probability value less than 0.4 are actually capture the region where the Merlion stands, but it is difficult to tell where the Merlion is due to its small size from the frame.

Precision and recall. Next, we studied the accuracy and redundancy of HUGVid. We manually watched all the videos and recorded the IDs of video frames which showed the query location. This was considered as the ground-truth (GT for short). We then compared the video segments retrieved using HUGVid with the GT. The precision and recall of HUGVid is presented in Table 4.2. The high recall shows that HUGVid retrieves almost all the video scenes in the GT. Conversely, it also includes some FOVs not in the GT, which leads to the low value for precision. The reason is two-fold: first, the probabilistic method finds more possible FOVs using the uncertain data model, and second, extra FOVs are included during video segmentation. Although more FOVs are found by HUGVid, it returns only half of the video segments after segment recombination.

Table 4.2: The precision and recall of HUGVid with different queries.

Query	*Q1*	*Q2*	*Q3*	*Q4*	*Q5*	*Q6*	*Q7*
Precision	0.7675	0.4821	0.6968	0.6973	0.5644	0.6749	0.6518
Recall	0.9976	0.9948	0.9971	0.9967	0.9959	0.9742	0.9655

User study. It is difficult to find an objective method to evaluate the query results of searching visual content and perform ranking. Therefore, a user study is an appropriate methodology to evaluate how well our approach satisfies the user perspec-



(a) Sample frames for query $Q1$.

Figure 4.9: Sample frames for queries in the Marina Bay Sands region. Ten snapshots are chosen from resulting videos with different probabilities.

(b) Sample frames for query *Q2*.

Figure 4.9: Sample frames for queries in the Marina Bay Sands region. Ten snapshots are chosen from resulting videos with different probabilities.

tive. Our study involved 21 persons (11 females and 10 males). The participants, which included students and professionals working in different fields (*e.g.,* computer science, biological engineering, and public services), were familiar with the query region. We processed *Q1* to *Q5* and then selected five different video segments (overall about 30 minutes) according to their probabilities from each query result. We chose segments of different probabilistic levels, *e.g.,*one segment with a probability higher than 0.8, one with a probability higher than 0.6 and lower than 0.8, and so on. This made it easy for the users to differentiate. We ranked these five segments according to their probabilities and scored them in descending order. The participants were then asked to watch these videos and rank the segments according to the time duration, the position, and the integrality of the queried place appearing in the scene, while ignoring the video quality, the weather and the time. The HUGVid ranking was then compared with the user ranking.

Table 4.3 presents the comparison between ranking by the algorithm and ranking by the users from *Q1* to *Q5*. The first row shows the score assigned by HUGVid while the last two rows are statistics from the user ranking. We then presented all the five query results: *Q1* targets a tall and wide landmark, *Q2* targets at a small statue, *Q3* targets at a theater, *Q4* target at a sky wheel, while *Q5* targets at a tall office building. The ranking between HUGVid and the users for *Q1* does not exactly match, especially for the resulting Video 1, even though Video 1 captures *Q1* from a close location. Even users disagreed on Video 1: some chose it as their favourite while others disliked it. The reason is that for a large building, some users desire to watch a panoramic view while others wish to view an up close shot with details instead. Conversely, for targets that are not so large (*e.g., Q2, Q3*), HUGVid shows consistent results. The scores given by the users are almost the same as those by the algorithm. Moreover, the low standard deviation indicates that most users agree with the manner in which HUGVid ranks videos. Although *Q4* is also very large, the algorithm score still satisfies the users' perspectives at

the higher score interval (*i.e.,* the score of 4 and 5) but not exactly matches at the lower score interval. The reason is that the most interesting part of the Singapore Flyer halls in the air, people prefer video segments captured nearby. It makes not much difference to people when segments are recorded from far away, the score of which falls into the lower score interval. The score for *Q5* from algorithm ranking is quite different from that from users' evaluations due to the obstructions: there exist a few buildings in front of the One Marina Boulevard building so that the targeted building is hidden behind.

Table 4.3: Scores from ranking by the HUGVid algorithm and by the users (1 – least, 5 – most relevant).

Q1	V1	V2	V3	V4	V5
Algorithm score	5	4	3	2	1
Average score	3.05	4.24	3.33	2.38	2.00
Standard deviation	1.717	1.091	0.966	0.973	1.140
Q2	V1	V2	V3	V4	V5
Algorithm score	5	4	3	2	1
Average score	4.86	4.00	3.00	1.90	1.24
Standard deviation	0.359	0.632	0.447	0.539	0.539
Q3	V1	V2	V3	V4	V5
Algorithm score	5	4	3	2	1
Average score	4.52	4.29	3	1.86	1.33
Standard deviation	0.602	0.561	0.548	0.478	0.913
Q4	V1	V2	V3	V4	V5
Algorithm score	5	4	3	2	1
Average score	4.57	4.10	2.24	2.86	1.23
Standard deviation	0.598	0.768	0.831	0.910	0.539
Q5	V1	V2	V3	V4	V5
Algorithm score	5	4	3	2	1
Average score	3.00	3.71	4.34	2.52	1.43
Standard deviation	1.517	0.845	0.796	1.030	0.746

Approximate distance. We also evaluated HUGVid on its ability to estimate the distance between the query and the video segments in a kNN query. BM shown in Figure 4.10 represents the average distance calculated using the geo-coordinates of the query point and all the camera locations. The resulting 311 video segments are sorted by ascending distance from BM. Comparing the four approximate distances

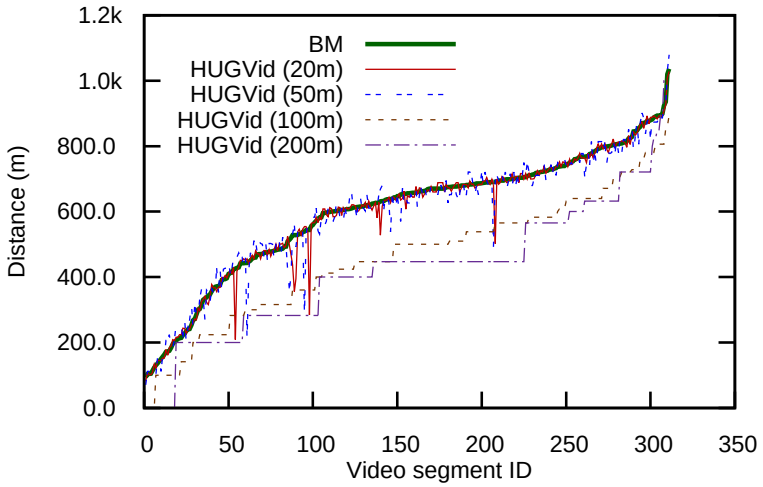


Figure 4.10: Comparison between the distance from BM and HUGVid with different micro-block sizes.

estimated by HUGVid with different micro-block sizes, the most accurate results are obtained from HUGVid with the smallest micro-block size. Since we utilized the position of a local POI to help estimating the distance, the errors are proportional to the block-size. In order to achieve accurate results, we chose to use 20 m as the default micro-block size (δ_b). There exist a few outliers where the distance from HUGVid significantly differs from that of BM when δ_b is no larger than 50 m. We manually checked those videos and found that all the segments with outliers are from the same unparsed video. In that video, the GPS raw data is extremely inaccurate, jumping from one location to another about 1 km away and then jumping back to its previous location. This situation is very rare and outside of the common GPS error range. Moreover, the reason that the outliers with different micro-block sizes appear in different segments is that different micro-block sizes lead to different video segmentation in some situations.

4.5.2 Experiments with Dataset3

All the experiments are conducted on a server with two quad core Intel® Xeon® $X5450$ 3.0 GHz CPUs and 32 GB memory running Linux 2.6.18. Among all the

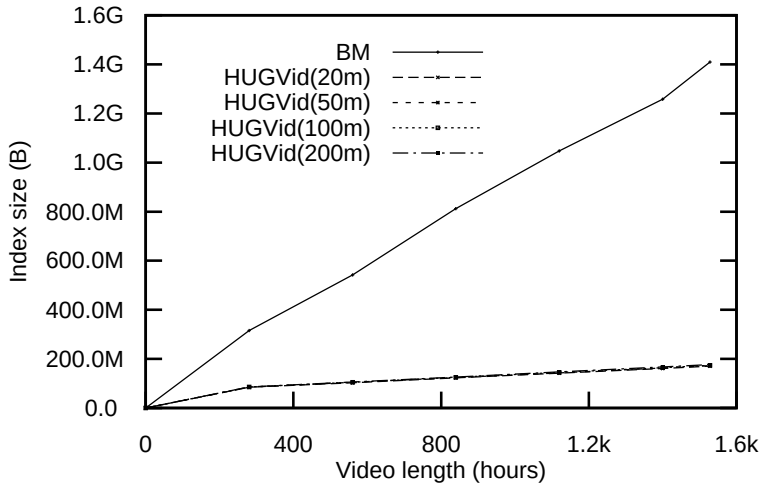
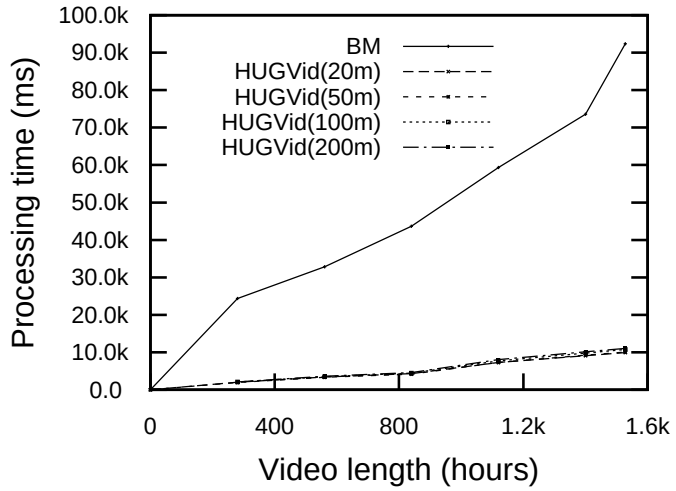


Figure 4.11: Comparison of the index sizes between BM and HUGVid with different micro-block sizes.

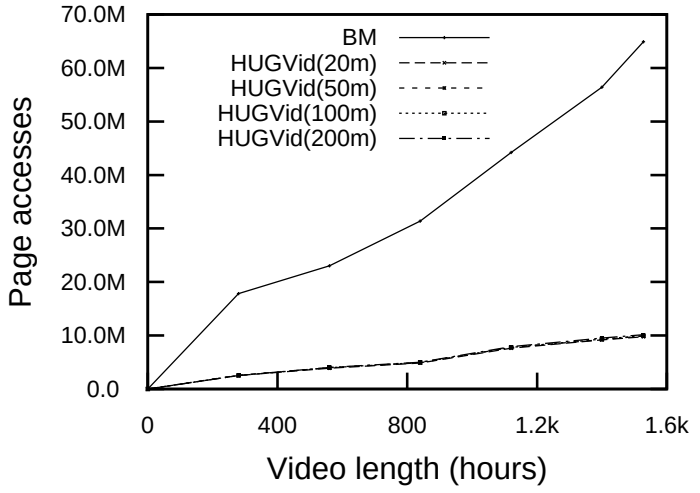
query types that our approach supports, the system workload for range queries is the heaviest. Therefore, we use range queries to present the performance of HUGVid on the large-scale dataset. In this experiment, the page and cache sizes are set to 4 kB, and we store one R-tree node per page. We generated 10,000 range queries of 500 m $\times$ 500 m rectangles within the 75 km $\times$ 75 km test region and counted the cumulative processing time and the overall number of page accesses for answering 10,000 queries.

We conducted experiments with different video lengths. Figure 4.11 shows the in-memory index size of different methods with different test sets. The index size of all the methods grows linearly, but the rate for HUGVid is much smaller than for BM. Although the video segmentation is carried out using the Monte Carlo method with different micro-block sizes, the segmentation is still mostly related to the spatial properties of the video itself. Hence the in-memory index sizes of HUGVid with different micro-block sizes are almost the same. The main difference is in the storage of the secondary index: it occupies more disk space when the micro-block size is smaller. However, as stated in Section 4.5.1, one extra benefit is that it achieves more accurate results. As shown in Figure 4.12, HUGVid with

different micro-block sizes performs faster and accesses fewer pages than BM. Even when HUGVid involves video recombination and video ranking, it still answers the queries quickly, with an execution time of only about 12% of BM. We conclude from these experiments that HUGVid performs well on this large-scale dataset. Moreover, it is beneficial to select a relatively small micro-block size while the exact value may depend on different applications.



(a) Processing time.



(b) Page accesses.

Figure 4.12: Comparison of processing time and page accesses between BM and HUGVid with different micro-block sizes.

4.6 Summary

We explored the challenge introduced by naturally noisy data from GPS and compass sensors, as well as the possible obstacles appearing in the video scenes, which can result in inaccurate geo-descriptions of video scenes. This, in turn, may lead to undesirable query results for geo-tagged video searches. To address this issue, we proposed an uncertain data model to represent individual and sequences of field-of-views and finally constructed a light-weight approximate model for video segments based on sensor meta-data. With this architecture, probabilistic queries can be executed and upstream GIS tasks prioritized based on the most promising results. Experiments show that HUGVid achieves high recall and can be deployed in large-scale applications.

Compared from the user study results, a drawback of this work is that the ranked query results did not exactly match users' preference when querying large volume building. Therefore, we plan to improve the video ranking algorithm so as to satisfy users' perspective. Moreover, It is still difficult to evaluate the probability when obstacles appear. As a result, another possible future work is to detect obstacles utilizing content-based method.

Chapter 5

Scheduling of Video Transcoding for DASH in a Cloud Environment

5.1 Introduction

Recently, Over-The-Top (OTT) streaming, *i.e.,* delivering video and audio content through the public Internet infrastructure rather than proprietary infrastructures such as cable networks, has become an active topic in the broadcasting and content delivery communities. OTT in particular refers to content that arrives from a third party, such as Netflix [1] and Hulu [2], and is delivered to an end-user device, leaving the Internet provider responsible only for transporting packets. The final link to end-users is usually handled with HTTP streaming, or other proprietary technologies. Consumers can access OTT content through various Internet-connected devices such as desktops, laptops, tablets, smartphones, TVs and gaming consoles (*e.g.,* Xbox 360, PlayStation, Wii). The variety of devices requires the video hosting services to provide different bitrates of the original videos.

Moreover, the wide-spread availability of smartphones (and increasingly tablets) and the rapid improvement of wireless networks (3G/4G, or WiFi) have enabled the feasibility of frequent video streaming through mobile devices. An important consideration is that the bandwidth for mobile devices varies depending on location and time. The changes in bandwidth influence the quality of video streaming. For exam-

[1] http://www.netflix.com/
[2] http://www.hulu.com/

ple, assume that all available bandwidth of a mobile client is used to watch a video. When the bandwidth increases, it has the capacity to watch a higher quality video. Conversely, when the bandwidth decreases, the playback would be interrupted due to insufficient bandwidth for the current bitrate. Therefore, to guarantee smooth playback and enable streaming of the highest possible quality, it is essential that media streaming can adapt to the current network bandwidth and conditions. Dynamic Adaptive Streaming over HTTP (DASH) is designed to provide high quality streaming of media content over the Internet delivered from conventional HTTP web servers. Figure 5.1 illustrates the architecture of DASH. Media content is encapsulated into a parallel sequence of segments with a number of different bitrates so that an MPEG DASH client can automatically and seamlessly select the next segment to download and play back based on current network conditions.

The popularity of video streaming has highlighted video transcoding as a challenging problem. In recent years, cloud computing has become an effective paradigm for many applications. It is a technology aimed at sharing resources and providing various computing and storage services flexibly over the Internet. For multimedia applications and services, there are strong demands for cloud hosting because of the significant amount of computation required for serving millions of Internet and mobile users simultaneously [Zhu 2011]. The complex nature of video transcoding (*e.g.,* CPU-intensity) and the high demand requirements of streaming have enabled cloud computing to be uniquely suitable for video transcoding, especially in the context of large-scale video hosting systems. Therefore, transcoding is preferably executed in a powerful cloud environment, rather than on the source computer (which may be a mobile device with limited memory, CPU speed and battery life).

In order to support live streaming of media events and to provide a satisfactory user experience, the overall *video transcoding completion time* should be minimized. In the rest of the chapter we refer to this simply as the video completion time or the completion time. Minimizing the video completion time is desirable for sev-

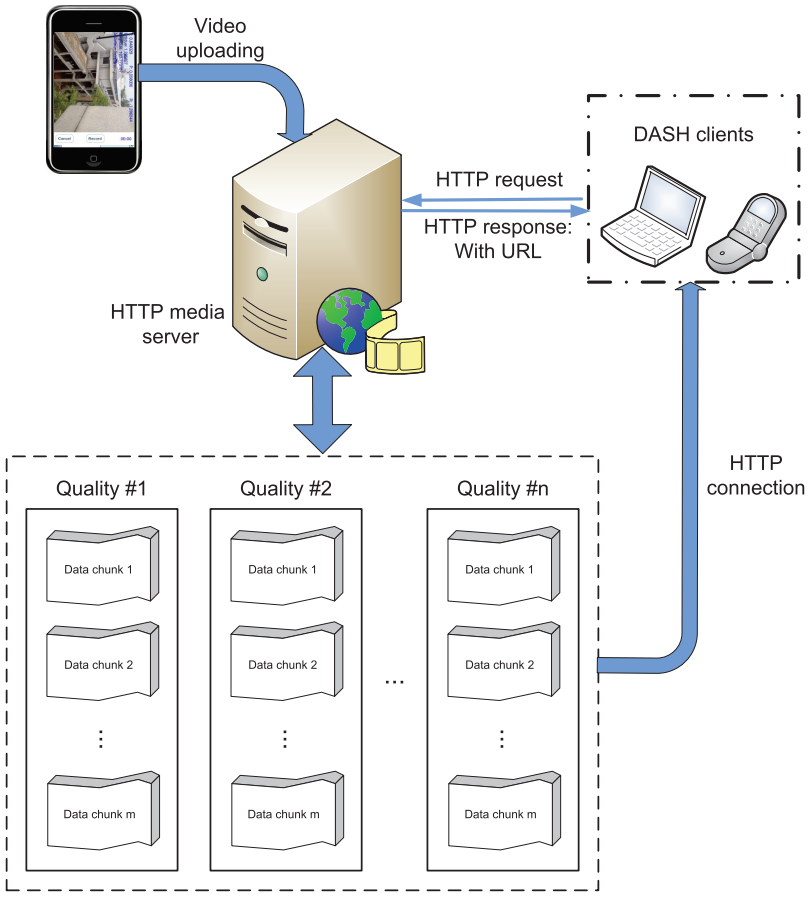


Figure 5.1: A general architecture of DASH.

eral reasons. Obviously it is crucial not to exceed the waiting tolerance of clients. Because of the high computational complexity, scheduling video transcoding jobs in a cloud environment to minimize the processing latency and satisfy users' expectations is a challenging problem. Furthermore, it is important to balance the workloads among all the processing nodes in the cloud. In this work we propose a dynamic scheduling algorithm for DASH video transcoding designed for cloud environments. The scheduler makes use of the estimation of the video transcoding time (VTT). Jobs are distributed to free processors when they are not urgent, but to the fastest processors if video watching requests are pending. Experimental results show that the scheduler performs very well in executing video transcoding jobs and balancing the workload among all the processors. The main contributions of our

method are summarized as follows:

- We introduce a video transcoding time (VTT) estimation, with respect to video segment duration and targeted bitrates, that is modeled based on measured statistics and probabilistic theory.
- We incorportate a job distrmechanism among processors that achieves load balancing among the processing resources.
- The scheduler includes workload and status monitoring for each processor, and dynamically selects the fastest processors to run high-priority jobs when videos have pending viewing requests.
- The scheduler dynamically optimizes the video transcoding mode (VTM) when the number of processors is insufficient to support all video watching requests, so that the processors prioritize transcoding of the requested bitrates and leave other bitrates to be processed later.

The rest of this work is organized as follows. Section 5.2 models the VTT estimation methodology considering the video duration. The scheduling algorithm and evaluation metrics are detailed in Section 5.3. Section 5.4 presents the experimental results and analysis. Finally, Section 5.5 summarizes this work.

5.2 Transcoding Time Estimation

We first introduce our approach of estimating the video transcoding time (VTT) from the original stream to other qualities, *i.e.,* different bitrates. For DASH, the server needs to prepare multiple bitrates and multiple formats of the originally uploaded videos. Since the time for the video transformatting jobs is only a few milliseconds, we focus on scheduling the video transcoding jobs in this work. Without loss of generality, we consider two reduced quality streams, namely encoding the original video segments at a medium bitrate (768 kbps with resolution of 480×360) and low bitrate (256 kbps with resolution 360×240), respectively. The scheduling

algorithm is also applicable to a larger number of targeted bitrates. The scheduler will hence distribute video transcoding jobs (VTJ) based on the estimation of VTT. The VTT includes the sum of the following two parts: the time for the file transfer between the storage repository and the processing nodes and the time for the actual transcoding procedure. In order to model the estimation of VTT given the video duration, we measure the actual VTT (denoted as VTT_{mea}) for statistics on a set of video segments in the cloud environment.

5.2.1 Configuration of the Cloud Environment and Description of the Testing Dataset

To test the performance of the scheduler in different environments, we set up two different cloud environments: ***Cloud1*** and ***Cloud2***. Cloud1 is a shared cloud environment where the used processors could also be accessed by other users and used to run other CPU or memory intensive jobs. It includes one master node and 25 processing nodes. All the nodes are running CentOS 5.5 and can access a shared storage system. The master node is used to run the scheduler with two quad core Intel® Xeon® $E5440$ 2.83 GHz CPUs and 16 GB of memory. The processor nodes used to run $VTJs$ consist of two quad core Intel® Xeon® $E5620$ 2.4 GHz CPUs and 24 GB memory.

Cloud2 is a private cloud environment where all the computing resources can be fully utilized by the scheduler. It includes ten commodity PCs connected with a high speed gigabit network, where one PC was the master node and the others are the processing nodes. Each PC contains an Intel® Quad Core® 2.66GHZ CPU, 4GB memory and is running under CentOS Linux 5.6.

The testing dataset used to measure the parameter of the cloud environment includes $11,194$ video segments from 339 videos which we collected with Android phones and are coded in MPEG-4 with resolution 720x480. These videos are segmented at the mobile clients and uploaded to remote servers [Seo 2012]. For smooth

and seamless rendering of the segments, each contains an integral number of Group-Of-Pictures (GOP). In the configuration of the mobile application, the segment durations were chosen as 3, 4 or 5 seconds, and the duration of the last segment of each video varies from about 0.2 second to 6.5 seconds. On the remote servers, all the segments are transcoded and transformatted with the open source software FFmpeg [ffm].

5.2.2 VTT Estimation Methodology

To calculate the VTT of encoding individual segment to different bitrates in different cloud environment, the VTJs are processed in both Cloud1 and Cloud2. In order to avoid the effect of caching on the $VTJs$, we deployed the VTJ for low and medium bitrates on different processors, which could be fully utilized without being occupied by other jobs. To minimize the runtime bias (*e.g.,* the variability of the connection speed between processors and the storage repository, and individual processing time) on $VTJs$, we ran these jobs 10 times across different processors and calculated the mean values of VTT. Figure 5.2 presents the measured VTT to low and medium bitrates, respectively, as well as the corresponding fitting curves (denoted as VTT_{cal}), with respect to the video segment duration. We use Matlab® to apply the curve fitting procedure. We tried different types of fitting curves and found that the standard deviation on the VTT between the estimated value from the power fitting curve and the measured one from statistics is minimum, compared to other fitting curves. Therefore, we choose the power fitting curve to estimate the VTT with respect to the video duration. The fitting curves can be calculated as formulated in Equation 5.1.

$$VTT_{cal}\left(dur(V_{B_k})\right) = a \cdot \left(dur(V_{B_k})\right)^b \tag{5.1}$$

where $dur(V_{B_k})$ indicates the video duration, and a and b are the fitting coefficients. In the current configuration of Cloud1, the values of a and b are shown in

Equation 5.2:

$$[a\ b] = \begin{cases} [0.798 \quad 0.621] & (low) \\ [1.152 \quad 0.700] & (medium) \end{cases} \tag{5.2}$$

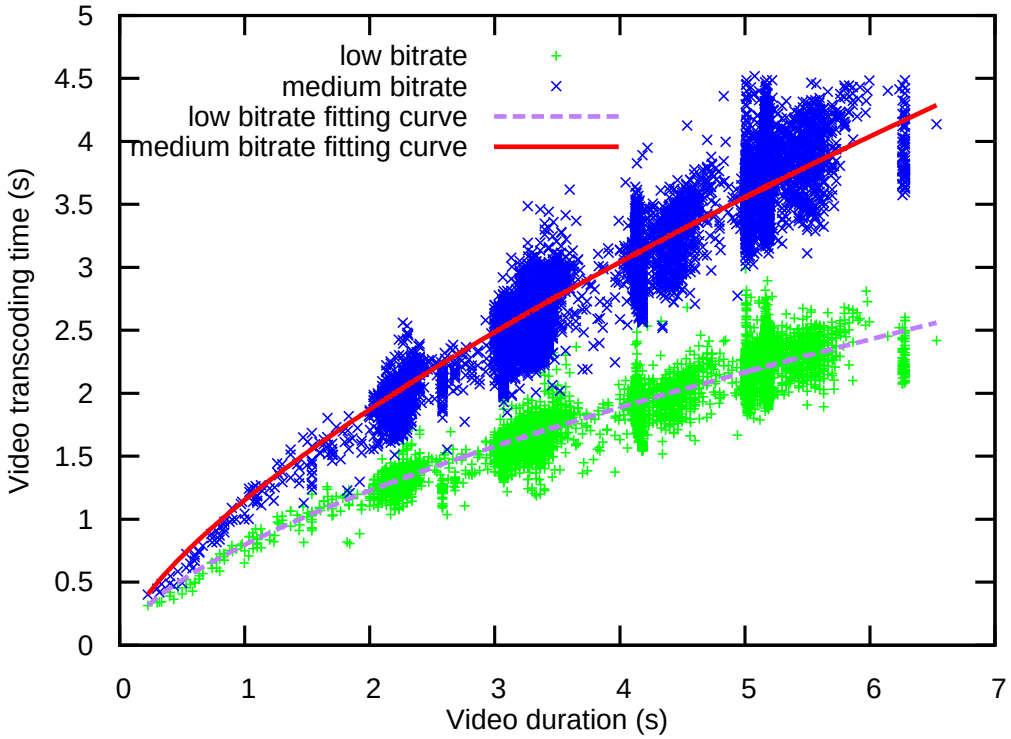


Figure 5.2: VTT statistics and the fitting curves to different bitrates with respect to video duration in Cloud1.

As shown in Figure 5.2, the actual VTTs are always different from the value calculated by the fitting curve for most of the time. In order to match the VTT estimation with the measurements, we calculate the bias of the measured VTT with respect to the calculated VTT. The normalized error for each VTT is calculated as shown in Equation 5.3:

$$T_{err} = \frac{VTT_{mea} - VTT_{cal}}{VTT_{cal}} \tag{5.3}$$

The distributions of the value of T_{err} follow a Gamma distribution (shown in Equation 5.4):

$$Prob(\widetilde{T_{err}}) \sim \frac{1}{G_\theta^{G_k}} \cdot \frac{1}{\Gamma(G_k)} \cdot \widetilde{T_{err}}^{G_k - 1} \cdot e^{-\frac{\widetilde{T_{err}}}{G_\theta}} \tag{5.4}$$

where $\widetilde{T_{err}} = round(T_{err} \cdot 100)$. Overall, we calculate $Prob(\widetilde{T_{err}})$ over the whole dataset for each individual VTT and compute the value of G_k and G_θ, which are the coefficients of the Gamma distribution (Table 5.1):

Table 5.1: The coefficients of Gamma distribution in Cloud1.

	G_k	G_θ
Low bitrate	8	2.4
Medium bitrate	5.8	3.4

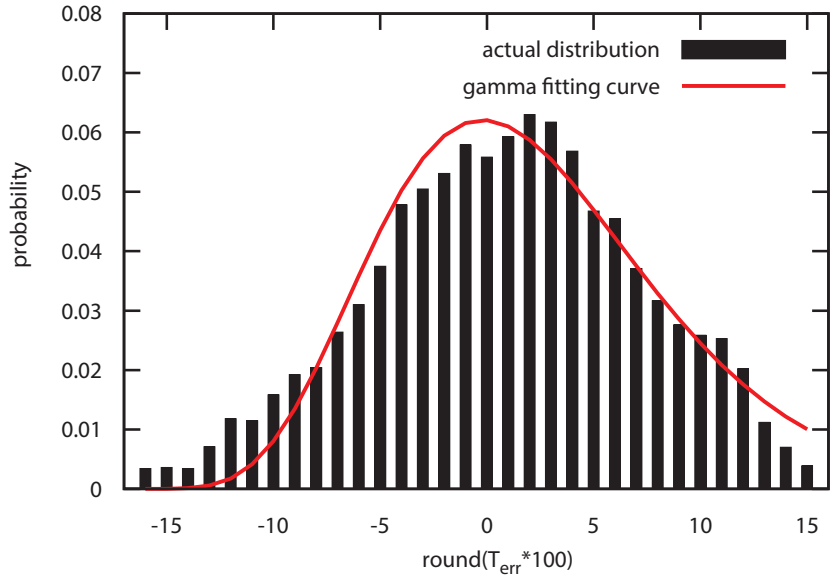
Based on this, the estimated VTT can be calculated as:

$$VTT_{est} = VTT_{cal} + VTT_{err} = VTT_{cal}(1 + T_{err}) \tag{5.5}$$

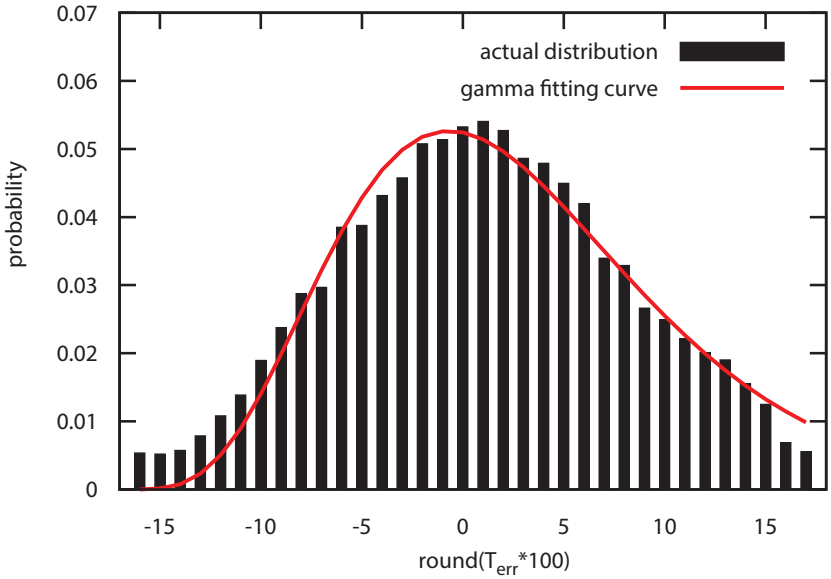
Figures 5.3(a) and 5.3(b) illustrate the distribution of $\widetilde{T_{err}}$ to low and medium bitrates, respectively.

Ideally, when a processor is not running other jobs, the VTT of a given segment follows the calculation in Equation 5.5. However, due to various reasons (*e.g.,* shared usage of CPU and/or memory, as well as the congestion of the network), the actual VTT might be significantly different from the estimated value. Since the scheduler works based on the VTT estimation, this situation would affect the performance of the scheduler, especially when processing videos with viewing requests. Practically, the scheduler compares the actual VTT with the estimated value and treats the difference between these two values as the feedback from the processor that ran the VTJ. This feedback helps to reflect the current working status (*e.g.,* indicating the transcoding speed and the workload of that processor) of that processor. For example, if the actual VTT is twice of the estimated VTT, it is considered as that only half of the computing resources from that processor can be utilized to run the VTJ. The scheduler will thus distribute VTJs, also considering the feedback from each of the processors until a new feedback is obtained.

We also follow the same method introduced above to calculate the VTT of



(a) Low bitrate.



(b) Medium bitrate.

Figure 5.3: The distribution of $\widetilde{T_{err}}$ in Cloud1.

encoding individual segment to different bitrates in Cloud2. The coefficients of the fitting curves and the Gamma distributions of Cloud2 are summarized in Table 5.2, and the VTT statistics and fitting curves to different bitrates are shown in Figure 5.4

Table 5.2: The coefficients of the fitting curves and the Gamma distributions used in Cloud2.

	low bitrate	medium bitrate
a	0.3068	0.4523
b	0.5655	0.518
G_k	7.8	2.5
G_θ	5.5	3.5

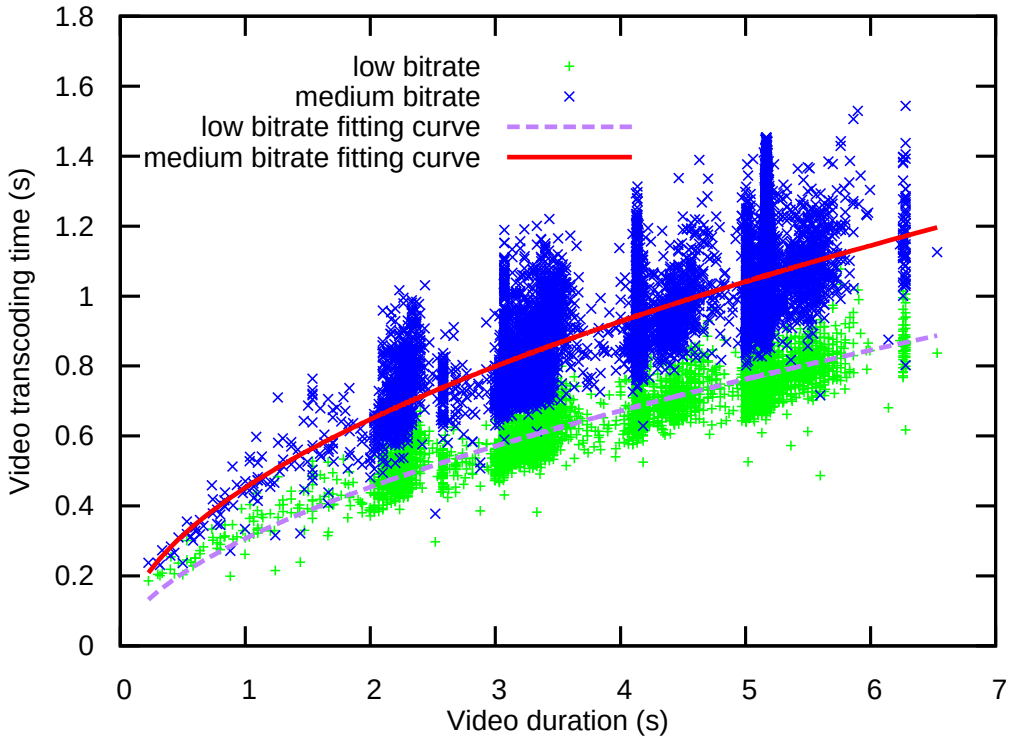


Figure 5.4: *VTT* statistics and the fitting curves to different bitrates with respect to video duration in Cloud2.

5.3 Scheduling Algorithm

This section introduces the dynamic scheduling algorithm for video transcoding with DASH in a cloud environment. Figure 5.5 shows the overall architecture of the framework. On the master node, the job scheduler maintains two queues: one queue keeps all the normally uploaded jobs (referred as *NQueue*), while the other one maintains the jobs with high-priority (referred as *PQueue*). When a video is uploaded to the front-end web interface (referred to as a *job*), a video transcoding

request is generated by the web interface and forwarded to the master node. This job is then inserted into NQueue. At the same time, the uploaded video is stored in the video storage repository. The job scheduler assigns jobs once any of the processors is available according to its scheduling strategy. When the processor gets VTJ from the job scheduler, it loads the targeted segment from the storage repository and saves the encoded segments back after finishing the VTJ. When a mobile client sends the video retrieval request (*i.e.,* for viewing) to the web interface, the web interface checks whether the required video segment is available for viewing. If so, an HTTP connection is set up between the mobile client and the storage repository for streaming. Otherwise, the request is converted to a high-priority job for the targeted video and this job will be inserted into PQueue. Note that the job scheduler keeps monitoring the transcoding speed of each processor according to the feedback. It will then assign enough number of processors, which are the fastest among the ones run $VTJs$ from NQueue, to process $VTJs$ from PQueue. The calculation on the number of processors needed to satisfy the job migration from NQueue to PQueue is detailed in Section 5.3.2.

5.3.1 Evaluation Metrics

To evaluate the performance of the scheduler, the following evaluation metrics are considered.

Startup latency (L_s): This metric measures the time interval from when the video segment is started to be transcoded until the video is available for playback.

Number of quality switches (N_{QS}): The playback will switch to other bitrates if the subsequent segment of the current bitrate is not available, or the network conditions change.

Number of rebuffering events (N_{RE}): The video client has to pause the playback of both audio and video during rebuffering if there exists no lower bitrate

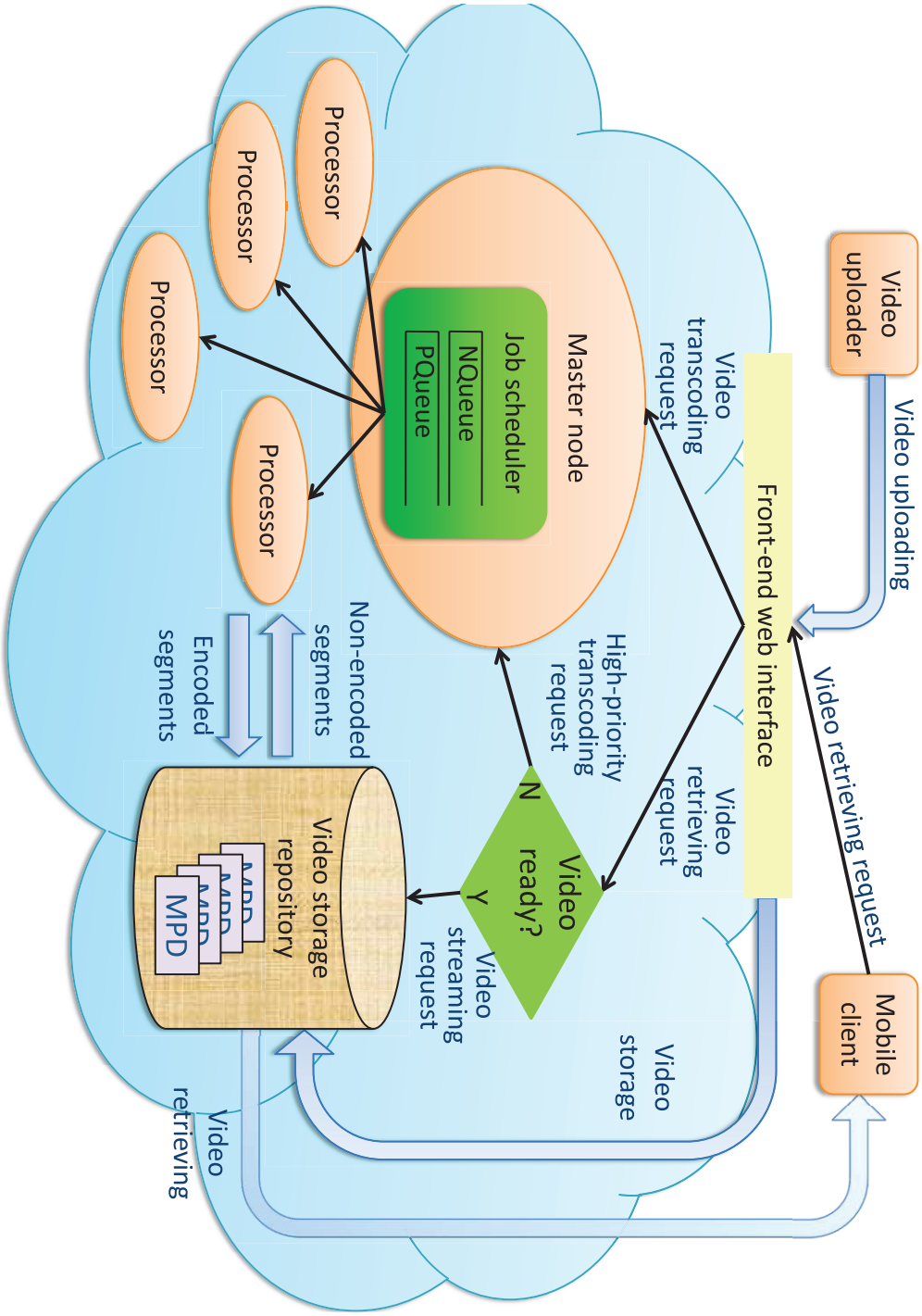


Figure 5.5: The architecture of scheduling video transcoding tasks for DASH in a cloud environment.

to switch to.

Mean rebuffering time (T_{mr}): During rebuffering, the video is paused until the video can be restarted. The duration of the rebuffering period varies. The longer the rebuffering period is, the worse the video streaming performs.

Mean Opinion Score (MOS): We apply the calculation from Mok *et al.* [Mok 2011] to express users' quality of experience (QoE). The metric is based on a regression analysis to acquire the relationship between QoE and the application quality of service (QoS). The MOS can be calculated as shown in Equation 5.6:

$$MOS = 4.23 - 0.0672L_s - 0.742(N_{QS} + N_{RE}) - 0.106T_{mr} \tag{5.6}$$

Load Balance Factor (LBF): This measures the balance among different processors. We use the standard deviation of the overall VTT of all the processors normalized by the average total VTT to estimate the load balance factor.

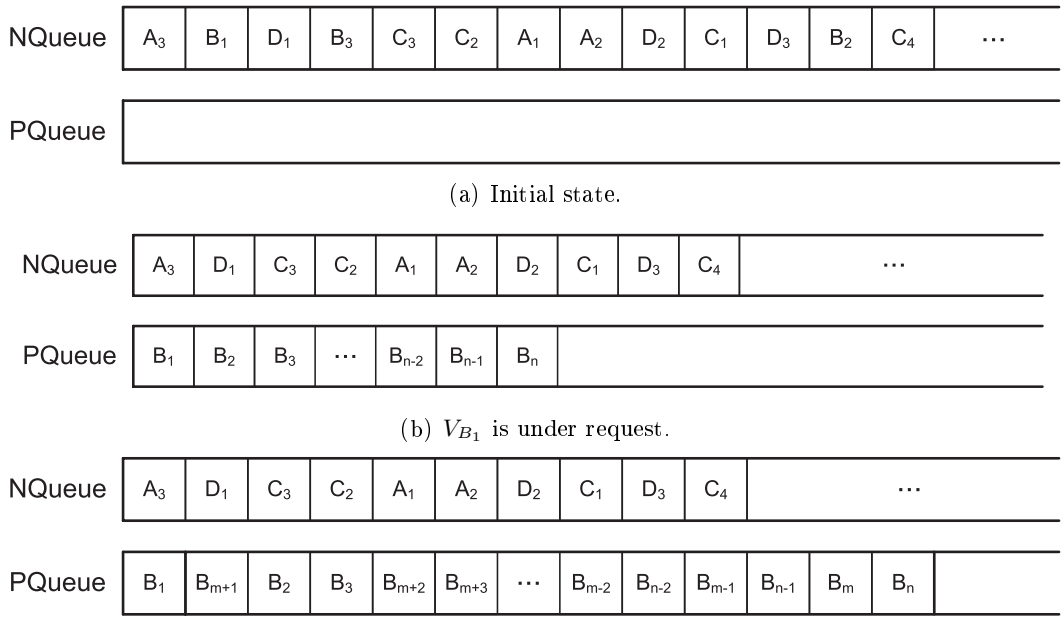
$$LBF = \sqrt{\frac{1}{N_{pro}} \sum_{i=0}^{N_{pro}-1} \left(T_{pro(i)} - \frac{1}{N_{pro}} \sum_{i=0}^{N_{pro}-1} T_{pro(i)} \right)^2} \Bigg/ \left(\frac{1}{N_{pro}} \sum_{i=0}^{N_{pro}-1} T_{pro(i)} \right) \tag{5.7}$$

where N_{pro} denotes the number of processors in the cloud environment and $T_{pro(i)}$ indicates the overall running time of processer i.

5.3.2 Scheduler

As stated in Section 5.3, the job scheduler on the master node maintains two queues: *NQueue* and *PQueue*. All jobs are initially inserted into NQueue, and jobs are migrated to PQueue only if the corresponding videos are requested for watching. Figure 5.6 illustrates the switching of jobs between NQueue and PQueue. Figure 5.6(a) shows an example of the initial state of NQueue and PQueue at time t. Initially, Videos A to D are uploaded to the server and are waiting to be transcoded. Utilizing

the methods introduced in Section 5.2, jobs in NQueue are sorted according to their estimated VTT ascendingly (shown in Algorithm 8). Once new video segments are uploaded to the server, the scheduler calculates the estimated job completion time and inserts the related job into NQueue at the correct location. In our current approach, we publish the video to be available for watching when the beginning L_{va} seconds of a video are transcoded. As shown in the example of Figure 5.6, once Segment V_{B_1} is under request, all of its subsequential segments from Video B (V_B) are assigned to be high-priority and transferred to PQueue (shown in Figure 5.6(b) and lines 10-14 of Algorithm 9). Note that the jobs in PQueue are sorted by the their deadlines ascendingly. When one of the subsequent segments (*e.g.*, $V_{B_{m+1}}$) is required by another client, itself and the subsequent jobs are reinserted into PQueue according to the updated deadlines (shown in Figure 5.6(c) and lines 15-18 of Algorithm 9).



(a) Initial state.

(b) V_{B_1} is under request.

(c) Another video retrieval request on $V_{B_{m+1}}$

Figure 5.6: Illustration of inserting jobs into the NQueue and PQueue.

Definition 1 *The deadline of a job.*

When a video segment is requested to be watched by end-users, its corresponding and

subsequent jobs are selected as a high-priority jobs. To guarantee smooth playback for video streaming, the VTJ needs to be accomplished before its previous segment is played to the end. The predicted time for its previous segment to be played to the end is defined as the deadline of a job (referred as Job_{DL}). As presented in the above paragraph, we publish a video as available for watching after the beginning L_{va} seconds of the video are transcoded. Therefore, Job_{DL} equals the cumulative video duration of all its previous segments. Considering the example of Figure 5.6(b), Equation 5.8 shows the calculation of Job_{DL} given a required job on video segment V_{B1} and all its subsequent segments.

$$Job_{DL}^{B_i} = \sum_{k=1}^{i-1} (dur(V_{B_k}) \quad (i > 1) \tag{5.8}$$

where $dur(V_{B_k})$ is the duration of V_{B_k}.

Algorithm 8: videoUpload()

Input: An uploaded video segment V_{Ij}
J_{ID} = assignID(V_{Ij}); `// assign job ID to` V_{Ij}
t_{est} = timeEst(duration of V_{Ij}); `// transcoding time estimation`
insert2NQ(J_{ID}, t_{est}); `// insert into NQueue according to time by SJF`

Algorithm 10 presents the proposed scheduling strategy. The basic idea is to transcode the jobs in both NQueue and PQueue simultaneously, guaranteeing the smooth playback for videos under request. A segment can be transcoded with five different modes (denoted as VTM): trancoding to low and medium bitrates simultaneously (0); transcoding to low bitrate first and then medium (1); transcoding to medium bitrate first followed by low (2); only transcoding to low bitrate (3); and only transcoding to medium bitrate (4). If there exist no video viewing requests, then all the processors are assigned to run jobs of NQueue. Jobs in NQueue are transcoded with VTM setting 0. The number of processors necessary to run a high-priority job needs to satisfy the following condition: the VTT of the next segment should be smaller or equal to the playback time (*i.e.,* video duration) of its previ-

Algorithm 9: videoUnderRequest()

Input: Video segment V_{Ij} is under request,
The request bitrate is B_{req}
for $j \leq \widehat{j} \leq m$ **do**
  J_{ID} = getID($V_{I\widehat{j}}$); `// get the job ID`
  if J_{ID} *is accomplished* **then**
    send the link to the HTTP-client;
  else if J_{ID} *is being processed* **then**
    wait until J_{ID} is accomplished;
    send the link to the HTTP-client;
  else if J_{ID} *is in NQueue* **then**
    t_{dl} = calDL(J_{ID}); `// calculate the deadline of` J_{ID}
    insert2PQ(J_{ID}, B_{req}, t_{dl}); `// move` J_{ID} `to PQueue`
    removeNQ(J_{ID}); `// remove` J_{ID} `from NQueue`
  else
    `/*` J_{ID} `is in PQueue */`
    t_{dl} = calDL(J_{ID});
    updatePQ(J_{ID}, B_{req}, t_{dl}); `// reinsert` J_{ID} `into PQueue and remove the old entry`

ous segment. To minimize the bias on VTT estimation, we assign one additional processor to run each high-priority job. For example, when one video V_{B_1} is under request, the scheduler calculates the number of processors (referred to as N_{pr}) needed in order to ensure smooth playback (shown in Equation 5.9) and decides to follow VTM (chosen from 1 or 2) according to which bitrate of video is under request. Only when not enough processors are available to run high-priority jobs will the scheduler switch the VTM ($1 \rightarrow 3$, $2 \rightarrow 4$ accordingly) to minimize the VTM, prepare for the required bitrate first and leave the other bitrates afterwards when the system is under a lighter workload.

$$N_{pr} = \left\lceil \frac{timeEst(dur(V_{B_k}))}{dur(V_{B_{k-1}})} \right\rceil + 1 \quad (k > 1) \tag{5.9}$$

The scheduler will then assign this number of processors to run jobs in PQueue when they are available. The scheduler keeps monitoring the work status of all the processors, as well as the jobs in NQueue and PQueue, and adjusts the processors

to run jobs in either NQueue or PQueue according to the current workload. For practical purposes, the scheduler also monitors the system load (*e.g.,* CPU and memory usage) of each processor and finds the fastest processor to run jobs in PQueue.

Algorithm 10: scheduler()

Input: Processor lists: $ListPQ\{\}$, $ListNQ\{\}$
job queues: $NQueue\{\}$, $PQueue\{\}$
for *all available processors* **do**
    $ListNQ$.insert(P_{ID}); `// assign all processors to run jobs in NQueue`
num_job = # of requested jobs;
num_pro = cal_pro(num_job, VTM); `// calculate the number of processors reserved to run high-priority jobs for smoothly playback`
$i = 0$;
`/* reserve` num_pro `processors to run high-priority jobs */`
while $i \leq num\_pro$ **do**
    find the fastest P_{ID};
    if P_{ID} *in NQueue is available* **then**
        $ListPQ$.insert(P_{ID});
        $ListNQ$.delete(P_{ID});
    i++;
if *not enough number of* P_{ID} **then**
    switch VTM according to the required bitrate;
    num_pro = cal_pro(num_job, VTM);
while !$NQueue.isEmpty()$ // !$PQueue.isEmpty()$ **do**
    if $\exists P_{ID}$ *in ListNQ is available* **then**
        assign(P_{ID}, $NQueue$.pop());
    if $\exists P_{ID}$ *in ListPQ is available* **then**
        assign(P_{ID}, $PQueue$.pop());

5.4 Experimental Evaluation

To test the performance of the scheduler in different environments, both Dataset4 and Dataset5 are used.

5.4.1 Experiment with Dataset4

We report on the performance of the scheduler with respect to segment completion time, MOS, load balance and playback smoothness with Dataset4.

5.4.1.1 Experiment Configuration

To test the functionality whether the scheduler can dynamically identify the fastest processor to run high-priority jobs, all the experiments were conducted in Cloud1. The used processors could also be accessed by other users and used to run other CPU or memory intensive jobs. The configuration of Cloud1 were earlier presented in Section 5.2.1. We used Dataset4 to test the performance of the proposed scheduling algorithm.

5.4.1.2 Performance Evaluation

In the following experiments, we use two scenarios which commonly happen among video hosting services: (1) the videos (*e.g.,* historical videos) are only uploaded to the server and none of them is being watched before all alternative bitrates are transcoded; and (2) the videos (usually news clips and live sports events) are watched shortly after being uploaded and not all required bitrates are prepared yet. Table 5.3 summarizes the parameters used in the experiments.

Scenario 1: No video watching requests. In this scenario, we tested the performance of the scheduler on the process of the videos being uploaded to the cloud and none of them being requested for watching by end-users. Since no video watching requests arrived, all the segments are transcoded to medium and low bitrates simultaneously, which means that each segment needs to be transferred between the storage repository and the processors once. The video uploading streams are generated by M_i video uploaders and each of them submits one segment every Δt seconds until all the segments are uploaded to the server. The arrival time of the first uploaded segment from each uploader follows a Poisson distribution with mean

Table 5.3: Parameters used in the experiments to analyze the system capacity in Cloud1 with Dataset4.

Parameters	Configurations
Stream 1 (uploading)	MPEG-4 video, AAC audio
Number of uploaders (M_{up_1})	6
Uploading frequecy (Δt)	1 segment per second
Stream 2 (uploading)	MPEG-4 video, AAC audio
Number of uploaders (M_{up_2})	7
Uploading frequecy (Δt)	1 segment per second
Stream 3 (uploading)	MPEG-4 video, AAC audio
Number of uploaders (M_{up_3})	8
Uploading frequecy (Δt)	1 segment per second
Mean inter-arrival time (λ)	50 seconds
Video Trans. Manner (VTM)	0: low and medium bitrates together 1: first low then medium bitrate 2: first medium then low bitrate 3: only low bitrate 4: only medium bitrate
Video available latency (L_{va})	5, 10, 15 seconds
Targeted video bitrates (Bit_r)	
Medium bitrate	480×360, 768 kbps
Low bitrate	360×240, 256 kbps
Request arrival rate (N_{wr})	5 per second

inter-arrival time of $\lambda = 50$ seconds.

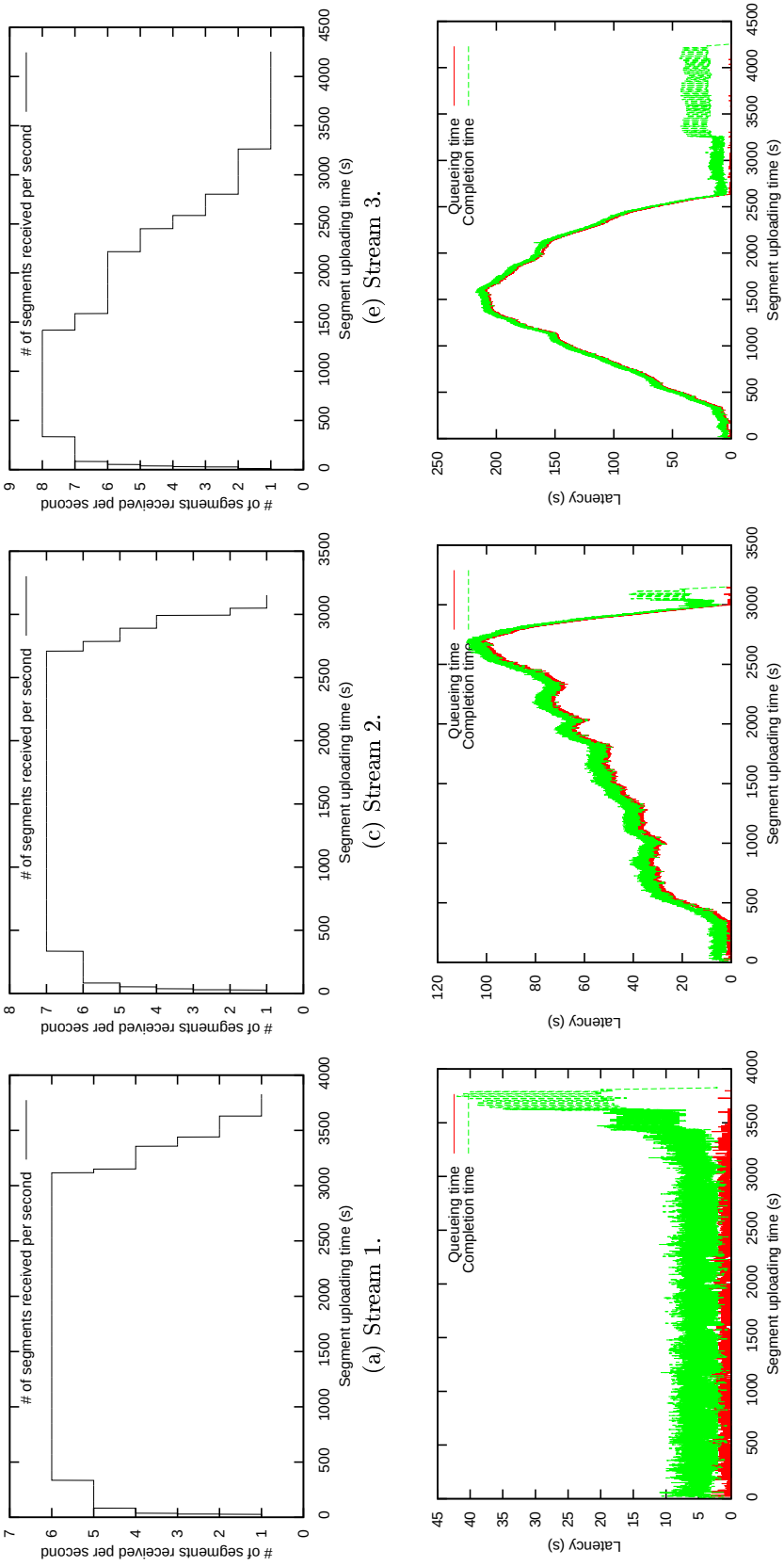
We first analyze the system capacity by comparing the latencies of uploading Streams 1 to 3. Figures 5.7(a), 5.7(c), and 5.7(e) show the number of segments received at the server side per second, and for each stream, the server works at the maximum workload for a period of time. Figures 5.7(b), 5.7(d), and 5.7(f) show the latency (both the queueing time and the completion time) of each uploaded segment for Streams 1 to 3, respectively. When the maximum arrival rate of segments is 6 (Figure 5.7(b)), the queueing time of each segment is fairly small and the completion time also remains less than 10 seconds during the first 50 minutes. We can conclude that receiving six segments per second is within the capacity of the current system, and the scheduler can support near-live streaming because of the small segment completion latency. Figure 5.7(d) shows that when the maximum

arrival rate increase to 7, both the queuing time and the completion time for the segments grow. This indicates that the workload in Stream 2 exceeds the system capacity. The uploaded segments start to queue up and the queue size grows as time elapses. Compared with the latency when the arrival rate is 7, Stream 3 (Figure 5.7(f)) overloads the system more as the growth rate of the latency is larger than that of Stream 2. The latency decreases as soon as the workload is smaller than 7. One important observation is that there exist two situations of completion time increase: (a) the completion time increases with the same pattern as the queueing time (shown in Figures 5.7(d) and 5.7(f)), and (b) the completion time increases while the queueing time remains a small and relatively constant value (illustrated in the last minutes of Figures 5.7(b), 5.7(d), and 5.7(f)). Situation (a) is due to the growing queue size as the segment uploading rate is larger than the system capacity and hence it represents an accumulation of the previous transcoded segments. On the other hand, situation (b) is due to the long duration of VTT for specific segments. For all these three streams, VTT of the last segments are almost the same. Consequently, the system capacity is sensitive to the VTT of each segment. The scheduler needs to be smart enough to find the fastest processors to run urgent jobs.

Table 5.4: Statistics on LBF for each stream in Cloud1 with Dataset4.

Stream	LBF
Stream 1	0.0199
Stream 2	0.0036
Stream 3	0.0132

We next present the load balancing results of the system. Table 5.5 illustrates the statistics of all processors to transcode videos in Stream 2 as an example. From the table, we observe that although the number of jobs completed by each processor varies from 550 to 946, the overall VTT for each processor only differs little. Some processors execute more jobs and transcode a higher duration of videos than others. The same situation occurs when uploading Stream 1 and Stream 3. We can conclude



(a) Stream 1. (b) Latency for Stream 1. (c) Stream 2. (d) Latency for Stream 2. (e) Stream 3. (f) Latency for Stream 3.

Figure 5.7: Experimental results for uploading Streams 1 to 3 with Scenario 1 in Cloud1 with Dataset4.

that the scheduler performs well with respect to load balancing. Table 5.4 summarizes the value of LBF for processing Streams 1 to 3. When comparing the LBF of these three streams, the scheduler achieves the best load balance when transcoding Stream 2. It is inferred that the scheduler works well when the system reaches its full capacity, and the performance decreases by a small percentage when the system is less or more loaded.

Table 5.5: Statistics on each processor for Stream 2. The similarity of the overall VTT shows that the workloads among processors are well balanced, while the normalized VTT and the overall duration differentiate the processing capacity of each individual processor in Cloud1 with Dataset4.

Node ID	# of jobs	Overall VTT (s)	Mean normalized VTT (s)	Median normalized VTT (s)	Overall duration (s)
1	810	2963.9	0.889	0.811	3628.2
2	832	2965.2	0.857	0.788	3737.4
3	718	2973.4	1.016	0.913	3183.2
4	860	2957.9	0.832	0.745	3874.6
5	946	2954.4	0.745	0.682	4278.3
6	797	2968.1	0.927	0.823	3543.2
7	777	2967.5	0.917	0.830	3498.9
8	921	2951.6	0.797	0.701	4110.8
9	725	2958.0	0.993	0.898	3234.5
10	714	2957.7	1.003	0.910	3163.8
11	727	2964.9	1.005	0.894	3238.7
12	604	2973.3	1.181	1.101	2708.9
13	863	2948.2	0.819	0.739	3909.2
14	854	2944.4	0.815	0.753	3842.5
15	657	2965.7	1.078	0.980	2938.4.
16	809	2950.8	0.887	0.798	3629.4
17	846	2945.6	0.835	0.773	3801.3
18	898	2948.9	0.797	0.720	4020.1
19	926	2948.7	0.794	0.703	4121.2
20	886	2958.2	0.810	0.731	3973.2
21	887	2952.1	0.812	0.728	3936.9
22	648	2971.3	1.106	1.028	2907.8
23	550	2988.2	1.283	1.193	2483.7
24	833	2960.3	0.845	0.777	3782.1
25	912	2943.9	0.775	0.701	4085.6

Scenario 2: Videos are requested for watching while they are being

uploaded to the server. In this scenario, we study the performance of the scheduler while some of the videos are requested for watching before all the segments are transcoded. One assumption is that the clients do not interact with the videos during playback, such as pausing and forward/backward seeking. This means that each video is watched from the beginning to the end. Note that the processors in Scenario 2 are working under different workloads during the experiments, indicating that the VTT for the same video segment might vary on different processors. Therefore, the scheduler needs to search for the fastest processors to transcode videos that are being watched.

Table 5.6: Measured values for the evaluation metrics on varying L_{va} in Cloud1 with Dataset4.

Client ID	L_{va}	L_s	N_{QS}	N_{RE}	T_{mr}	MOS
1	5	4.26	1	1	3.54	2.08
	10	8.51	0	0	0	3.66
	15	13.11	0	0	0	3.35
2	5	4.78	1	8	3.14	-3.10
	10	9.05	1	3	2.05	0.44
	15	16.08	1	2	3.41	0.56
3	5	6.96	1	3	1.24	0.66
	10	11.44	1	1	2.6	1.70
	15	14.99	1	1	2.17	1.51
4	5	7.15	1	1	3.16	1.93
	10	10.69	1	0	0	2.77
	15	14.32	0	0	0	3.27
5	5	6.82	0	0	0	3.77
	10	8.47	0	0	0	3.66
	15	13.15	0	0	0	3.35

We first tested the extreme condition that videos are requested to be watched as soon as they are available. In this case, we used Stream 2 to upload videos and generated five clients to send video watching requests when the system reached its maximum capacity. The requests are sent to the server as soon as the initial L_{va} seconds of video are transcoded, and we keep Bit_r as the medium bitrate. The value of Bit_r changes to the low bitrate only when the required medium rate is not

available and it will never change back again. Table 5.6 summarizes the values of the evaluation metrics of each client when L_{va} changes. Figure 5.8 shows the VTT normalized by segment duration for all the clients and the average among processors running jobs in NQueue at the time interval since Stream 2 starts to upload videos. When comparing Clients 1 to 5, a small L_{va} always causes quality switches during the playback for the subsequent segments. The other outlier is for Client 5 whose requests are received last. The reason is the scheduling algorithm carried out by the server. Since the scheduler always searches for the fastest CPU and reserves enough processors to run jobs in PQueue, the processing speed of processors that transcode for Clients 1 to 4 are faster than or at least equal to that for Client 5. Therefore, transcoding segments for Client 5 on any processors causes no quality switch. On the other hand, when segments requested by other clients are transcoded by the processors joined due to Client 5 sending a video watching request, the playback might be interrupted due to the slower speed of the these processors. For example, there exist jitters with the normalized VTT for Clients 2 to 4, which indicates that the video transcoding speed varies significantly during the playback. This situation hence causes quality switches and rebuffering events on Clients 2 to 4. When comparing the values of MOS, there are no best parameters. Small values of L_{va} enable end-users to access the video shortly after it is uploaded, while a large L_{va} supports smooth playback better. Consequently, we need to consider the trade-off among these parameters and adjust them based on the target application. In the following experiments, we set L_{va} to 10 seconds as it neither delays the playback too long nor causes too many interrupts.

We next show that the proposed scheduler distributes urgent jobs to the fastest processors. Figure 5.8 compares the normalized VTT among processors transcoding segments for clients and those for normally uploaded segments. For most of the time, the normalized VTT is smaller for Clients 1 to 5 than that for NQueue, indicating that the fastest processors are used to transcode video segments for the watching

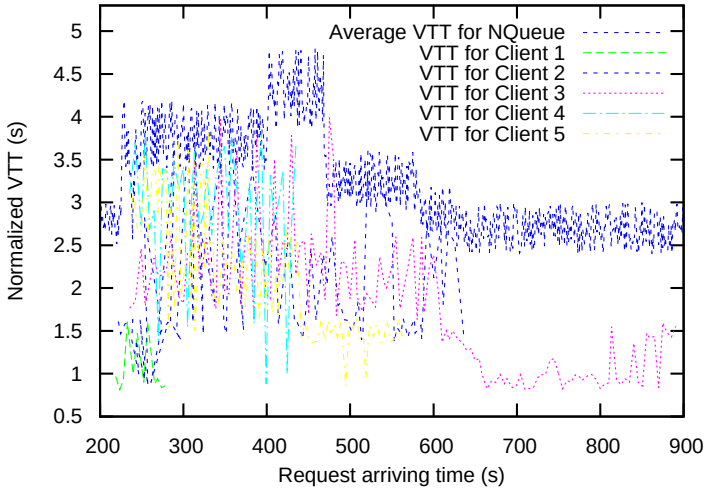


Figure 5.8: Comparison of the normalized video transcoding time VTT (L_{va} = 10 seconds) in Cloud1 with Dataset4.

clients. On the other hand, the average VTT for NQueue increases when the server receives video watching requests. The reason is that only the slowest processors are assigned to run jobs in NQueue.

Finally, we illustrate the benefits of the automatic VTM switching functionality. Figure 5.9 shows the comparison between the deadline and completion time of transcoding. At the beginning, after the server receives the video watching request from Client 2, the time difference between the two curves increases since faster processors transcoding for Client 1 can serve Client 2. After Clients 3 to 5 join, some slower processors also serve Client 2. The sharp VTT changes on Client 2 and the increasing VTT result in a quality switch when Segment #38 is finished transcoding. After the quality switch, Client 2 changes its request from medium to low bitrate and the scheduler only changes VTM from 2 to 1. This strategy shortens the completion time for the required bitrate for a single transcoding job but does not help subsequentially in transcoding since the overall VTT on the processor remains the same. The next three rebuffering events also happen for the same reason. Conversely, when the scheduler switches VTM after encountering the first rebuffering event, the transcoding completion time is shortened significantly. This

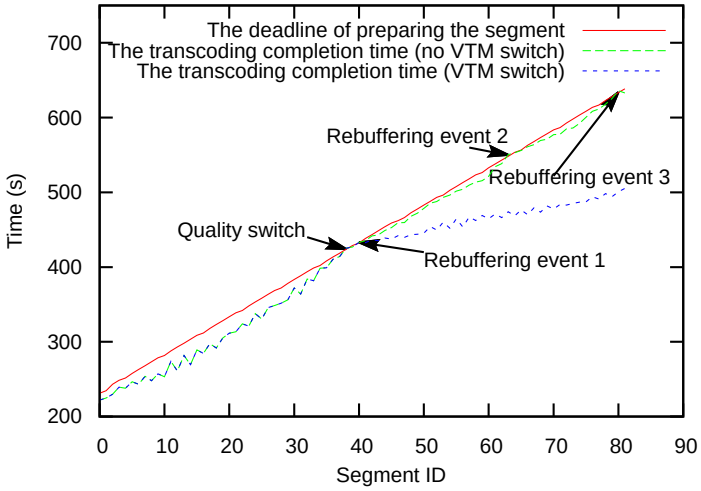


Figure 5.9: Comparison between the deadline and completion time for each video segment watched by Client 2 (L_{va} = 10 seconds) in Cloud1 with Dataset4.

can improve the performance of the scheduler and the system throughput, as well as improve end-users' experiences. However, the drawback of automatic VTM is that the system might suffer from frequent changing of the required bitrates, which seldom happens.

5.4.2 Experiment with Dataset5

We compare the proposed scheduling algorithm with the FIFO policy in Cloud2 with Dataset5 where all the computing resources can be fully utilized by the scheduler.

5.4.2.1 Performance Evaluation

We study the efficiency of the proposed algorithm compared with standard FIFO policy with a real-world video uploading stream set. In this experiment, there exist only video uploading streams but no video viewing requests. Therefore, the video segments are transcoded to both low bitrate and medium bitrate simultaneously (VTM = 0). Figure 5.10(a) shows the number of segments received at the server side per second, which during some periods of time exceeds the maximum capacity of the system. Figures 5.10(b) and 5.10(c) present the comparison of the cumulative

segment queueing time (CSQT for short) and completion time (CSCT) between the proposed algorithm and the FIFO algorithm, respectively. Overall, the differences between CSQT and CSCT for both of the algorithms are not significant. The proposed algorithm presents a slight improvement on both the CSQT and CSCT. The reason is that we only have a small uploading dataset with a light workload and most of the video segments are of the same length. The transcoding sequences for both algorithms are almost the same. When the workload exceeds the system capacity, the scheduler starts to distribute the predicted faster jobs to other processors and hence the CSQT and CSCT decrease. Moreover, since a processing node can finish most VTJs with around 1.5 to 2 seconds, the difference on the CSQT is always small.

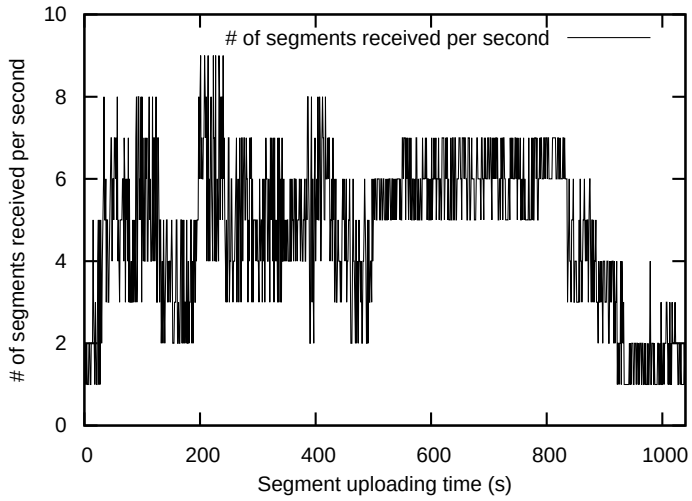
At the beginning of the uploading procedure (*i.e.,* for the first 200 segments), the workload never reaches the maximum capacity of the system. The cumulative segment queueing time is almost zero and there exists no difference between the two algorithms. When the workload starts to exceed the system capacity (*e.g.,* at the 240^{th} segment) and the queue grows, the values of CSQT and CSCT increase and a small gap begins to exist between the two algorithms. While the system works under overload (*i.e.,* uploading segments with IDs around 2,500 to 4,000), the CSQT has a larger increasing rate and both the gaps in CSQT and CSCT increase. From the above analysis, it can be inferred that the proposed scheduler can improve the video transcoding performance, especially when the system is overloaded with a heavy workload.

5.5 Summary

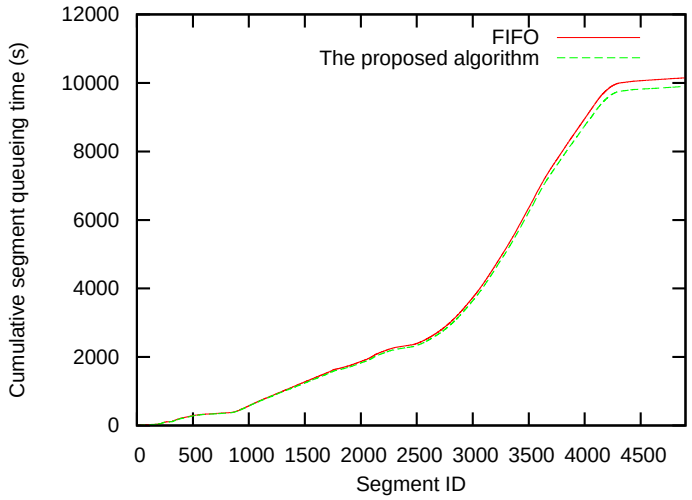
In this work we proposed a dynamic scheduling algorithm for video transcoding jobs designed to support Dynamic Adaptive Streaming over HTTP in a cloud environment. We first modeled the video transcoding time VTT estimation based on statistics from video segment durations and the targeted bitrate. The sched-

uler then dynamically distributes video transcoding jobs VTJ according to VTT estimation and the system workload. Overall, the proposed scheduler can support near-live streaming while balancing the workload and provide smooth and seamless playback. Most importantly it can dynamically serve requests and adjust the video transcoding mode VTM accordingly. Experimental results show that the scheduler can distribute urgent jobs to the fastest processors and shorten the transcoding completion time by using VTM switching. Multiple configurations of the scheduler allow it to be adjusted according to the needs of different applications.

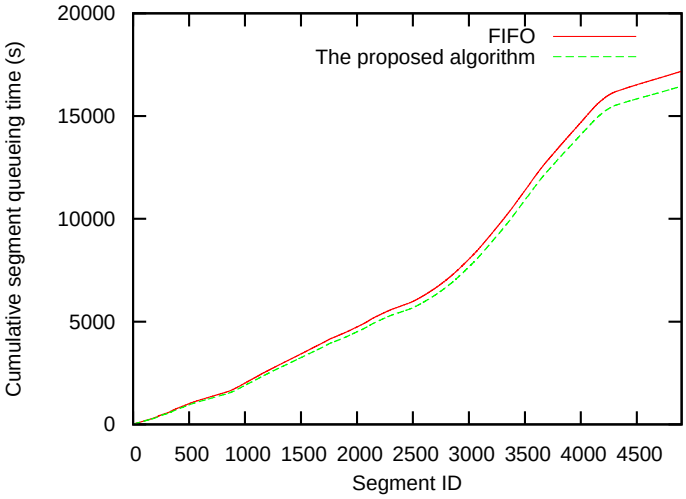
The limitation of this work lies in that the VTT estimation methodology relies on the experimental environment, which might change all the time. In this case, we plan to design a self-tuning algorithm to adjust the VTT estimation algorithm and parameters while the workload in the cloud environment changes. Besides, more alternative bitrates and comparison algorithms will be included in future work. Furthermore, we plan to investigate a cost model for the scheduler, which would be helpful for administrators to select appropriate cloud services while considering multiple parameters based on the user's preference.



(a) Uploading stream.



(b) Cumulative segment queueing time.



(c) Cumulative segment completion time.

Figure 5.10: Comparison on the segment queueing time and completion between the proposed algorithm and FIFO algorithm in Cloud2 with Dataset5.

Chapter 6

Conclusions and Future Work

In conclusion, my research aims at improving the performance of geo-tagged video search framework by considering the storage, queries and streaming methodologies. We have studied the important role that sensor meta-data associated with visual content play in geo-tagged video management.

In the first work, we have built a multi-level grid-based index structure for effectively and efficiently searching for geo-tagged videos for both typical spatial queries and queries with radius and direction restriction. The proposed index structure manages video segments, neither the entire video clip nor a single frame, as the results. Experiments on the real-world dataset show the importance of the queries with bounded radius and viewing direction restriction. The experimental results with large-scale synthetic dataset show that the proposed structure outperforms well for both types of queries, and speed up the query procedure compared with typical R-tree index structure. In the future, we will keep collecting data so that the dataset can be widely used. In addition, the parameters used in the FOV model (*e.g.,* camera zoom information) could be collected and used in the future research so that the query results can be improved. Furthermore, since the queries with bounded radius and direction restriction can provide specific geo-properties of a place or building, the 3D building construction from video is considered as one of the potential research topics. As the growth of the dataset and increasing concurrent video streaming requests, a grid-based index structure in the Cloud for geo-tagged videos can be developed for parallel processing queries efficiently.

In order to deal with the uncertainty of the sensor meta-data collected from

mobile devices, we have also designed the uncertain data model for individual frames and video segments. The proposed data model overcomes the noisy nature of the sensor data, as well as obstacles in front of a camera. The video segmentation method based on the geo-properties of videos is also provided. We constructed a light-weight approximate model for video segments based on the sensor meta-data. With this architecture, probabilistic queries can be executed and upstream GIS takes prioritized based on the most promising results. The work related to storage and index are mainly dealt with meta-data. Regarding to the current work, the point-of-interest (POI) detection can also be processed with the current techniques. In the future research, we plan to set up the ground-truth dataset of digital compass error distribution, by comparing the embedded compass readings with that obtained from more accurate sensors and other information. Moreover, one of the possible directions is to identify the most aesthetical representative frame or image of a query (*e.g.,* a building or landmark), considering both the meta-data of the videos and the information of the target. Based on this, a new video ranking algorithm can be designed to satisfy users' preference and this algorithm could be used to provide the best video summarization. In addition, detecting obstacles utilizing content-based method could be a complementary method.

In the third work, we investigated the streaming methodology and proposed the scheduling strategy on video transcoding for DASH in the cloud environment. The statistical VTT estimation method is designed with respect to the video duration. The scheduler dynamically distributed high-priority $VTJs$ to fastest processors and normally uploaded $VTJs$ to free processors to shorten the transcoding completion time by using VTM switching, as well as balancing the workloads among different processors. Multiple configurations of the scheduler allow it to be adjusted according to the needs of different applications. The short latency between the video uploaded time and transcoding completion time indicates that the scheduler can support near-live DASH streaming. As a follow-up step, we plan to provide

updated scheduling algorithm based on the current version, which would support transcoding to more alternative bitrates of videos in a smart way, where the situation during video transcoding is more complex. Moreover, a scheduling algorithm in elastic cloud would also be interesting. In addition, the VTT estimation algorithm can be improved in a self-tuning way, so that it can adjust the parameters when the cloud environment periodically changes. Last but not least, a cost model can be added to the scheduler, the scheduler could then carry out different strategies according to user's preference.

Bibliography

[Adobe System Inc] Adobe System Inc. *HTTP Dynamic Streaming*. URL:http://www.adobe.com/products/hds-dynamic-streaming.html. (Cited on page 22.)

[Agarwal 2003] Pankaj K Agarwal, Lars Arge and Jeff Erickson. *Indexing Moving Points*. Journal of Computer and System Sciences, vol. 66, no. 1, pages 207–243, 2003. (Cited on page 15.)

[Akrami 2013] Farahnaz Akrami and Farzad Zargari. *An Efficient Compressed Domain Video Indexing Method*. Multimedia Tools and Applications, pages 1–17, 2013. (Cited on page 11.)

[Amazon 2013] Amazon. *Amazon Elastic Transcoder*, 2013. URL: http://aws.amazon.com/elastictranscoder/. (Cited on page 18.)

[Angiulli 2012] Fabrizio Angiulli and Fabio Fassetti. *Indexing Uncertain Data in General Metric Spaces*. TKDE, IEEE Transactions on Knowledge and Data Engineering, vol. 24, no. 9, pages 1640–1657, 2012. (Cited on page 17.)

[Arslan Ay 2008] Sakire Arslan Ay, Roger Zimmermann and Seon Ho Kim. *Viewable Scene Modeling for Geospatial Video Search*. In SIGMM, ACM International Conference on Multimedia, pages 309–318, 2008. (Cited on pages 2, 12, 21, 27, 78 and 89.)

[Arslan Ay 2010] Sakire Arslan Ay, Roger Zimmermann and Seon Ho Kim. *Generating Synthetic Meta-data for Georeferenced Video Management*. In ACM SIGSPATIAL GIS International Conference on Advances in Geographic Information Systems, pages 280–289, 2010. (Cited on page 27.)

[Beckmann 1990] N. Beckmann, H.P. Kriegel, R. Schneider and B. Seeger. *The R*-tree: An Efficient and Robust Access Method for Points and Rectangles*. In SIGMOD, ACM International Conference on Management of Data, pages 322–331, 1990. (Cited on pages 13 and 35.)

[Beckmann 2009] Norbert Beckmann and Bernhard Seeger. *A Revised R*-tree in Comparison with Related Index Structures*. In sigmod, pages 799–812. ACM, 2009. (Cited on page 13.)

[Bentley 1975] J.L. Bentley. *Multidimensional Binary Search Trees Used for Associative Searching*. Communications of the ACM, vol. 18, no. 9, pages 509–517, 1975. (Cited on page 14.)

[Berchtold 2001] Stefan Berchtold, Daniel A Keim and Hans-Peter Kriegel. *The X-tree: An Index Structure for High-dimensional Data*. Readings in multimedia computing and networking, page 451, 2001. (Cited on page 14.)

[Biveinis 2007] Laurynas Biveinis, Simonas Šaltenis and Christian S Jensen. *Main-memory Operation Buffering for Efficient R-tree Update.* In the 33rd International Conference on Very Large Data Bases, pages 591–602. VLDB Endowment, 2007. (Cited on page 16.)

[Böhm 2006] Christian Böhm, Alexey Pryakhin and Matthias Schubert. *The Gauss-Tree: Efficient Object Identification in Databases of Probabilistic Feature Vectors.* In ICDE, page 9, 2006. (Cited on page 17.)

[Boreczky 1998] J.S. Boreczky and L.D. Wilcox. *A Hidden Markov Model Framework for Video Segmentation Using Audio and Image Features.* In Acoustics, IEEE International Conference on Speech and Signal Processing, volume 6, pages 3741–3744, 1998. (Cited on page 10.)

[Budvytis 2011] Ignas Budvytis, Vijay Badrinarayanan and Roberto Cipolla. *Semi-Supervised Video Segmentation using Tree Structured Graphical Models.* In IEEE Conference on Computer Vision and Pattern Recognition (CVPR), pages 2257–2264. IEEE, 2011. (Cited on page 10.)

[Cernekova 2006] Z. Cernekova, N. Nikolaidis and I. Pitas. *Temporal Video Segmentation by Graph Partitioning.* In Acoustics, IEEE International Conference on Speech and Signal Processing, volume 2, pages 209–212, 2006. (Cited on page 10.)

[Chen 2007] Jinchuan Chen and Reynold Cheng. *Efficient Evaluation of Imprecise Location-Dependent Queries.* In ICDE, IEEE International Conference on Data Engineering, pages 586–595, 2007. (Cited on page 17.)

[Chen 2008] S. Chen, B.C. Ooi, K.L. Tan and M.A. Nascimento. *ST^2B-tree: A Self-Tunable Spatio-Temporal B^+-Tree Index for Moving Objects.* In Proceedings of the 2008 ACM SIGMOD international conference on Management of data, pages 29–42. ACM, 2008. (Cited on page 16.)

[Cheng 2004a] Reynold Cheng, Sunil Prabhakar and Dmitri V. Kalashnikov. *Querying Imprecise Data in Moving Object Environments.* In TKDE, IEEE Transactions on Knowledge and Data Engineering, volume 16, pages 1112–1127, 2004. (Cited on page 17.)

[Cheng 2004b] Reynold Cheng, Yuni Xia, Sunil Prabhakar, Rahul Shah and Jeffrey Scott Vitter. *Efficient Indexing Methods for Probabilistic Threshold Queries over Uncertain Data.* In VLDB, International Conference on Very Large Data Bases, pages 876–887, 2004. (Cited on page 17.)

[Chon 2003] H.D. Chon, D. Agrawal and A.E. Abbadi. *Range and KNN Query Processing for Moving Objects in Grid Model.* Mobile Networks and Applications, vol. 8, no. 4, pages 401–412, 2003. (Cited on page 15.)

[Ciaccia 1997] Paolo Ciaccia, Marco Patella and Pavel Zezula. *M-tree: An Efficient Access Method for Similarity Search in Metric Spaces.* In 23rd International Conference on Very Large Data Bases, pages 426–435, 1997. (Cited on page 14.)

[Cisco 2014] Cisco. *Cisco Visual Networking Index: Global Mobile Data Traffic Forecast Update, 2013-2018*, 2014. URL: http://www.cisco.com/en/US/solutions/collateral/ns341/ns525/ns537/ns705/ns827/white_paper_c11-520862.pdf. (Cited on pages 1 and 2.)

[Crandall 2009] David J. Crandall, Lars Backstrom, Daniel Huttenlocher and Jon Kleinberg. *Mapping the World's Photos.* In Proceedings of the 18th international conference on World wide web, WWW '09, pages 761–770, New York, NY, USA, 2009. ACM. (Cited on page 11.)

[DAS 2012] *ISO/IEC 23009-1: 2012 Information technology – Dynamic adaptive streaming over HTTP (DASH) – Part 1: Media presentation description and segment formats.*, 2012. URL:http://www.iso.org/iso/iso_catalogue/catalogue_tc/catalogue_detail.htm?csnumber=57623. (Cited on page 23.)

[Emrich 2012] Tobias Emrich, Hans-Peter Kriegel, Nikos Mamoulis, Matthias Renz and Andreas Züfle. *Indexing Uncertain Spatio-Temporal Data.* In the 21st ACM International Conference on Information and Knowledge Management, pages 395–404. ACM, 2012. (Cited on page 18.)

[EncoderCloud] EncoderCloud. URL: http://www.encodercloud.com/. (Cited on page 19.)

[Eppstein 2005] D. Eppstein, M.T. Goodrich and J.Z. Sun. *The Skip Quadtree: A Simple Dynamic Data Structure for Multidimensional Data.* In Annual Symposium on Computational Geometry, 2005. (Cited on page 15.)

[Epshtein 2007] Boris Epshtein, Eyal Ofek, Yonatan Wexler and Pusheng Zhang. *Hierarchical Photo Organization Using Geo-Relevance.* In 15^{th} ACM Intl. Symposium on Advances in Geographic Information Systems (GIS), pages 1–7, 2007. (Cited on page 12.)

[Erifiu 2012] Alexander Erifiu and Gerald Ostermayer. *Hardware sensor aspects in mobile augmented reality.* In Computer Aided Systems Theory–EUROCAST, pages 536–543. Springer, 2012. (Cited on page 74.)

[Evangelidis 1997] Georgios Evangelidis, David Lomet and Betty Salzberg. *The hB^{Π}-tree: a Multi-attribute Index Supporting Concurrency, Recovery and Node Consolidation.* The VLDB Journal, vol. 6, no. 1, pages 1–25, 1997. (Cited on page 15.)

[Fang 2012] Ying Fang, Jiaheng Cao, Junzhou Wang, Yuwei Peng and Wei Song. *HTPR*-Tree: An Efficient Index for Moving Objects to Support Predictive Query and Partial History Query.* In Web-Age Information Management, pages 26–39. Springer, 2012. (Cited on page 16.)

[Faria 2002] G. Faria, C.B. Medeiros and M.A. Nascimento. *An Extensible Framework for Spatio-Temporal Database Applications.* In International Conference

on Scientific and Statistical Database Management, pages 202–205. IEEE, 2002. (Cited on page 15.)

[ffm] *FFmpeg*. URL:http://www.ffmpeg.org/. (Cited on page 104.)

[Finkel 1974] R.A. Finkel and J.L. Bentley. *Quad Trees: A Data Structure for Retrieval on Composite Keys*. Acta informatica, vol. 4, no. 1, pages 1–9, 1974. (Cited on page 14.)

[Gao 2005] Xinbo Gao, Bing Han and Hongbing Ji. *A Shot Boundary Detection Method for News Video Based on Rough Sets and Fuzzy Clustering*. In Image Analysis and Recognition, volume 3656, pages 231–238. Springer, 2005. (Cited on page 10.)

[Gieseke 2014] Fabian Gieseke, Justin Heinermann, Cosmin Oancea and Christian Igel. *Buffer kd Trees: Processing Massive Nearest Neighbor Queries on GPUs*. In Proceedings of The 31st International Conference on Machine Learning, pages 172–180, 2014. (Cited on page 15.)

[Graham 1965] Clarence H. Graham, Neil R. Bartlett, John Lott Brown, Yun Hsia, Conrad C. Mueller and Lorrin A. Riggs. *Vision and Visual Perception*. 1965. (Cited on pages 22 and 89.)

[Green 2010] Melinda Green. *R-Tree, Templated C++ Implementation*, 2010. URL: http://superliminal.com/sources/RTreeTemplate.zip. (Cited on pages 60 and 88.)

[Grundmann 2010] Matthias Grundmann, Vivek Kwatra, Mei Han and Irfan Essa. *Efficient Hierarchical Graph-based Video Segmentation*. In IEEE Conference on Computer Vision and Pattern Recognition (CVPR), pages 2141–2148. IEEE, 2010. (Cited on page 10.)

[Guttman 1984] Antonin Guttman. *R-Trees: A Dynamic Index Structure for Spatial Searching*. In SIGMOD, ACM International Conference on Management of Data, pages 47–57, 1984. (Cited on pages 13 and 35.)

[Hadjieleftheriou 2002] M. Hadjieleftheriou, G. Kollios, V. Tsotras and D. Gunopulos. *Efficient Indexing of Spatiotemporal Objects*. Advances in Database Technology-EDBT 2002, pages 251–268, 2002. (Cited on page 15.)

[Hadjieleftheriou 2006] Marios Hadjieleftheriou, George Kollios, Vassilis J Tsotras and Dimitrios Gunopulos. *Indexing Spatiotemporal Archives*. The VLDB Journal, vol. 15, no. 2, pages 143–164, 2006. (Cited on page 15.)

[Hao 2011] Jia Hao, Seon Ho Kim, Sakire Arslan Ay and Roger Zimmermann. *Energy-Efficient Mobile Video Management using Smartphones*. In the 2nd ACM Multimedia Systems Conference, pages 11–22, 2011. (Cited on page 6.)

[He 2005] Yong He, Haihong Yu and Hui Fang. *Study on improving GPS measurement accuracy.* In I2MTC, IEEE International Instrumentation and Measurement Technology Conference, volume 2, pages 1476–1479. IEEE, 2005. (Cited on page 78.)

[Henrich 1989] Andreas Henrich, Hans-Werner Six, FemUniversit& Hagen and Peter Widmayer. *The LSD Tree: Spatial Access to Multidimensional Point and Non-saint Objects.* pages 45–53, 1989. (Cited on page 15.)

[Hu 2011] Weiming Hu, Nianhua Xie, Li Li, Xianglin Zeng and Stephen Maybank. *A Survey on Visual Content-based Video Indexing and Retrieval.* Systems, Man, and Cybernetics, Part C: Applications and Reviews, IEEE Transactions on, vol. 41, no. 6, pages 797–819, 2011. (Cited on page 11.)

[Hwang 2003] Tae Hyun Hwang, Kyoung Ho Choi, In Hak Joo and Jong Hun Lee. *MPEG-7 Metadata for Video-based GIS Applications.* In IEEE International Geoscience and Remote Sensing Symposium, volume 6, pages 3641–3643, 2003. (Cited on page 11.)

[Ilarri 2010] Sergio Ilarri, Eduardo Mena and Arantza Illarramendi. *Location-Dependent Query Processing: Where We Are and Where We Are Heading.* ACM Computing Surveys (CSUR), vol. 42, no. 3, page 12, 2010. (Cited on page 15.)

[Jaffe 2006] Alexander Jaffe, Mor Naaman, Tamir Tassa and Marc Davis. *Generating Summaries for Large Collections of Geo-referenced Photographs.* In 15^{th} International Conference on the World Wide Web, pages 853–854, 2006. (Cited on page 12.)

[Jensen 2004] C.S. Jensen, D. Lin and B.C. Ooi. *Query and Update Efficient B^+-Tree Based Indexing of Moving Objects.* In Proceedings of the Thirtieth international conference on Very large data bases-Volume 30, pages 768–779. VLDB Endowment, 2004. (Cited on page 16.)

[Kamel 1994] Ibrahim Kamel and Christos Faloutsos. *Hilbert R-tree: An Improved R-tree using Fractals.* pages 500–509, 1994. (Cited on page 14.)

[Kim 2003] Kyong Ho Kim, Sung Soo Kim, Sung Ho Lee, Jong Hyun Park and Jong Hyun Lee. *The Interactive Geographic Video.* In IGARSS, IEEE International Geoscience and Remote Sensing Symposium, volume 1, pages 59–61, 2003. (Cited on page 11.)

[Kim 2007] Dong Oh Kim, Dong Suk Hong, Hong Koo Kang and Ki Joon Han. *UR-Tree: An Efficient Index for Uncertain Data in Ubiquitous Sensor Networks.* In GPC, International Conference on Advances in Grid and Pervasive Computing, pages 603–613, 2007. (Cited on page 17.)

[Kim 2010] S.H. Kim, S. Arslan Ay, B. Yu and R. Zimmermann. *Vector Model in Support of Versatile Georeferenced Video Search.* In ACM SIGMM Conference on Multimedia Systems, 2010. (Cited on page 12.)

[Kim 2014] Youngwoo Kim, Jinha Kim and Hwanjo Yu. *GeoTree: Using Spatial Information for Georeferenced Video Search.* Knowledge-Based Systems, vol. 61, pages 1–12, 2014. (Cited on page 14.)

[Kllapi 2011] Herald Kllapi, Eva Sitaridi, Manolis M Tsangaris and Yannis Ioannidis. *Schedule Optimization for Data Processing Flows on the Cloud.* In 2011 sigmod, pages 289–300. ACM, 2011. (Cited on page 20.)

[Larson 2011] Martha Larson, Mohammad Soleymani and Pavel Serdyukov. *Automatic tagging and geotagging in video collections and communities.* In the 1st ACM International Conference on Multimedia Retrieval, ICMR '11, pages 51:1–51:8, New York, NY, USA, 2011. ACM. (Cited on page 11.)

[Lezama 2011] José Lezama, Karteek Alahari, Josef Sivic and Ivan Laptev. *Track to the Future: Spatio-temporal Video Segmentation with Long-range Motion Cues.* In IEEE Conference on Computer Vision and Pattern Recognition (CVPR), pages 3369–3376. IEEE, 2011. (Cited on page 10.)

[Li 2005] Ping Li, Bharadwaj Veeravalli and Ashraf A Kassim. *Design and Implementation of Parallel Video Encoding Strategies Using Divisible Load Analysis.* Circuits and Systems for Video Technology, IEEE Transactions on, vol. 15, no. 9, pages 1098–1112, 2005. (Cited on page 20.)

[Li 2007] Rui Li, Bir Bhanu, Chinya V. Ravishankar, Michael Kurth and Jinfeng Ni. *Uncertain Spatial Data Handling: Modeling, Indexing and Query.* In Computers & Geosciences, volume 33, pages 42–61, 2007. (Cited on page 17.)

[Li 2012] Xiaowei Li, Yi Cui and Yuan Xue. *Towards an Automatic Parameter-Tuning Framework for Cost Optimization on Video Encoding Cloud.* International Journal of Digital Multimedia Broadcasting, vol. 2012, 2012. (Cited on page 20.)

[Lin 2005] Dan Lin, Christian S Jensen, Beng Chin Ooi and Simonas Šaltenis. *Efficient Indexing of the Historical, Present, and Future Positions of Moving Objects.* In MDM, the 6th International Conference on Mobile Data Management, pages 59–66. ACM, 2005. (Cited on page 16.)

[Lin 2006] Dan Lin, Rui Zhang and Aoying Zhou. *Indexing Fast Moving Objects for kNN Queries Based on Nearest Landmarks.* GeoInformatica, vol. 10, no. 4, pages 423–445, 2006. (Cited on page 16.)

[Liu 2005] X. Liu, M. Corner and P. Shenoy. *SEVA: Sensor-Enhanced Video Annotation.* In SIGMM, ACM International Conference on Multimedia, pages 618–627, 2005. (Cited on pages 2 and 12.)

[Liu 2012] Xiao-Dan Liu, Jia-Ze Wu and Chang-Wen Zheng. *KD-tree based Parallel Adaptive Rendering.* The Visual Computer, vol. 28, no. 6-8, pages 613–623, 2012. (Cited on page 15.)

[Lomet 1990] David B Lomet and Betty Salzberg. *The hB-tree: A Multiattribute Indexing Method with Good Guaranteed Performance*. ACM Transactions on Database Systems (TODS), vol. 15, no. 4, pages 625–658, 1990. (Cited on page 15.)

[Luo 2010] Zhiping Luo, Haojie Li, Jinhui Tang, Richang Hong and Tat-Seng Chua. *Estimating Poses of World's Photos with Geographic Metadata*. In Advances in Multimedia Modeling, pages 695–700. Springer, 2010. (Cited on page 12.)

[Ma 2011] Kevin J Ma, Radim Bartos, Swapnil Bhatia and Raj Nair. *Mobile video delivery with HTTP*. Communications Magazine, IEEE, vol. 49, no. 4, pages 166–175, 2011. (Cited on page 5.)

[Ma 2012a] He Ma, Sakire Arslan Ay, Roger Zimmermann and Seon Ho Kim. *A Grid-Based Index and Queries for Large-Scale Geo-tagged Video Collections*. In SIM^3, 17th International Conference, DASFAA workshops, pages 216–228, 2012. (Cited on page 6.)

[Ma 2012b] He Ma, Roger Zimmermann and Seon Ho Kim. *HUGVid: Handling, Indexing and Querying of Uncertain Geo-Tagged Videos*. In ACM SIGSPATIAL GIS International Conference on Advances in Geographic Information Systems, pages 319–328, 2012. (Cited on page 6.)

[Ma 2013] He Ma, Sakire Arslan Ay, Roger Zimmermann and Seon Ho Kim. *Large-scale Geo-tagged Video Indexing and Queries*. GeoInformatica, 2013. (Cited on page 6.)

[Ma 2014] He Ma, Beomjoo Seo and Roger Zimmermann. *Dynamic Scheduling on Video Transcoding for MPEG DASH in the Cloud Environment*. In MMSys, ACM Multimedia Systems Conference, 2014. (Cited on page 7.)

[Martin 2006] Jason Daniel Martin, Jens Krosche and Susanne Boll. *Dynamic GPS-position Correction for Mobile Pedestrian Navigation and Orientation*. In WPNC, Workshop on Positioning, Navigation and Communication, pages 199–208, 2006. (Cited on page 4.)

[Meagher 1982] Donald Meagher. *Geometric modeling using octree encoding*. Computer graphics and image processing, vol. 19, no. 2, pages 129–147, 1982. (Cited on page 14.)

[Metropolis 1949] N. Metropolis and S. Ulam. *The Monte Carlo Method*. In Journal of the American Statistical Association, volume 44, pages 335–341, 1949. (Cited on page 82.)

[Microsoft Corporation 2012] Microsoft Corporation. *Smooth Streaming Protocol Specification*, 2012. URL:`http://download.microsoft.com/download/9/5/E/95EF66AF-9026-4BB0-A41D-A4F81802D92C/[MS-SSTR].pdf`. (Cited on page 22.)

[Mok 2011] Ricky K. P. Mok, Edmond W. W. Chan and Rocky K. C. Chang. *Measuring the Quality of Experience of HTTP Video Streaming.* In 12th IFIP/IEEE International Symposium on Integrated Network Management, pages 485–492, 2011. (Cited on page 111.)

[Nanopoulos 2006] Alexandros Nanopoulos, Apostolos N Papadopoulos and Yannis Theodoridis. R-trees: Theory and Applications. Springer, 2006. (Cited on page 14.)

[Navarrete 2002] T. Navarrete and J. Blat. *VideoGIS: Segmenting and Indexing Video Based on Geographic Information.* In AGILE, Conference on Geographic Information Science, pages 1–9, 2002. (Cited on page 12.)

[Ngo 2003] C.W. Ngo, T.C. Pong and H.J. Zhang. *Motion Analysis and Segmentation through Spatio-Temporal Slices Processing.* Image Processing, IEEE Transactions on, vol. 12, no. 3, pages 341–355, 2003. (Cited on page 11.)

[Nievergelt 1984] J. Nievergelt, H. Hinterberger and K.C. Sevcik. *The Grid File: An Adaptable, Symmetric Multikey File Structure.* ACM Transactions on Database Systems (TODS), vol. 9, no. 1, pages 38–71, 1984. (Cited on page 14.)

[Nutanong 2008] S. Nutanong, R. Zhang, E. Tanin and L. Kulik. *The V*-Diagram: A Query-dependent Approach to Moving KNN Queries.* Proceedings of the VLDB Endowment, vol. 1, no. 1, pages 1095–1106, 2008. (Cited on page 15.)

[O'Connor 2008] N.E. O'Connor, T. Duffy, C. Gurrin, H. Lee, D.A. Sadlier, A.F. Smeaton and K. Zhang. *A Content-Based Retrieval System for UAV-like Video and Associated Metadata.* In Airborne Intelligence, Surveillance, Reconnaissance Systems and Applications, 2008. (Cited on page 11.)

[Okabe 2000] A. Okabe. Spatial Tessellations: Concepts and Applications of Voronoi Diagrams. John Wiley & Sons Inc, 2000. (Cited on page 14.)

[Okuda 1993] Kuniharu Okuda and Akio Yasuda. *Distribution of positioning error in a global positioning system.* Electronics and Communications in Japan (Part I: Communications), vol. 76, no. 1, pages 95–103, 1993. (Cited on page 78.)

[Pantos 2012] R Pantos and Ed. W. May. *HTTP Live Streaming*, 2012. URL:`http://tools.ietf.org/pdf/draft-pantos-http-live-streaming-10.pdf`. (Cited on page 22.)

[Pelanis 2006] Mindaugas Pelanis, Simonas Šaltenis and Christian S Jensen. *Indexing the Past, Present, and Anticipated Future Positions of Moving Objects.* ACM Transactions on Database Systems (TODS), vol. 31, no. 1, pages 255–298, 2006. (Cited on page 16.)

[Peng 2012] Shangfu Peng, Yin Yang, Zhenjie Zhang, Marianne Winslett and Yong Yu. *DP-tree: Indexing Multi-dimensional Data under Differential Privacy.*

In Proceedings of the 2012 ACM SIGMOD International Conference on Management of Data, pages 864–864. ACM, 2012. (Cited on page 15.)

[peter Kriegel 2006] Hans peter Kriegel, Peter Kunath, Martin Pfeifle and Matthias Renz. *Probabilistic Similarity Join on Uncertain Data*. In DASFAA, International Conference on Database Systems for Advanced Applications, pages 295–309, 2006. (Cited on page 17.)

[Pettersen 2014] Svein Arne Pettersen, Dag Johansen, Håvard Johansen, Vegard Berg-Johansen, Vamsidhar Reddy Gaddam, Asgeir Mortensen, Ragnar Langseth, Carsten Griwodz, Håkon Kvale Stensland and Pål Halvorsen. *Soccer Video and Player Pposition Dataset*. In Proceedings of the 5th ACM Multimedia Systems Conference, pages 18–23. ACM, 2014. (Cited on page 12.)

[Pfoser 2000] D. Pfoser, C.S. Jensen and Y. Theodoridis. *Novel Approaches in Query Processing for Moving Object Trajectories*. In Proceedings of the 26th International Conference on Very Large Data Bases, pages 395–406. Morgan Kaufmann Publishers Inc., 2000. (Cited on page 15.)

[Priyantha 2000] Nissanka B. Priyantha, Anit Chakraborty and Hari Balakrishnan. *The Cricket Location-Support System*. In MobiCom, ACM International Conference on Mobile Computing and Networking, pages 32–43, 2000. (Cited on page 21.)

[Rackspace] Rackspace. *The Rackspace Cloud*. URL:`http://www.rackspace.com/cloud/`. (Cited on page 19.)

[Rife 2012] Jason Rife and Boris Pervan. *Overbounding Revisited: Discrete Error-Distribution Modeling for Safety-Critical GPS Navigation*. IEEE Transactions on Aerospace and Electronic Systems, vol. 48, no. 2, pages 1537–1551, 2012. (Cited on page 78.)

[Rigaux 2001] Philippe Rigaux, Michel Scholl and Agnes Voisard. Spatial Databases with Application to GIS. Morgan Kaufmann, 2001. (Cited on page 13.)

[Robinson 1981] John T Robinson. *The K-D-B-tree: a Search Structure for Large Multidimensional Dynamic Indexes*. In sigmod, pages 10–18. ACM, 1981. (Cited on page 15.)

[Roussopoulos 1987] N. Roussopoulos, C. Faloutsos and S. Timos. *The R^+-tree: A Dynamic Index for Multi-dimensional Objects*. In VLDB International Conference on Very Large Databases, pages 507–518, 1987. (Cited on pages 13 and 35.)

[Rowlands 2012] David D Rowlands, Mitchell McCarthy and Daniel A James. *Using Inertial Sensors to Index into Video*. Procedia Engineering, vol. 34, pages 598–603, 2012. (Cited on page 12.)

[Šaltenis 2000] S. Šaltenis, C.S. Jensen, S.T. Leutenegger and M.A. Lopez. *Indexing the Positions of Continuously Moving Objects*. In Proceedings of the 2000

ACM SIGMOD international conference on Management of data, pages 331–342. ACM, 2000. (Cited on page 16.)

[Samet 2006] Hanan Samet. Foundations of Multidimensional and Metric Data Structures. Morgan Kaufmann, 2006. (Cited on page 15.)

[Samet 2013] Hanan Samet, Jagan Sankaranarayanan and Michael Auerbach. *Indexing Methods for Moving Object Databases: Games and Other Applications*. In International Conference on Very Large Data Bases, pages 169–180. ACM, 2013. (Cited on page 16.)

[Santos 2012] Artur Santos, João Marcelo Teixeira, Thiago Farias, Veronica Teichrieb and Judith Kelner. *Understanding the Efficiency of kD-tree Ray-traversal Techniques over a GPGPU Architecture*. International Journal of Parallel Programming, vol. 40, no. 3, pages 331–352, 2012. (Cited on page 15.)

[Seo 2012] Beomjoo Seo, Weiwei Cui and Roger Zimmermann. *An Experimental Study of Video Uploading from Mobile Devices with HTTP Streaming*. In ACM Multimedia Systems Conference, pages 215–225, 2012. (Cited on pages 6, 22 and 103.)

[Shen 2011] Zhijie Shen, Sakire Arslan Ay, Seon Ho Kim and Roger Zimmermann. *Automatic Tag Generation and Ranking for Sensor-rich Outdoor Videos*. In acmmm, pages 93–102. ACM, 2011. (Cited on page 70.)

[Singh 2007] Sarvjeet Singh, Chris Mayfield, Sunil Prabhakar, Rahul Shah and Susanne E. Hambrusch. *Indexing Uncertain Categorical Data*. In ICDE, IEEE International Conference on Data Engineering, pages 616–625, 2007. (Cited on page 17.)

[Sistla 1998] Prasad Sistla, Ouri Wolfson, Sam Chamberlain and Son Dao. *Querying the Uncertain Position of Moving Objects*. In Temporal Databases: Research and Practice, pages 310–337. Springer, 1998. (Cited on page 78.)

[Sodagar 2011] Iraj Sodagar. *The MPEG-DASH Standard for Multimedia Streaming Over the Internet*. MultiMedia, IEEE, vol. 18, no. 4, pages 62–67, 2011. (Cited on page 24.)

[Song 2001] Zhexuan Song and Nick Roussopoulos. *Hashing Moving Objects*. In Mobile Data Management, pages 161–172. Springer, 2001. (Cited on page 16.)

[Stockhammer 2011] Thomas Stockhammer. *Dynamic Adaptive Streaming over HTTP - Standards and Design Principles*. In ACM Multimedia Systems Conference, pages 133–144, 2011. (Cited on page 23.)

[Sundaram 2000] Hari Sundaram and Shih-Fu Chang. *Video Scene Segmentation using Video and Audio Features*. In Multimedia and Expo, 2000. ICME 2000. 2000 IEEE International Conference on, volume 2, pages 1145–1148. IEEE, 2000. (Cited on page 10.)

[Tao 2007] Yufei Tao, Xiaokui Xiao and Reynold Cheng. *Range Search on Multidimensional Uncertain Data*. In ACM Transactions on Database Systems, volume 32, pages 15–68, 2007. (Cited on page 17.)

[Theodoridis 2002] Y. Theodoridis, T. Sellis, A.N. Papadopoulos and Y. Manolopoulos. *Specifications for Efficient Indexing in Spatiotemporal Databases*. In Scientific and Statistical Database Management, 1998. Proceedings. Tenth International Conference on, pages 123–132. IEEE, 2002. (Cited on page 15.)

[Wang 2012] Guanfeng Wang, Beomjoo Seo and Roger Zimmermann. *Automatic Positioning Data Correction for Sensor-Annotated Mobile Videos*. In International Conference on Advances in Geographic Information Systems, pages 470–473, 2012. (Cited on page 6.)

[Wang 2013] Lin Wang, Bin Chen and Yuehu Liu. *Distributed Storage and Index of Vector Spatial Data based on HBase*. In Geoinformatics (GEOINFORMATICS), 2013 21st International Conference on, pages 1–5. IEEE, 2013. (Cited on page 14.)

[Xiong 2006] Xiaopeng Xiong, Mohamed F Mokbel and Walid G Aref. *LUGrid: Update-tolerant Grid-based Indexing for Moving Objects*. In Mobile Data Management, 2006. MDM 2006. 7th International Conference on, pages 13–13. IEEE, 2006. (Cited on page 16.)

[Xu 2012] Chenliang Xu, Caiming Xiong and Jason J Corso. *Streaming Hierarchical Video Segmentation*. In ECCV, European Conference on Computer Vision, pages 626–639. Springer, 2012. (Cited on page 10.)

[Yiu 2008] Man Lung Yiu, Yufei Tao and Nikos Mamoulis. *The B^{dual}-Tree: Indexing Moving Objects by Space Filling Curves in the Dual Space*. The VLDB Journal, vol. 17, no. 3, pages 379–400, 2008. (Cited on page 16.)

[YouTube 2013] YouTube. *YouTube Press Statistics*, 2013. URL: `http://www.youtube.com/t/press_statistics`. (Cited on page 1.)

[Zaharia 2009] Matei Zaharia, Dhruba Borthakur, J Sen Sarma, Khaled Elmeleegy, Scott Shenker and Ion Stoica. *Job Scheduling for Multi-user Mapreduce Clusters*. EECS Department, University of California, Berkeley, Tech. Rep. UCB/EECS-2009-55, 2009. (Cited on page 20.)

[Zencoder 2010] Zencoder. *Zencoder Cloud*, 2010. URL: `http://zencoder.com/en/cloud`. (Cited on page 19.)

[Zhai 2006] Yun Zhai and Mubarak Shah. *Video Scene Segmentation using Markov Chain Monte Carlo*. Multimedia, IEEE Transactions on, vol. 8, no. 4, pages 686–697, 2006. (Cited on page 10.)

[Zhang 2010] Ying Zhang, Xuemin Lin, Wenjie Zhang, Jianmin Wang and Qianlu Lin. *Effectively Indexing the Uncertain Space*. In TKDE, IEEE Transactions

on Knowledge and Data Engineering., volume 22, pages 1247–1261, 2010. (Cited on page 18.)

[Zhang 2012] Ying Zhang, Wenjie Zhang, Qianlu Lin and Xuemin Lin. *Effectively Indexing the Multi-dimensional Uncertain Objects for Range Searching.* In the 15th International Conference on Extending Database Technology, pages 504–515. ACM, 2012. (Cited on page 18.)

[Zhang 2013] Ying Zhang, He Ma and Roger Zimmermann. *Dynamic Multi-Video Summarization of Sensor-Rich Videos in Geo-Space.* In MMM 2013, the 19th International Conference on Multimedia Modeling, pages 380–390, 2013. (Cited on page 6.)

[Zhu 2005] Z. Zhu, E.M. Riseman, A.R. Hanson and H. Schultz. *An Efficient Method for Geo-referenced Video Mosaicing for Environmental Monitoring.* In Machine Vision and Applications, volume 16, pages 203–216. Springer, 2005. (Cited on page 11.)

[Zhu 2011] Wenwu Zhu, Chong Luo, Jianfeng Wang and Shipeng Li. *Multimedia Cloud Computing.* Signal Processing Magazine, IEEE, vol. 28, no. 3, pages 59–69, 2011. (Cited on page 100.)