January 2016

A Cost-Effective Cloud-Based System for Analyzing Big Real-Time Visual Data From Thousands of Network Cameras

Ahmed Mohamed
Purdue University

Graduate School Form
30 Updated 12/26/2015

PURDUE UNIVERSITY
GRADUATE SCHOOL
Thesis/Dissertation Acceptance

This is to certify that the thesis/dissertation prepared

By Ahmed Sayed Ahmed Kaseb Mohamed

Entitled
A Cost-Effective Cloud-Based System for Analyzing Big Real-Time Visual Data From Thousands of Network Cameras

For the degree of Doctor of Philosophy

Is approved by the final examining committee:

Yung-Hsiang Lu Chair	
Dongyan Xu	
Edward J. Delp	
Wei Tsang Ooi	

To the best of my knowledge and as understood by the student in the Thesis/Dissertation Agreement, Publication Delay, and Certification Disclaimer (Graduate School Form 32), this thesis/dissertation adheres to the provisions of Purdue University's "Policy of Integrity in Research" and the use of copyright material.

Approved by Major Professor(s): Yung-Hsiang Lu

Approved by: Venkataramanan Balakrishnan 12/5/2016
Head of the Departmental Graduate Program Date

A COST-EFFECTIVE CLOUD-BASED SYSTEM FOR ANALYZING BIG REAL-TIME VISUAL DATA FROM THOUSANDS OF NETWORK CAMERAS

A Dissertation

Submitted to the Faculty

of

Purdue University

by

Ahmed S. Kaseb

In Partial Fulfillment of the

Requirements for the Degree

of

Doctor of Philosophy

December 2016

Purdue University

West Lafayette, Indiana

To my family

ACKNOWLEDGMENTS

I would like to thank my advisor Prof. Yung-Hsiang Lu for his great guidance, support, help, and everything else. He has been the ideal advisor for me. His sage advice, constructive criticisms, and patient encouragement aided my research in innumerable ways. I would like to thank Prof. Edward J. Delp, Prof. Wei Tsang Ooi, and Prof. Dongyan Xu for serving on my committee and reviewing my work. I would like to thank my team members in the High Efficiency Low Power Systems (HELPS) research group for their help and collaboration.

I would like to thank my father for always being there during difficult times, my mother for her great love and prayers, my wife for supporting me for reasons not always obvious, and my four-year-old daughter and my four-month-old son for making everything possible.

TABLE OF CONTENTS

LIST OF TABLES

LIST OF FIGURES

Figure Page

Figure Page

ABBREVIATIONS

Amazon EC2	Amazon Elastic Compute Cloud
API	Application Programming Interface
CAM^2	Continuous Analysis of Many CAMeras
FPS	Frames Per Second
FT	Feature Tracking (an analysis program)
GPU	Graphics Processing Unit
HD	Human Detection (an analysis program)
IA	Image Archival (an analysis program)
IO	Input/Output
ME	Motion Estimation (an analysis program)
MOD	Moving Objects Detection (an analysis program)

ABSTRACT

Kaseb, Ahmed S. PhD, Purdue University, December 2016. A Cost-Effective Cloud-Based System for Analyzing Big Real-Time Visual Data from Thousands of Network Cameras. Major Professor: Yung-Hsiang Lu.

Thousands of network cameras stream public real-time visual data from different environments, such as streets, shopping malls, and natural scenes. The big visual data from these cameras can be useful for many applications, but analyzing this data presents many challenges, such as (i) retrieving data from heterogeneous cameras (e.g. different brands and data formats), (ii) providing a software environment for users to simultaneously analyze the large amounts of data from the cameras, (iii) allocating and managing significant amounts of computing resources. This dissertation presents a web-based system designed to address these challenges. The system enables users to execute analysis programs on the data from more than 120,000 cameras. The system handles the heterogeneity of the cameras and provides an Application Programming Interface (API) that requires slight changes to the existing analysis programs reading data from files. The system includes a resource manager that allocates cloud resources in order to meet the analysis requirements. Cloud vendors offer different cloud instance types with different capabilities and hourly costs. The manager reduces the overall cost of the allocated instances while meeting the performance requirements. The resource manager monitors the allocated instances; it allocates more instances if needed and deallocates existing instances to reduce the cost if possible. The manager makes decisions based on many factors, such as analysis programs, frame rates, cameras, and instance types.

1. INTRODUCTION

Over the past years, millions of network cameras have been deployed around the world [1] resulting in an unprecedented amount of big real-time visual data. The cameras stream real-time visual data from a variety of environments, such as streets, shopping malls, landmarks, and natural scenes. The visual data can be analyzed for many applications, such as traffic monitoring, surveillance, and weather detection. From the same data stream, meteorologists may study the formation of storms, city planners may review traffic management, and emergency responders may consider evacuation routes. Our team has discovered more than 120,000 public cameras deployed by departments of transportation, universities, companies, individuals, etc. This tremendous amount of visual data is lost if not analyzed. Hence, there is a need to analyze this data for a better understanding of the world around us.

Meanwhile, hundreds of image and video analysis programs are developed for a wide range of applications. *How could we use these programs to analyze the visual data from many cameras?* Simultaneously analyzing data streams from thousands of cameras presents many challenges, such as (i) retrieving data from heterogeneous cameras, (ii) providing a software environment for users to simultaneously analyze large amounts of data from the cameras, (iii) allocating and managing significant amounts of resources (e.g., CPU, memory). This dissertation presents CAM^2 (Continuous Analysis of Many CAMeras) as a system designed to address these challenges. Chapter 2 overviews the related work. Chapter 3 introduces CAM^2 as a system for large-scale analysis of the visual data from network cameras. Chapter 4, Chapter 5, and Chapter 6 propose various resource managers that can be used by CAM^2.

1.1 CAM^2: System, Website, and API (Chapter 3)

Chapter 3 introduces CAM^2 as a system that addresses the following problems: (i) Given image analysis programs, how can these programs be executed on a very large scale without adding much burden to the user? A user may develop analysis programs that can detect weather conditions, analyze traffic, monitor crowd, etc. CAM^2 takes responsibility for executing the analysis programs on the data from thousands of cameras. (ii) How can these analysis programs be executed on cameras selected by the user, and at the desired frame rates for the desired durations? CAM^2 retrieves data from the selected cameras and executes the analysis programs based on the specified parameters. (iii) How can data be retrieved from many heterogeneous cameras, i.e., different brands, data formats, frame sizes, etc.? CAM^2 simplifies migrating existing analysis programs by providing a simple Application Programming Interface (API) that hides the heterogeneity and requires only slight changes to the existing analysis programs. CAM^2 allocates cloud instances to meet the computation and storage requirements of the analysis.

Chapter 3 also describes how to use the website [2] and the API of CAM^2. A user can upload, execute, and download the results of analysis programs using the website by following this procedure:

1. View a world map with the geotagged cameras along with their recent snapshots.
2. Select the cameras to analyze using a variety of selection methods, e.g. timezone, country, state, city.
3. Specify the desired analysis frame rate and duration.
4. Upload the analysis program that uses the API of CAM^2. The API is event-driven: when new frames arrive, the analysis program is invoked. This event-driven API significantly simplifies the analysis program. The API requires slight changes to the existing analysis program; only the IO operations are modified.
5. Download the analysis results.

The experiments in Chapter 3 show that CAM^2 is capable of analyzing 2.7 million images from 1274 cameras over three hours using 15 Amazon Elastic Compute Cloud (Amazon EC2) [3] instances. That is more than 141 GB of images (at 107 Mbps). The average frame size of the cameras is 0.44 Mega Pixels (MP). The experiments use two analysis programs: motion estimation and human detection.

1.2 Cloud Resource Management (Chapter 4)

A major challenge in CAM^2 is the ability to allocate and manage significant amounts of resources (CPU, memory, etc.) to analyze thousands of data streams simultaneously. Analyzing a single image (assuming 100KB per image) every one minute from 120,000 cameras means analyzing more than 10 TB of visual data per day. Using the cloud for this big data analysis is desirable due to the elasticity of resources. Cloud vendors offer many instance types with different CPU, memory, and network capabilities. Choosing the right instance type for the analysis can be more cost-effective than other instance types. The right instance type depends on both the capabilities of the instance and the requirements of the analysis.

It is a challenging problem to manage cloud resources in order to reduce the cost of analyzing data streams from thousands of cameras while meeting the performance requirements. This problem can be divided into two parts: resource allocation and resource scaling. Regarding resource allocation, the manager decides what types of cloud instances to use, how many instances to allocate, and which cameras to assign to which instances. Regarding resource scaling, the manager decides how to handle resource overutilization or underutilization, and when to scale up or down the number of running instances. All these decisions are affected by many factors, such as the analysis programs, the desired frame rates, the frame sizes of the cameras, the visual content from the cameras, the types and costs of the instances. Chapter 4 proposes a resource manager that tackles these problems.

The resource manager allocates cost-effective cloud instances based on evaluating the resource requirements of analysis programs and assessing the effective cost of using different cloud instances. The manager continuously monitors the CPU and memory utilization of the allocated instances to ensure that they are not overutilized or underutilized. The resource manager automatically scales up or down the number of running instances by allocating more instances if needed and deallocating existing instances to reduce the cost if possible. The manager migrates data streams among instances in order to reduce the overall analysis cost and meet the performance requirements. If the same user executes multiple analysis programs at different times, the manager can reuse the running instances to reduce the overall analysis cost.

To evaluate the resource manager, the experiments use four analysis programs that represent different workloads in terms of CPU and memory: image archival, motion estimation, moving objects detection, and human detection. The experiments show that different cloud instances are more cost-effective for different analysis programs. One experiment analyzes data streams from 1026 cameras simultaneously for six hours using different analysis programs at different frame rates. The experiment analyzes 5.5 million images (260GB data), and costs \$12.77. Without using the proposed resource manager, this experiment costs \$14.63. In other words, the proposed resource manager leads to a 13% reduction in cost.

1.3 Enhanced Resource Management (Chapter 5)

Chapter 5 proposes an enhanced resource manager that improves the first manager (Chapter 4) in many ways. For example, the enhanced manager: (i) is able to handle multiple analysis programs at different frame rates, (ii) considers the desired frame rates and the camera frame sizes while estimating the resource requirements of analyzing the data stream from each camera, (iii) models the resource allocation problem as a 2D vector bin packing problem [4] and solves it using a greedy heuristic algorithm, (iv) avoids conducting test runs on all instances and for different frame rates by modeling the relationship between the frame rate and the resource requirements.

To evaluate the enhanced resource manager, the experiments evaluate the effect of many factors (e.g., camera frame sizes and instance types) on the resource management decisions. The experiments also show that the resource manager is able to reduce up to 61% of the overall cost. One experiment analyzes more than 97 million images (3.3 TB of data) from 5,310 cameras simultaneously over 24 hours using 15 instances. The experiments use five analysis programs to represent different workloads: image archival, motion estimation, moving objects detection, features tracking, and human detection.

1.4 Management of CPU and GPU Resources (Chapter 6)

Some cloud instances have GPUs and some instances do not (referred to as GPU instances and non-GPU instances respectively). Using GPUs can accelerate analysis programs and achieve higher frame rates, but incurs additional cost because GPU instances are more expensive. This chapter proposes a resource manager that uses the GPU to achieve frame rates that are not possible using the CPU only. The manager also considers both GPU and non-GPU instances to reduce the overall cost. To achieve that, the manager conducts a test run to estimate the resource requirements of analysis programs if they are executed using the CPU or the GPU. The manager formulates the resource allocation problem as a multiple-choice vector bin packing problem and decides what instance types to use, how many instances to allocate, which camera streams to assign to which instances, and which CPU or GPU to analyze each stream.

To evaluate the proposed manager, the experiments use two analysis programs for object detection. Each program uses a convolutional neural network (VGG-16 [5] or ZF [6]) to detect objects (e.g. persons and cars) in images. The experiments show that the manager can use the GPU to achieve a speedup of around 13 (or 16) for

VGG-16 (or ZF). The experiments evaluate the effect of the desired frame rates and the number of cameras on the resource requirements. The experiments show that each analysis program has two choices of resource requirements depending on whether it is executed by the CPU or the GPU. The experiments compare the resource allocation strategy of the manager with other allocation strategies and show that the manager can reduce up to 61%.

1.5 Contributions

The main contributions of this dissertation can be summarized as follows:

1. To our knowledge, CAM^2 is the first and probably the only system that enables users to simultaneously analyze real-time image and video streams from thousands of network cameras. CAM^2 has a public website and can be used for a variety of applications.

2. CAM^2 provides an API that makes it easy to migrate existing analysis programs with only slight changes. In particular, only the IO operations are modified.

3. CAM^2 provides access to more than 120,000 worldwide distributed cameras discovered by our team. The system handles the heterogeneity of these cameras.

4. It proposes a cloud resource manager aiming at reducing the cost of analyzing data streams from network cameras while meeting the performance requirements. This is achieved by allocating cost-effective instances, monitoring and automatically scaling the cloud resources. The experiments show that the resource manager can reduce up to 13% cost.

5. It proposes an enhanced resource manager that allocates and scales the cloud resources based on many factors, including: (i) the resource requirements of the analysis programs, (ii) the desired frame rates, (iii) the frame sizes of the cameras, (iv) the types and costs of the cloud instances, (v) and the utilization of the running instances.

6. The enhanced resource manager models the resource allocation problem as a 2D vector bin packing problem [4] and solves it using a greedy heuristic algorithm. The experiments show that the enhanced manager can reduce up to 61% cost.

7. It proposes a resource manager that uses the GPU to achieve frame rates that are not possible using the CPU only and considers both GPU and non-GPU instances to reduce the overall cost. The manager formulates the resource allocation problem as a multiple-choice vector bin packing problem and solves it using an existing algorithm. The experiments show that the manager is able to reduce 61% of the cost compared with other allocation strategies.

2. RELATED WORK

2.1 Network Cameras: Sources and Analysis

There are many sources for the visual data from network cameras. AMOS (Archive of Many Outdoor Scenes) [7] is a dataset that contains more than 800 million images captured from nearly 30,000 cameras since 2006. The dataset can be downloaded for offline analysis. Recent visual data from many traffic cameras are publicly available through the websites of the Departments of Transportation (DOT), such as New York City DOT (`http://nyctmc.org/`) and Massachusetts DOT (`http://www1.eot.state.ma.us/`). Some websites (e.g., `http://www.webcams.travel/` and `http://www.wunderground.com/webcams/`) show recent snapshots from thousands of public cameras.

Network cameras have been used by researchers for many applications, such as: surveillance [8], detecting anomalous activities in crowded scenes [9], studying vehicular traffic and mobility models [10], and improving physics-based illumination models [11]. Cameras have also been used to study various environmental issues, such as: detecting weather conditions [12] [13], monitoring vegetation [14], monitoring plant phenology [15], measuring water quality [16], measuring water levels [17], evaluating the composition of water [18], and monitoring foam formation downstream of wastewater treatment [19].

Many systems have been built to analyze the visual data from network cameras. Hong et al. [20] proposed a distributed framework for spatio-temporal analysis applications on large-scale camera networks. IBM Smart Surveillance System [21] is a system for large-scale video analytics for surveillance applications. Yu et al. [22] proposed a system for near real-time video stream analysis. They developed a prototype for video surveillance in a retail store using several cameras. Target Container [23] is a framework enabling users to track multiple targets in a camera network. CAM^2 is different because it provides the visual data from the cameras, reduces the cost of using the cloud to analyze the data streams from thousands of cameras simultaneously, and can be used for a wide range of applications.

2.2 Cloud Resource Management

Many studies have been conducted on managing cloud resources for multimedia applications, such as image and video analysis. Zhu et al. [24] explained the benefits of using the cloud for multimedia applications without considering the cost. Several papers considered applications streaming data *out of* the cloud [25] [26]. Hossain et al. [27] considered minimizing the number of allocated instances for cloud-based video surveillance applications and evaluated their solution by simulation. Vijaykumar et al. [28] introduced a dynamic resource allocation algorithm to minimize the overall cost of using the cloud for data streaming applications. Their solution primarily considered the CPU utilization. In contrast, this dissertation considers streaming data *into* the cloud for analysis, reduces the cost of using the cloud, considers both the CPU and memory resources to perform both image and video analysis, and is evaluated using Amazon EC2.

GPUs have been used to accelerate general purpose computation, such as image processing and computer vision [29]. Different studies used GPUs for face detection [30], motion estimation [31], body tracking [32], etc. This dissertation considers using GPUs to accelerate and reduce the monetary cost of analyzing the real-time multimedia content from network cameras using the cloud.

2.3 Amazon Cloud Services

Amazon EC2 [3] provides three related services to the proposed resource managers: (i) CloudWatch (`https://aws.amazon.com/cloudwatch/`) for monitoring the CPU utilization of instances. (ii) Auto Scaling (`https://aws.amazon.com/autoscaling/`)

for scaling the number of running instances up or down automatically if they are overutilized or underutilized. Users can specify a single instance type to be used for launching new instances. (iii) Elastic Load Balancing (`https://aws.amazon.com/elasticloadbalancing/`) for distributing incoming web traffic across instances using Round-Robin or Least Outstanding Request. The proposed managers are different because they monitor both the CPU and memory utilization of the instances, launch instances of different types based on the existing workloads such that the overall cost is reduced, and balance the load among the instances based on their resource utilization.

2.4 Our Relevant Work

This dissertation is based on our relevant work: We introduced CAM^2 as a system for analyzing visual data from network cameras [33]. The website and the Application Programming Interface (API) of CAM^2 [34] enable users to submit their analysis programs. We described the challenges and the potential applications of CAM^2 [35] and developed an Android mobile application that enables users to watch the world and plan their routes using the cameras in CAM^2 [36]. Hacker and Lu [37] presented CAM^2 as an educational tool to teach students big data analytics. Then, we proposed cloud resource managers reducing the cost of analyzing image and video streams from network cameras [38] [39] [40] [41]. Table 2.1 compares the resource managers in our relevant work.

2.5 Bin Packing

In the bin packing problem [42], it is required to pack different-sized objects into unit-size bins. The objective is to minimize the number of used bins while maintaining the overall size of the objects in any bin less than one. The bin packing is an NP-hard problem [43]. Several generalizations of the bin packing problem have been studied:

Table 2.1.: Comparison of our resource management work: Chapter 4 [38], [39], Chapter 5 [40], and Chapter 6 [41].

Criteria	[38]	[39]	[40]	[41]
Cameras				
Allocate more resources for higher frame size cameras			✓	✓
Evaluate the effect of the camera frame sizes on the resource requirements			✓	
Evaluate the effect of the camera visual content on the resource requirements			✓	
Consider the camera locations		✓		
Analysis Programs				
Consider GPU execution				✓
Propose an allocation procedure that can handle multiple analysis programs at different frame rates			✓	✓
Evaluate the effect of the analysis frame rates on the resource requirements			✓	✓
Cloud Instances				
Consider GPU resources				✓
Consider memory resources	✓		✓	✓
Use different instance families	✓		✓	✓
Reuse running instances for newly launched analysis	✓		✓	✓
Consider the instance locations		✓		

1. Variable Sized Bin Packing [44] [45] [46] allows bins to have different sizes. The objective is to minimize the overall size of the used bins.
2. Vector Bin Packing [47] [48] [49] uses a multidimensional vector (as opposed to a scalar) for the size of each object or bin. The overall size of the objects in any bin in any dimension must not exceed one.
3. Vector Bin Packing with Heterogeneous Bins [4] allows bins to have different sizes and costs. The objective is to minimize the overall cost of the used bins. This problem is used by the resource manager in Chapter 5 to formulate the resource allocation problem.
4. Multiple-Choice Vector Bin Packing [50] [51] allows multiple choices for the size of each object. In other words, each object may have one of several possible sizes. This problem is used by the resource manager in Chapter 6 to formulate the resource allocation problem.

Table 2.2.: The classification of the bin packing related work.

	1D	Multidimensional
Homogeneous Bins	[42]	[47] [48] [49]
Heterogeneous Bins	[44] [45] [46]	[52] [50] [4] [40]

2.6 Image and Video Analysis

To evaluate the proposed manager, this dissertation uses existing image and video analysis techniques for background subtraction [53] [54], feature detection [55], feature tracking [56], human detection [57], and object detection [5] [6].

3. CAM^2: SYSTEM, WEBSITE, AND API

This chapter is organized as follows: Section 3.1 describes the architecture of CAM^2 and the heterogeneity of the existing cameras. Section 3.2 demonstrates the use of CAM^2. Section 3.3 shows how to use the website of CAM^2 to execute analysis programs on a large-scale. Section 3.4 describes how to use the API of CAM^2 in order to migrate existing analysis programs to CAM^2.

3.1 Overview of CAM^2

CAM^2 is a cloud-based system for the analysis of the visual big data from network cameras. To our knowledge, CAM^2 is the first system that enables users to simultaneously analyze real-time image and video streams from thousands of network cameras. Figure 3.1 shows the architecture of CAM^2 with the following main components:

1. The website is users' portal to select cameras and upload analysis programs.
2. The database maintains the information about the cameras, such as brands, data formats, and frame sizes.
3. The resource manager allocates and manages cloud instances to execute the analysis programs.
4. The instances retrieve the visual data from the cameras and execute the analysis programs.

CAM^2 provides access to more than 120,000 network cameras. The cameras are deployed by various organizations, including departments of transportation, universities, companies, or individuals. The cameras provide unprecedented amount of information that can help us understand the world better. The ultimate goal of CAM^2

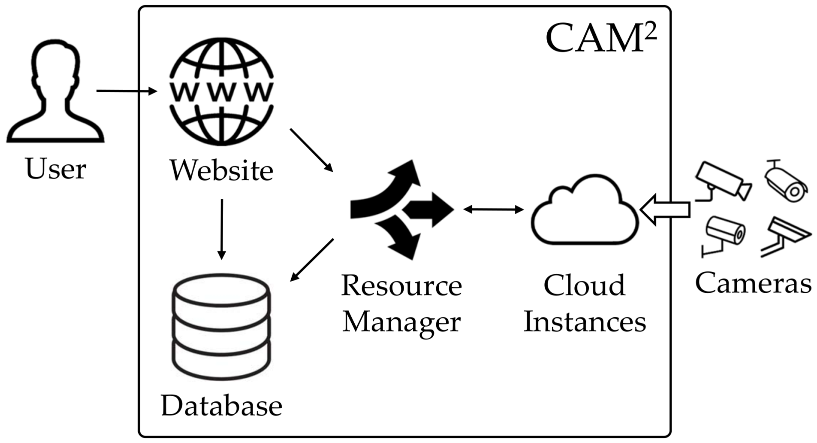


Fig. 3.1.: The architecture of CAM^2.

is to bridge the gap between users with their analysis programs and the thousands of online public cameras. In other words, the goal is to enable the users to execute their analysis programs on the visual data from the cameras. To achieve that, CAM^2 has three main design goals: flexibility, ease of use, and scalability.

Flexibility: CAM^2 achieves flexibility by not assuming any prior knowledge about the analysis programs. CAM^2 can be used for a wide range of image analysis and computer vision applications, such as traffic monitoring, surveillance, weather detection, etc.

Ease of use: CAM^2 reduces the burden on the users. The users are responsible for only using the website and uploading the analysis programs. In addition:

1. CAM^2 handles the heterogeneity of the cameras, i.e., different brands, data formats, frame sizes, etc.The API provides a uniform way for the analysis programs to access the images from all the cameras.

2. CAM^2 handles the underlying computing infrastructure. CAM^2 uses cloud resources to meet the computation and storage requirements of the large-scale analysis. The users do not need to worry about cloud computing resource management. Instead, the users can focus only on their analysis programs.

3. The API of CAM^2 requires few changes to the existing analysis programs. In particular, only the IO operations are changed. This enables the users to migrate their analysis programs easily. More details about the API are discussed in Section 3.4.

Scalability: CAM^2 has to be scalable in order to analyze the tremendous amount of data from the thousands on cameras. CAM^2 handles scalability by using both private and public cloud resources. The system allocates and manages Amazon EC2 and Microsoft Azure cloud instances in order to meet the computation and storage requirements of large-scale analyses. This system is designed such that the cloud instances communicate directly with the cameras without going through the server of CAM^2. This reduces the latency and enhances the scalability of the system.

The cameras in CAM^2 are heterogeneous in many ways:

Types: IP cameras have known public IP addresses and can provide real-time image or video streams. Non-IP cameras are available only through some websites that provide recent snapshots periodically.

Brands and Data Formats: Each brand (e.g. Axis, Panasonic) has a different way of communication and supports different data formats (e.g. image, MJPEG, H.264). All the cameras can provide individual images. MJPEG is the most widely supported video format. Some newer cameras support H.264 as well.

Frame Sizes: The cameras provide images and videos in different frame sizes. One of the most common frame sizes is 640×480. More than 1,500 cameras have frame sizes above 1 Megapixels.

Frame Rates: Figure 3.2 shows a histogram for the video frame rates of more than 900 IP cameras providing MJPEG streams. The camera frame rates depends on the network distance between the instance and the cameras. The frame rates in the figure are measured using an instance in Oregon.

Visual Content: Figure 3.3 shows the heterogeneity of the visual content from the cameras in CAM^2. The cameras provide a variety of scenes, such as tourist attractions, highways, etc. Hence, the cameras can be used for a variety of applications.

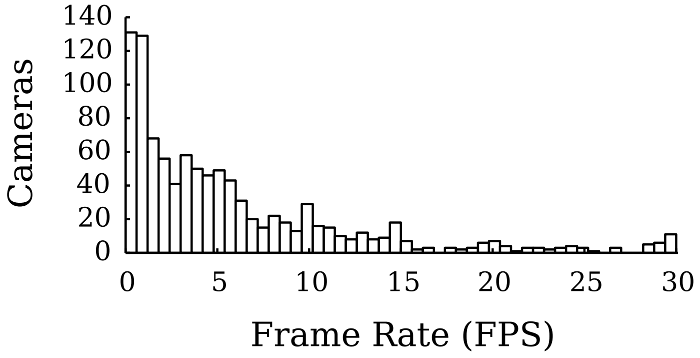


Fig. 3.2.: The heterogeneity of the frame rates of more than 900 IP cameras in CAM^2.

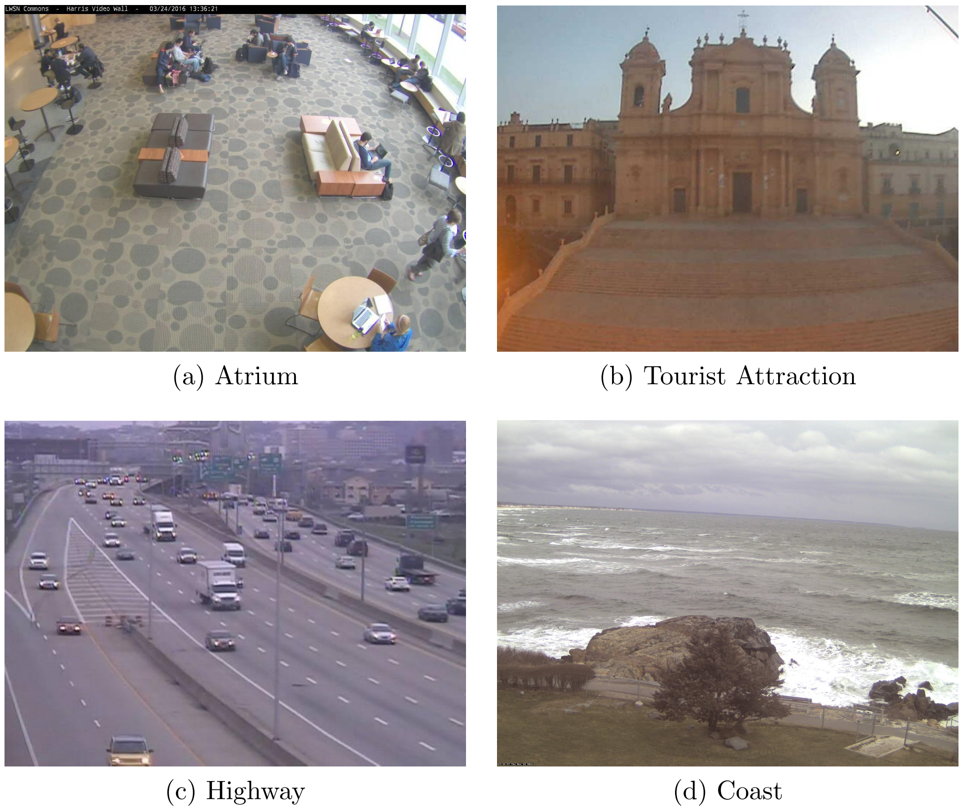


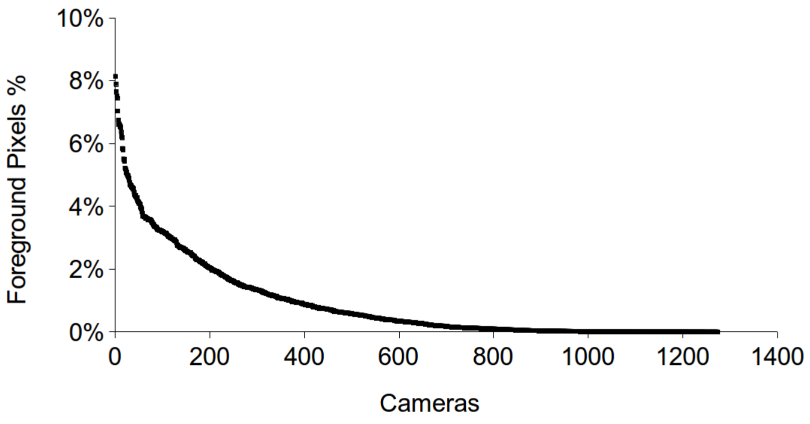
Fig. 3.3.: The heterogeneity of the visual content from the cameras in CAM^2.

3.2 CAM^2 Demonstration

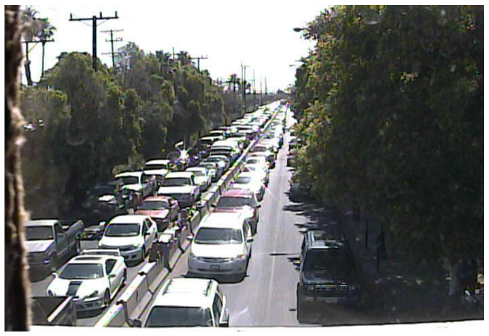
The primary goal of CAM^2 is to enable users to execute a variety of programs to analyze the data from thousands of cameras. The responsibility of users is to submit their programs (in Python) that analyze individual frames. Users can select cameras based on different criteria, such as country, state, city, and timezone. The system is responsible for executing the submitted analysis program for all the selected cameras, and providing the aggregated results as well. In order to meet the resource requirements, the system allocates and manages cloud instances.

In order to demonstrate the capability of CAM^2 to perform large-scale analysis, we conducted an experiment that analyzed 2.7 million images from 1274 cameras over three hours using one frame every five seconds from each camera. During the experiment, the system used 15 Amazon cloud instances to analyze 141 GB of images at 107 Mbps ($141 \times 8 \times 1024/(3 \times 60 \times 60)$). The average frame size of the cameras is 0.44 MP (approximately 768×576). The cameras are deployed across North America, West Europe, and East Asia. The experiment calculates the average amount of motion in the images from each camera. First, the foreground of each frame is detected using OpenCV's implementation of the method proposed by KaewTraKulPong and Bowden [53]. Then, the percentage of the foreground pixels with respect to the entire image is calculated. The average percentage of foreground pixels over a period of time is an indication of the average amount of motion in the images.

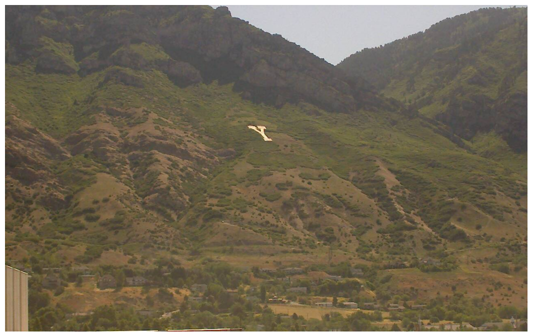
Figure 3.4 shows the results of the experiment. Figure 3.4 (a) shows the distribution of the average amount of motion for all the cameras. The horizontal axis represents the cameras, while the vertical axis represents the average percentage of foreground pixels. This experiment indicates that 84% cameras had less than 1% foreground pixels and 2% cameras did not have any detected motion at all over the period of three hours. Since the motion indicates the existence of new information from a camera, this experiment demonstrates the ability of CAM^2 to analyze the data from thousands of cameras simultaneously, and identify which cameras deserve



(a)



(b)



(c)

Fig. 3.4.: Results of estimating the average motion in 2.7 million images from 1274 cameras over three hours using one frame every five seconds. (a) Distribution of the average percentage of foreground pixels for all the cameras. (b) A camera in Mexico with high motion. (c) A camera in USA with low motion.

further investigation due to high degrees of motion. Figure 3.4 (b) shows a snapshot from a high-motion camera that is monitoring traffic, while figure Figure 3.4 (c) shows a snapshot from a low-motion camera that is looking at a landscape. It should be noted that panning cameras usually have the most amount of motion due to the frequent and sudden scene changes.

CAM^2 allocates cloud resources in order to meet the analysis requirements. The system is able to distribute the loads based on the different capabilities of each cloud instance. In this experiment, CAM^2 allocated 15 instances in order to meet the frame rate requirements. The achieved frame rate was 99% of the desired frame rate (0.2 FPS). Several factors could negatively affect the achieved frame rate: (i) Too few cloud instances are allocated and these instances are overloaded. The achieved frame rates drop noticeably when the CPU or memory utilization of the instances is over 90%. (ii) The cameras are geographically far from the cloud instances. For example, a cloud instance in USA analyzes the data from a camera in Europe.

3.3 CAM^2 Interactive Website

The website of CAM^2 is the users' portal to the system. The users need to learn about only the website in order to execute their CAM^2-compatible analysis programs on a large-scale. Through the website, the users can browse the cameras, select the cameras to analyze, set execution parameters (e.g. the analysis frame rate), upload their analysis programs, start and track the progress of their submissions, and download the analysis results.

The website of CAM^2 presents the cameras through an interactive map that shows a marker at the location of each camera as shown in Figure 3.5. The number of the cameras is large, and a marker for each camera would cause the map to load slowly. Hence, the website groups the camera markers into clusters, each showing the number of the cameras in the cluster. The website shows a recent snapshot from a camera when its marker is clicked. CAM^2 periodically downloads camera snapshots and stores them locally in the server in order to reduce the latency of showing camera snapshots.

To execute an analysis program on a set of cameras, a user should upload the program and select the set of cameras to analyze. The user might need to execute the same program on different sets of cameras, execute different analysis programs on the same set of cameras, or execute the same analysis program on the same set of cameras at different times. To enable this flexibility, CAM^2 offers a three-step process for executing analysis programs as shown in Figure 3.6:

1. Create a *configuration* by selecting the desired set of cameras to analyze and setting some execution parameters, e.g. the analysis frame rate.

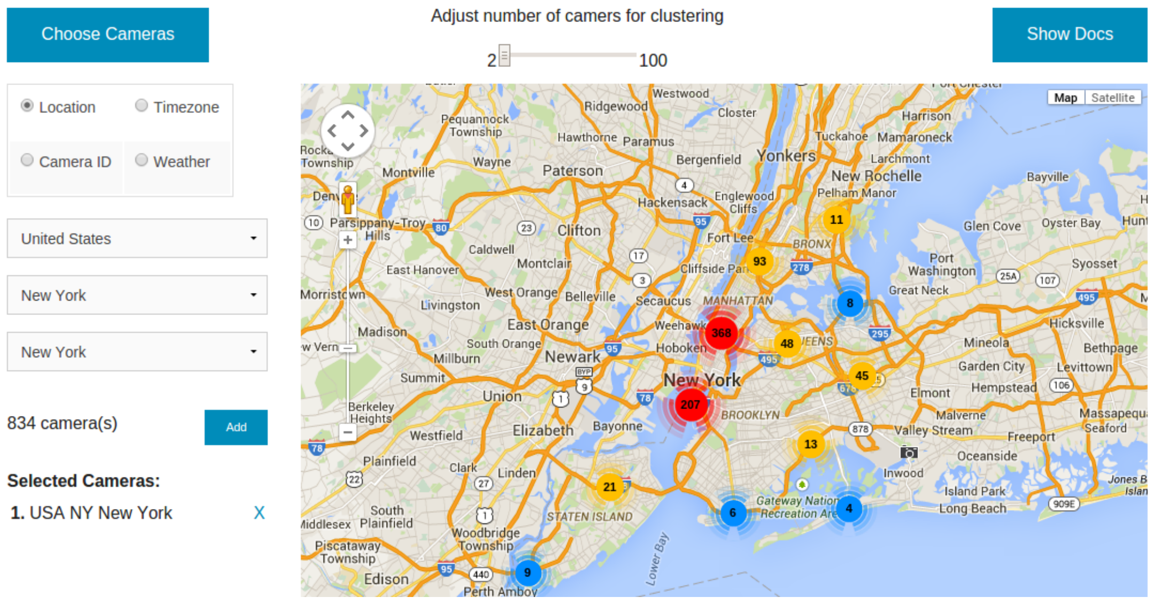


Fig. 3.5.: Using the website to browse the cameras through an interactive map and to select the cameras for analysis.

2. Upload a *module* which is a file that contains the source code of an analysis program. Currently, CAM^2 supports the modules that are written in Python and use OpenCV [58].
3. Start a *submission*, i.e. execute the analysis program in a selected *module* using the parameters of a selected *configuration*.

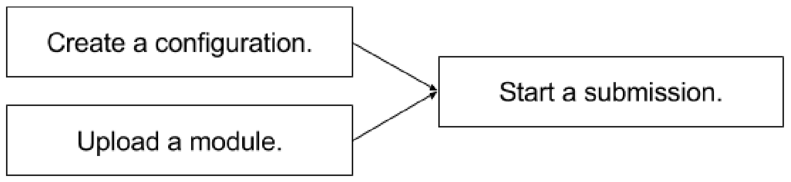


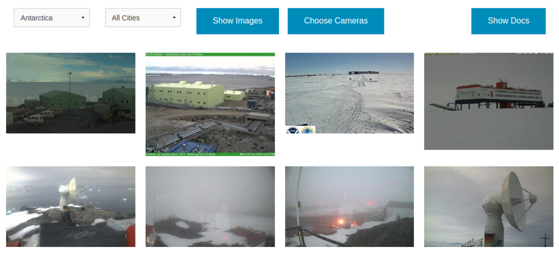
Fig. 3.6.: CAM^2 three-step process for executing analysis programs

In the next three sections, we present this three-step process. We show how this process enables users to execute analysis programs for a variety of applications. For more details, the website [2] of CAM^2 provides an online documentation and video tutorials illustrating the details about using the website and the API.

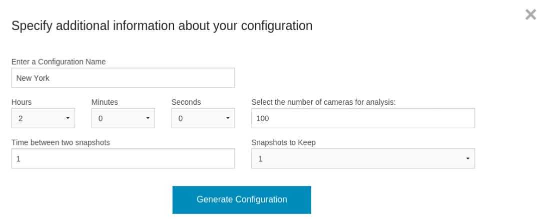
3.3.1 CAM^2 Configuration: Camera Selection and Execution Parameters

To create a configuration, a user should select the set of cameras to analyze and set some execution parameters, such as the analysis frame rate. Enabling the user to select a set of cameras for a variety of applications is indeed a challenge. In order to suit various applications, CAM^2 provides 6 ways of selecting cameras:

1. **Country, state, or city:** The user can select the cameras in a particular country, state, or city. This is essential if the user wants to analyze the data in a particular area, e.g. monitoring the traffic in Washington DC or Paris.
2. **Timezone:** The user can select the cameras in a particular timezone. This is useful if the uploaded analysis program has restrictions on the visual content from the cameras. For example, the analysis program might be designed to analyze outdoor images with high brightness, so the images should be taken during the daytime.
3. **Weather conditions:** The user can select the cameras whose cities have particular weather conditions (rain, wind, etc.). This is useful if the user wishes to execute a weather-related analysis program. For example, the user wants to execute a rain detection analysis program on the cameras that are likely to have rain. CAM^2 uses online weather services to retrieve the weather conditions of different cities.
4. **Camera IDs:** CAM^2 assigns a unique and fixed ID to each camera. If the user has a priori knowledge of the IDs of some particular cameras, the user can directly select these cameras using their IDs. The user can know the IDs of the cameras using the website.
5. **Camera map:** The user can select individual cameras through the interactive world map. When the user clicks a camera marker, the website shows a recent snapshot. If the snapshot is suitable for the user's analysis purposes, the user can add the camera to the desired camera set.



(a) Browsing and selecting the cameras for the analysis based on their visual content.

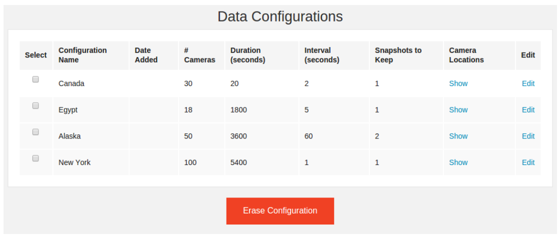

Specify additional information about your configuration

Enter a Configuration Name: New York

Hours: 2
Minutes: 0
Seconds: 0
Select the number of cameras for analysis: 100

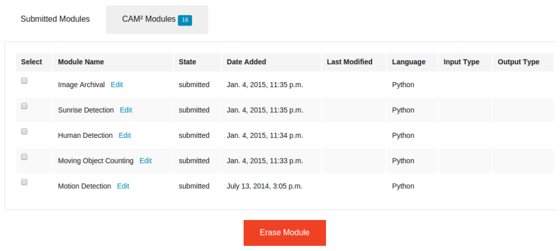
Time between two snapshots: 1
Snapshots to Keep: 1

(b) Setting the execution parameters of a configuration.


Data Configurations

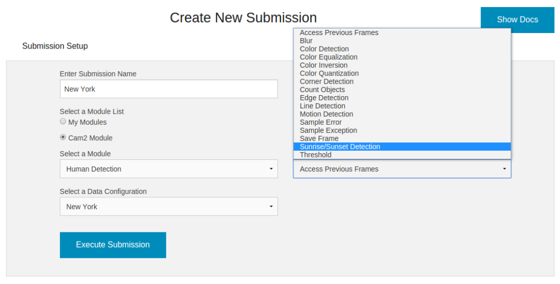
Select	Configuration Name	Date Added	# Cameras	Duration (seconds)	Interval (seconds)	Snapshots to Keep	Camera Locations	Edit
☐	Canada		30	20	2	1	Show	Edit
☐	Egypt		18	1800	5	1	Show	Edit
☐	Alaska		50	3600	60	2	Show	Edit
☐	New York		100	5400	1	1	Show	Edit

(c) Browsing, editing, and erasing the saved configurations.



Select	Module Name	State	Date Added	Last Modified	Language	Input Type	Output Type
☐	Image Archival Edit	submitted	Jan. 4, 2015, 11:35 p.m.		Python		
☐	Sunrise Detection Edit	submitted	Jan. 4, 2015, 11:35 p.m.		Python		
☐	Human Detection Edit	submitted	Jan. 4, 2015, 11:34 p.m.		Python		
☐	Moving Object Counting Edit	submitted	Jan. 4, 2015, 11:33 p.m.		Python		
☐	Motion Detection Edit	submitted	July 13, 2014, 3:05 p.m.		Python		

(d) Browsing, editing, and erasing the uploaded modules.


Create New Submission

Submission Setup

Enter Submission Name: New York

Select a Module List
○ My Modules
● Cam2 Module

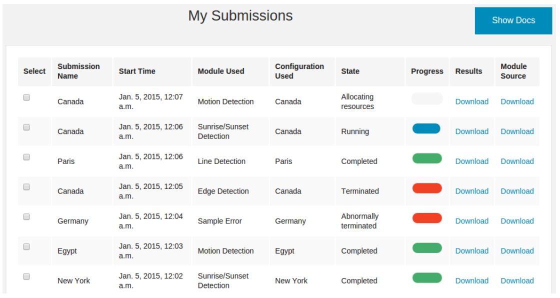
Select a Module: Human Detection

Select a Data Configuration: New York

Access Previous Frames
Blur
Color Detection
Color Equalization
Color Inversion
Color Quantization
Corner Detection
Count Objects
Edge Detection
Line Detection
Motion Detection
Sample Error
Sample Exception
Save Frame
Sunrise/Sunset Detection
Threshold

Access Previous Frames

(e) Starting a submission by selecting a module and a configuration.


My Submissions

Select	Submission Name	Start Time	Module Used	Configuration Used	State	Progress	Results	Module Source
☐	Canada	Jan. 5, 2015, 12:07 a.m.	Motion Detection	Canada	Allocating resources		Download	Download
☐	Canada	Jan. 5, 2015, 12:06 a.m.	Sunrise/Sunset Detection	Canada	Running	🟦	Download	Download
☐	Paris	Jan. 5, 2015, 12:06 a.m.	Line Detection	Paris	Completed	🟩	Download	Download
☐	Canada	Jan. 5, 2015, 12:05 a.m.	Edge Detection	Canada	Terminated	🟥	Download	Download
☐	Germany	Jan. 5, 2015, 12:04 a.m.	Sample Error	Germany	Abnormally terminated	🟥	Download	Download
☐	Egypt	Jan. 5, 2015, 12:03 a.m.	Motion Detection	Egypt	Completed	🟩	Download	Download
☐	New York	Jan. 5, 2015, 12:02 a.m.	Sunrise/Sunset Detection	New York	Completed	🟩	Download	Download

(f) Tracking the progress of, downloading the results of, and terminating the submissions.

Fig. 3.7.: Website Screenshots

6. **Visual content:** The user can select the cameras based on their visual content. The website can show a grid of recent snapshots from all the cameras in a selected country, state, or city. For example, Figure 3.7(a) shows the recent snapshots from the cameras in Antarctica. Then, the user can select the cameras that are suitable for the user's analysis purposes.

The user can select the cameras using multiple methods based on the desired applications. For example, the user might select the USA cameras that are likely to have rain, or the Eastern Time Zone cameras that are likely to have high wind speeds, etc. This can be beneficial for some computer vision applications, such as weather detection.

After selecting the cameras, the user should set the execution parameters which include: (i) the total duration of the analysis, (ii) the analysis frame rate, (iii) the limit on the number of cameras to analyze, and (iv) the number of past frames that are kept by CAM^2 and provided to the analysis program. This is essential if the analysis program needs to access old frames while processing new ones. Figure 3.7(b) shows a configuration that can analyze 100 cameras for 2 hours at 1 frame per second. CAM^2 executes the uploaded analysis program for the specified duration using the specified frame rate on the selected cameras. The user might need to execute different analysis programs using the same configuration (i.e. using the same set of cameras with the same execution parameters). That is why CAM^2 saves the created configuration so that it can be used later. The user can browse, edit, or erase the existing configurations as shown in Figure 3.7(c).

3.3.2 CAM^2 Module: The Analysis Program

A module is a file that contains the source code of an analysis program. In Section 3.4, we discuss how to write a CAM^2 module in details. A user can write a module to monitor the traffic, detect the weather conditions, etc. The user might need to execute the same analysis program on different sets of cameras. For example, the user might need to monitor the traffic in New York City, and later decide to monitor the traffic in Paris. That is why CAM^2 saves the uploaded module so that it can be used later. The user can browse, edit, or erase the uploaded modules as shown in Figure 3.7(d). The website of CAM^2 provides a dozen pre-written modules. These modules have a variety of analysis programs, such as motion detection, moving objects detection, sunrise/sunset detection, etc. These modules are beneficial for the users who wish to try CAM^2 or to learn how to use it. These modules are also useful for non-experts and students to learn about image processing and computer vision.

3.3.3 CAM^2 Submission: Executing the Analysis Program

To start a submission, a configuration and a module should be selected as shown in Figure 3.7(e). Then, CAM^2 takes the responsibility of executing the analysis program in the uploaded module on the cameras and with the parameters specified by the selected configuration. The module can be a user module or a CAM^2 pre-written module. Figure 3.7(f) shows how the user can track the progress and the current state of the submission. As shown in Figure 3.8, the submission can be in one of the following states:

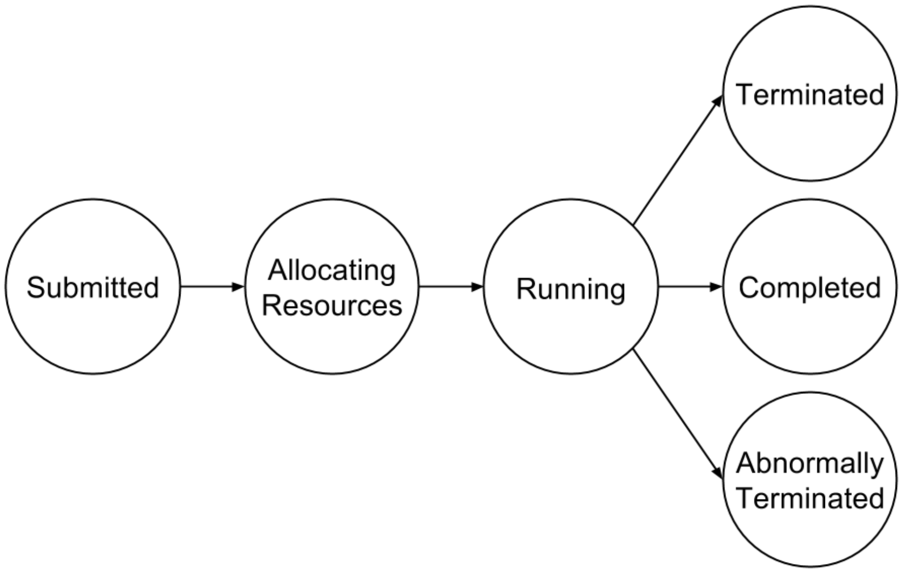


Fig. 3.8.: The life cycle of a CAM^2 submission

1. **Submitted:** When a user stars a submission, it remains in this state until CAM^2 starts allocating resources for the submission. The system starts allocating resources immediately after the submission is started.

2. **Allocating Resources:** The submission moves to the *Allocating Resources* state when the CAM^2 *Manager* module starts allocating cloud instances for the submission. Then, the CAM^2 *Worker* module, that is deployed on the cloud instances, starts executing the analysis program of the submission.

3. **Running:** When the *Worker* module starts executing the analysis program of a submission, the submission moves to the *Running* state. The submission remains in this state for the analysis duration specified in the configuration. The *Worker* retrieves the images from the selected cameras at the specified frame rate and invokes the analysis program. The website shows a progress bar to indicate the progress of a running submission as shown in Figure 3.7(f).

4. **Completed:** After the analysis program of a submission is executed for the specified duration, the running submission moves to the *Completed* state. The user can download the analysis results, and the website will provide a single compressed file that contains a directory for the results of each individual camera.

5. **Abnormally Terminated:** The uploaded module can have two types of errors: (i) errors that prevent the execution of the analysis program, such as syntax errors, API violations, or errors in the initialization stage of the analysis program. For this type of errors, CAM^2 terminates the running submission and moves it to the *Abnormally Terminated* state. (ii) runtime errors that might occur for only some frames, such as corrupted frames. For both types of errors, CAM^2 includes the stack trace of the errors in the downloaded analysis results so that the user can fix the errors.

6. **Terminated:** The user can download the intermediate analysis results of a running submission. If the user is not satisfied with the analysis results for any reason, CAM^2 allows the user to terminate the running submission. The system releases the cloud resources so that they can be used by other submissions. In this case, the submission is moved to the *Terminated* state.

3.4 CAM^2 Event-Driven API

The API of CAM^2 is event-driven, i.e. CAM^2 invokes the analysis programs when new frames arrive. This event-driven design approach significantly simplifies the analysis programs. They do not need to handle the underlying computing infrastructure and the heterogeneity of the cameras. Instead, the API of CAM^2 provides a uniform way to analyze the data from all the cameras. Users can migrate their existing analysis programs to CAM^2 easily because the API requires only slight changes to the existing programs.

The general structure of an existing analysis program can be divided into three main stages as shown in Figure 3.9(a): initialization, processing, and finalization. The first stage performs required initializations. The second stage reads, processes, and saves the results of the individual frames. The third stage releases the resources, computes and saves the overall results, etc. The operations performed by the analysis program can be categorized into: (i) IO operations that read the input frames (e.g. step 2 in Figure 3.9), or save the analysis results (e.g. steps 4 and 6 in Figure 3.9). (ii) non-IO operations that perform the actual analysis (e.g. steps 1, 3, and 5 in Figure 3.9). For most of the non-trivial analysis programs, the IO operations are usually significantly fewer than the non-IO operations.

3.4.1 The `Analyzer` Class: Event Handlers and IO APIs

The API of CAM^2 provides the `Analyzer` class as the base for any analysis program class. The goals of the `Analyzer` class are to: (i) define how the analysis program class should be. Users should implement the `initialize`, `on_new_frame`, and `finalize` event handlers as shown in Figure 3.9(b). Table 3.1 shows more details about these event handlers. (ii) provide a uniform way for any analysis program to read the input and save the results as shown in Table 3.2. The users do not need to worry about how to get the input frames from the heterogeneous cameras or how to manage the storage of the results on the cloud.

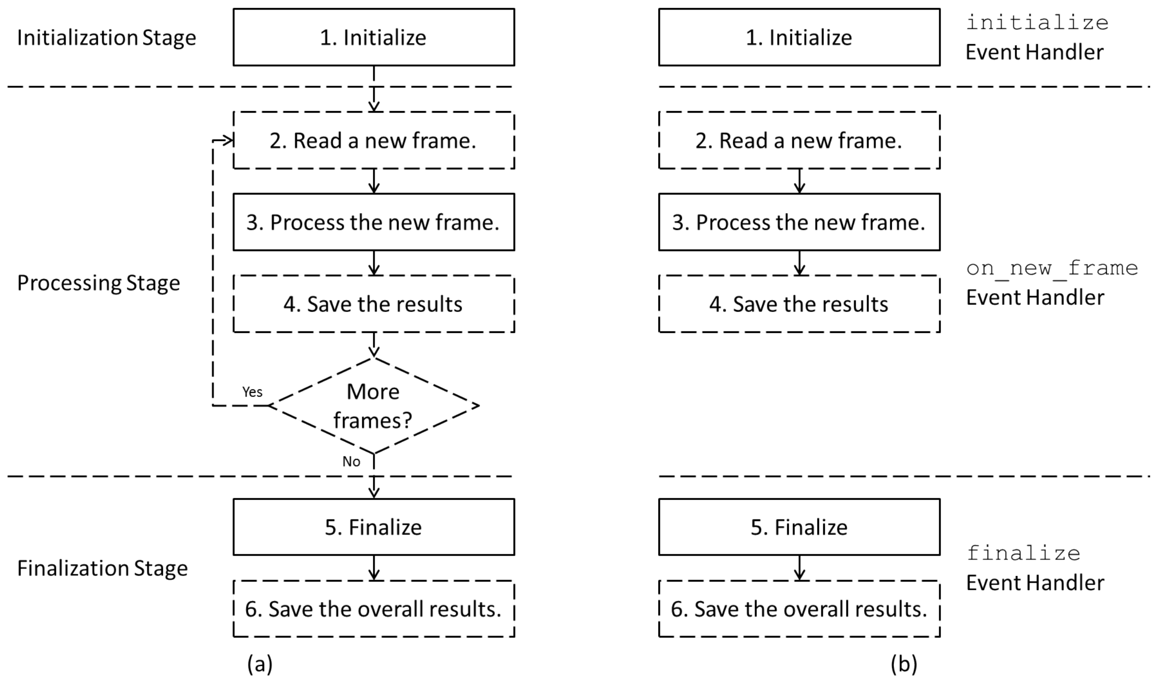


Fig. 3.9.: The general structure of: (a) an existing analysis program, and (b) the corresponding CAM^2-compatible event-driven analysis program. The dashed blocks represent the IO operations (i.e. reading the inputs and saving the results) which are modified to use the IO APIs of CAM^2. The solid blocks represent the non-IO operations which remain the same.

Event Handler	**Required?**	**Invocation**	**Usage**
`initialize`	No	At the beginning	To perform initializations.
`on_new_frame`	Yes	For each new frame	To read, analyze, and save the results of the individual frames.
`finalize`	No	At the end	To release the resources, compute and save the overall results, etc.

Table 3.1.: The event handlers provided by the `Analyzer` class. Users should implement these event handlers in order to migrate existing analysis programs to CAM^2.

In order to migrate existing analysis programs to CAM^2, users need to modify only the IO operations (e.g. steps 2, 4, and 6 in Figure 3.9). The IO operations are

Method	**Type**	**Usage**
`get_frame`	Input	To provide the most recent camera frames. If an old frame is needed, there is an optional parameter to specify the index of the frame.
`get_frame_metadata`	Input	To provide the metadata of the most recent frames. The frame metadata include the frame sequence number, the frame timestamp, and the metadata of the camera which the frame belongs to. The camera metadata include the camera ID, the latitude, and the longitude of the camera location.
`save`	Output	To save the results in a variety of formats, including text, or images.

Table 3.2.: The IO methods provided by the `Analyzer` class. These methods are used to perform IO operations.

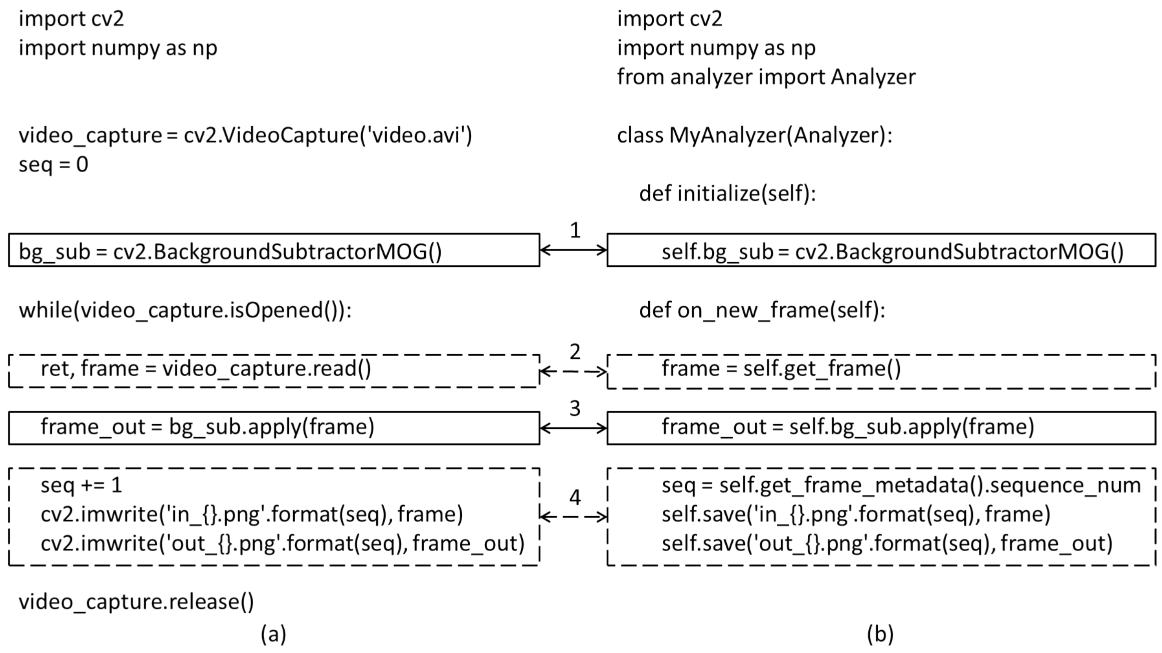
usually significantly fewer than the non-IO operations (i.e., computation steps 1, 3, and 5 in Figure 3.9). Hence, CAM^2 requires slight changes to the existing analysis programs. The following procedure should be followed to migrate an existing analysis program to CAM^2:

1. Create a class that inherits from the `Analyzer` class.
2. Move the initialization, processing, and finalization stages of the analysis program to the `initialize`, `on_new_frame`, and `finalize` event handlers respectively as shown in Figure 3.9(b). Variables that are needed by multiple stages should be defined as object attributes.
3. Modify the IO operations of the analysis program such that they use the IO APIs of CAM^2 as shown in Table 3.2. For example, use the `get_frame` method to read a new frame (step 2 in Figure 3.9), and the `save` method to save the analysis results (steps 4 and 6 in Figure 3.9).

The next subsection shows more details about how to migrate an existing analysis program to CAM^2.

3.4.2 Migration Example: Background Subtraction

Figure 3.10(a) shows an existing background subtraction analysis program that uses OpenCV's implementation of the method proposed by KaewTraKulPong and Bowden [53]. The first stage initializes a background subtractor object (step 1). The processing stage reads the input frame (step 2), subtracts the background (step 3), and saves the input frame and its foreground mask (step 4). Steps 1 and 3 are non-IO operations, while steps 2 and 4 are IO operations.

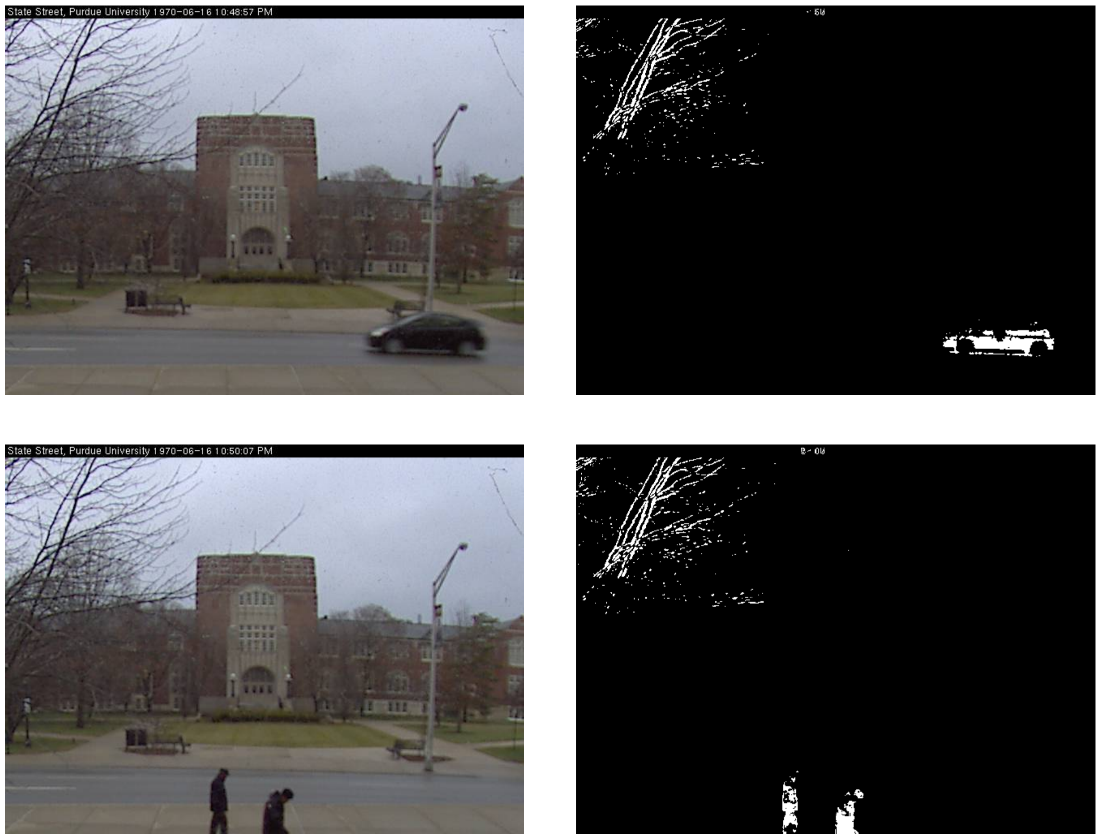


```
import cv2
import numpy as np


video_capture = cv2.VideoCapture('video.avi')
seq = 0


bg_sub = cv2.BackgroundSubtractorMOG()

while(video_capture.isOpened()):

  ret, frame = video_capture.read()

  frame_out = bg_sub.apply(frame)

  seq += 1
  cv2.imwrite('in_{}.png'.format(seq), frame)
  cv2.imwrite('out_{}.png'.format(seq), frame_out)

video_capture.release()
```

```
import cv2
import numpy as np
from analyzer import Analyzer

class MyAnalyzer(Analyzer):

  def initialize(self):

    self.bg_sub = cv2.BackgroundSubtractorMOG()

  def on_new_frame(self):

    frame = self.get_frame()

    frame_out = self.bg_sub.apply(frame)

    seq = self.get_frame_metadata().sequence_num
    self.save('in_{}.png'.format(seq), frame)
    self.save('out_{}.png'.format(seq), frame_out)
```

Fig. 3.10.: (a) Existing background subtraction analysis program. (b) The corresponding CAM^2-compatible program. The solid blocks represent the non-IO operations which remain the same. The dashed blocks represent the IO operations which are modified to use the APIs of CAM^2. For example, `self.get_frame` is used instead of `video_capture.read` to read a new frame, and `self.save` is used instead of `cv2.imwrite` to save the results. Note that steps 1-4 map to the corresponding steps in Figure 3.9.

Figure 3.10(b) shows the corresponding CAM^2-compatible program. The event-driven programming model adopted by CAM^2 makes it straightforward to map the initialization and processing stages to the corresponding event handlers. The non-IO operations (the solid blocks) remain the same, while the IO operations (the dashed blocks) are modified to use the APIs of CAM^2. Initializing and releasing the video capture object are no longer needed because CAM^2 manages the communication with the cameras. Figure 3.11 shows sample background subtraction results for a camera at Purdue University, USA.


State Street, Purdue University 1970-06-16 10:48:57 PM
State Street, Purdue University 1970-06-16 10:50:07 PM

(a) (b)

Fig. 3.11.: The background subtraction results for a camera at Purdue University, USA. (a) Two sample input frames. (b) The corresponding foreground masks. The white pixels represent the foreground (the moving objects), while the black pixels represent the background.

The website of CAM^2 provides a dozen pre-written CAM^2-compatible analysis programs for moving objects detection, sunrise/sunset detection, etc. The variety of the available CAM^2-compatible analysis programs emphasizes that the system is flexible and suitable for various applications. The migration example we presented and the examples available on the website show that CAM^2 requires only slight changes to the existing analysis programs. Only the IO operations need to be modified to use the APIs of CAM^2. If the users can organize the non-IO operations in well-defined methods (e.g. the `cv2.BackgroundSubtractorMOG` method at step 1 and the `apply` method at step 3 in Figure 3.10), these methods can be migrated without any changes.

4. CLOUD RESOURCE MANAGEMENT

A major challenge in CAM^2 is the ability to allocate and manage significant amounts of resources (CPU, memory, etc.) to analyze thousands of data streams simultaneously. It is a challenging problem to manage cloud resources in order to reduce the cost of analyzing data streams from thousands of cameras while meeting the performance requirements. This chapter proposes a resource manager that tackles this problem. This chapter is organized as follows: Section 4.1 defines the general cloud resource management problem. The rest of the chapter presents the first resource manager proposed by this dissertation. Section 4.2 explains how the manager allocates cloud instances. Section 4.3 shows how the manager monitors and scales the instances. Section 4.4 evaluates the resource manager.

4.1 Cloud Resource Management Problem

Consider the following example: a group of transportation officials study the effect of severe weather conditions on the behavior of the drivers in a particular city. They want to execute two analysis programs at the same time: (i) a vehicle tracking program on the video streams from 1000 traffic cameras at 10 Frames Per Second (FPS). This program requires high frame rate so that vehicles can be tracked across frames. (ii) a weather detection program on the image streams from 100 weather cameras at one frame per minute. This program does not require high frame rate because weather conditions do not change much across consecutive frames. The officials need to execute the analysis programs in real-time so that they can quickly respond to emergencies. The officials decide to use the cloud since they do not have enough resources for such large-scale analysis and they need to execute the analysis only occasionally during severe weather conditions. Their goal is to execute the two analysis programs at the respective desired frame rates while reducing the overall cost of using the cloud. This dissertation investigates how to achieve this goal given that the two programs have different resource requirements and desired frame rates, the cameras have different frame sizes and visual content, and the cloud instances have different types and costs.

It is a challenging problem to manage cloud resources for executing analysis programs on the real-time data streams from thousands of network cameras simultaneously. The goal is to execute the given analysis programs at the desired frame rates while reducing the overall cost of using the cloud. The problem can be divided into two parts: resource allocation and resource scaling. As shown in Figure 4.1, the resource manager makes decisions based on many factors:

1) Resource Requirements: Different analysis programs have different resource requirements. Some programs are CPU intensive, and some others are memory intensive.

2) Desired Frame Rates: The frame rate of an analysis program can have different effects on the CPU and memory requirements. Increasing the frame rate may cause the CPU requirements to increase linearly and the memory requirements to remain constant. This causes an analysis program to be CPU intensive at high frame rates, although the same program may be memory intensive at low frame rates.

3) Frame Sizes: Cameras have different frame sizes (e.g., 640×480). Analyzing streams with higher frame sizes requires more CPU and memory resources.

4) Visual Content: The visual content of a camera may vary over time. For example, a camera looking at a street may show more vehicles during the day and fewer vehicles during the night. Analyzing different content may require different amounts of resources. For example, tracking moving vehicles in a highly dynamic scene may require more resources than in a static scene. Hence, the resource requirements of analyzing the same stream may vary over time.

5) Cloud Instance Types and Costs: Cloud vendors offer dozens of instance types with different capabilities and hourly costs. For example, Table 4.1 shows

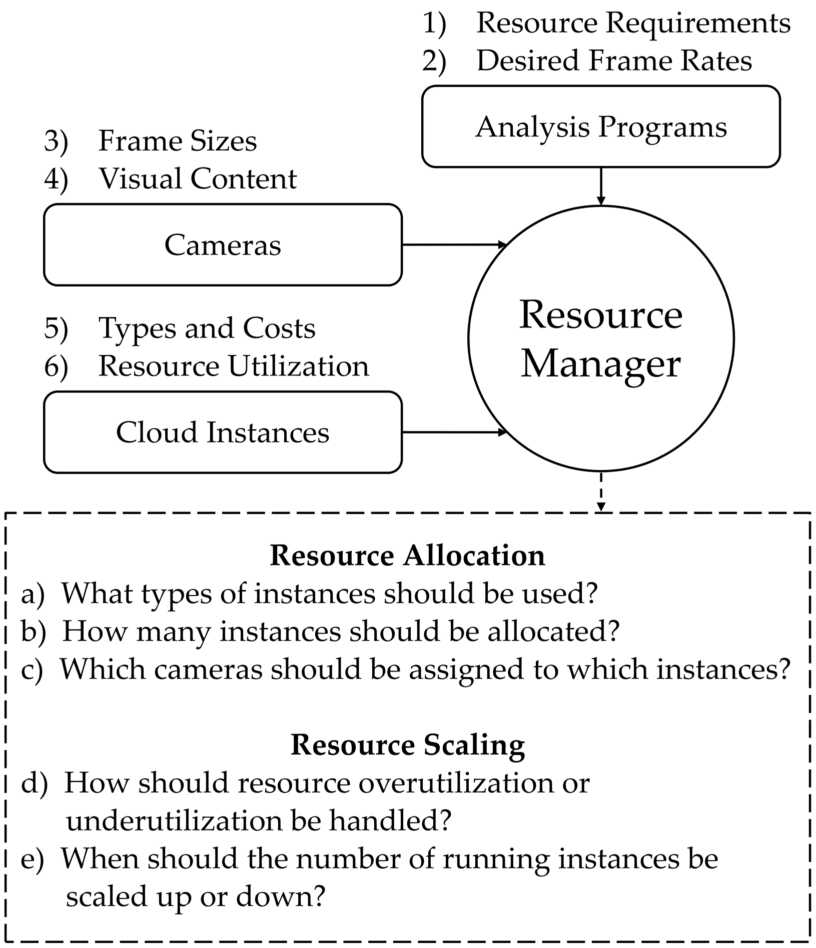

1) Resource Requirements
2) Desired Frame Rates
Analysis Programs
3) Frame Sizes
4) Visual Content
Cameras
5) Types and Costs
6) Resource Utilization
Cloud Instances
Resource Manager
Resource Allocation
a) What types of instances should be used?
b) How many instances should be allocated?
c) Which cameras should be assigned to which instances?
Resource Scaling
d) How should resource overutilization or underutilization be handled?
e) When should the number of running instances be scaled up or down?

Fig. 4.1.: The factors (1-6) affecting resource management decisions (a-e). The goal of the resource manager is to meet the performance requirements while reducing the overall cost of using the cloud.

different Amazon EC2 [3] instance types. The `m3.xlarge` and `m3.2xlarge` instances are general purpose, i.e., they provide a balance of compute and memory resources. The `c4.xlarge` and `c4.2xlarge` instances are compute optimized with the lowest cost per number of CPU cores. The `r3.xlarge` and `r3.2xlarge` instances are memory optimized with the lowest cost per GB of memory. The overall cost of using the cloud can be reduced by carefully selecting the right types of instances for the given analysis programs.

6) Resource Utilization: The currently running instances can be reused for executing new analysis programs according to the resource utilization of these instances.

Table 4.1.: The CPU, memory, and hourly price of different Amazon EC2 cloud instances.

Instance	Cores	Memory (GB)	Hourly Price
`m3.xlarge`	4	15.0	\$0.266
`m3.2xlarge`	8	30.0	\$0.532
`c4.xlarge`	4	7.5	\$0.220
`c4.2xlarge`	8	15.0	\$0.441
`r3.xlarge`	4	30.5	\$0.350
`r3.2xlarge`	8	61.0	\$0.700

Migration of data streams among instances may be necessary to maintain the resource utilization within acceptable levels (i.e., neither underutilized nor overutilized). Scaling up or down the number of running instances may also be necessary to maintain the utilization within acceptable levels.

4.2 Resource Allocation

This section presents a resource manager for executing analysis programs of the visual data from network cameras. The ultimate goal of our study is to reduce the overall cost for the scientific community to analyze large amounts of visual data using cloud. The resource manager allocates cost-effective cloud instances as presented in this section, and monitors and automatically scales the cloud resources as presented in Section 4.3.

Cloud vendors offer many cloud instance types with different capabilities in terms of numbers of cores, memory sizes, network performance, storage capacities, geographical locations, etc. With these options, the resource manager answers a number of questions that arise, e.g.

1. How much resources does one analysis program need?
2. How many data streams can one cloud instance analyze?
3. What is the most cost-effective cloud instance to use for a given analysis program?
4. How many instances are needed for executing a program that analyzes many (perhaps thousands) data streams at a desired frame rate?

We assume no prior knowledge about analysis programs so that system that can be used for a wide range of applications. Programs can be as simple as image (or video) archiving: downloading the individual images of a data stream without any analysis. Programs can be much more complex—any Python program. Due to the flexibility and hence the lack of prior knowledge about the analysis programs, we need to estimate the resource requirements of different analysis programs experimentally before determining which cloud instances are more cost-effective.

4.2.1 Models of Resource Requirements

To answer the first question "*How much resources does one analysis program need?*", we estimate the resource requirements of executing an analysis program at a given frame rate on a particular cloud instance. We monitor the resource utilization of the cloud instance while executing the analysis programs using the data from two different numbers of cameras. Consider the following settings:

- p: an analysis program
- f: a desired frame rate
- i: a type of cloud instance

The CPU utilization per camera (assuming a linear model) is denoted by $\text{CPU}^*_{i,p,f}$, and can be estimated as

$$\text{CPU}^*_{i,p,f} = \frac{\text{CPU}^m_{i,p,f} - \text{CPU}^n_{i,p,f}}{m-n}, \tag{4.1}$$

where $\text{CPU}^m_{i,p,f}$ and $\text{CPU}^n_{i,p,f}$ are the CPU utilization for analyzing the data from m and n cameras respectively. Similarly, the per camera memory utilization can be estimated as

$$\text{Mem}^*_{i,p,f} = \frac{\text{Mem}^m_{i,p,f} - \text{Mem}^n_{i,p,f}}{m-n}. \tag{4.2}$$

Equations (4.1) and (4.2) consider a constant frame rate f. The second question is "*How many data streams can one cloud instance analyze?*". To answer this question, we consider the effect of f. We define a performance metric as the ratio between the actual analysis frame rate and the desired frame rate. The analysis performance of a camera c is denoted by η^c, and can be calculated as

$$\eta^c = \frac{f^c_a}{f}, \tag{4.3}$$

where f_a^c is the actual analysis frame rate of the camera c, and f is the desired frame rate. The resource manager aims at maintaining the overall analysis performance above 90% for all data streams analyzed by one instance:

$$\eta = \frac{f_a}{f} = \frac{\frac{1}{N}\sum_{c=1}^{N} f_a^c}{f} \geq 90\%, \tag{4.4}$$

where f_a is the average actual frame rate for all the cameras, and N is the total number of cameras.

Satisfying the performance metric is tightly coupled with the resource utilization. Our experiments show that maintaining the CPU utilization under a threshold $\text{CPU}_H = 90\%$ and the memory utilization under a threshold $\text{Mem}_H = 90\%$ generally leads to meeting the performance requirements. Hence, the maximum number of streams a cloud instance of type i can analyze is estimated as

$$N_{i,p,f} = min(\frac{\text{CPU}_H}{\text{CPU}^*_{i,p,f}}, \frac{\text{Mem}_H}{\text{Mem}^*_{i,p,f}}) \tag{4.5}$$

4.2.2 Costs to Analyze Many Data Streams

The next question is "*What is the most cost-effective cloud instance to use for a given analysis program?*" We can compare the cloud instances in terms of how cost-effective they are while executing different analysis programs. We define the effective cost $\text{EC}_{i,p,f}$ of a cloud instance i as the price of analyzing one million images using a given analysis program p at a frame rate f. The effective cost can be estimated as

$$\text{EC}_{i,p,f} = \frac{c_i \times 10^6}{N_{i,p,f} \times f \times 3600}, \tag{4.6}$$

where c_i is the hourly cost of an instance type i. Hence, the most cost-effective cloud instance i^* is the one which minimizes the effective cost, and is defined as

$$i^* = \underset{i}{\text{argmin}} \frac{c_i}{N_{i,p,f} \times f}, \tag{4.7}$$

and to answer the last question, the number of needed cloud instances to analyze the data from N cameras is $\left\lceil \frac{N}{N_{i^*,p,f}} \right\rceil$, and the overall analysis cost would be $\left\lceil \frac{N}{N_{i^*,p,f}} \right\rceil c_{i^*}$.

4.2.3 Resource Allocation Procedure

The proposed resource manager uses the following procedure to allocate cost-effective cloud instances for executing a program analyzing the data from many network cameras at a specified frame rate:

Offline Stage: It aims at determining the most cost-effective cloud instance type for the given analysis. This stage is performed once, and can be used for future executions of the same analysis.

1. Execute the analysis program at the specified frame rate on cloud instances with different types using the data from two different numbers of cameras. Estimate the per camera resource utilization as shown in (4.1) and (4.2).
2. Estimate the maximum number of data streams that each cloud instance type can analyze as shown in (4.5).
3. Estimate the effective cost of each cloud instance type as shown in (4.6), and determinte the most cost-effective cloud instance type as shown in (4.7).

Online Stage - Allocation:

1. If the same user already has analysis programs running, reuse the currently running instances so that the added cost is zero. If the running instances are unable to handle the additional load, go to step 2.
2. Allocate the appropriate number of cloud instances of the most cost-effective instance type as shown in (4.7).

4.3 Monitoring and Scaling Cloud Instances

This section describes the need for continuous resource monitoring and migration of analysis programs, defines when the analysis programs are migrated and a set of migration policies, and presents a resource manager that monitors and scales the cloud resources in order to reduce the overall analysis cost while taking into consideration the quality of the analysis results.

The resource requirements of an analysis program may change due to many factors, for example,

- The frame rates from a network camera may change over time due to network conditions and concurrent access from multiple users.
- The content of the data may affect the execution time and the amount of memory running an analysis programs. For example, detecting the moving objects in a highly dynamic scene would consume more resources than a static scene.

This urges the need for continuous monitoring of the resource utilization of the cloud instances and automatic scaling of the cloud resources (allocating more instances when the analysis programs needs more, and deallocating some instances when the analysis programs needs fewer.) Migration of analysis programs between cloud instances is essential in this process, but it negatively affects the quality of the analysis results for many reasons: (i) When migration is performed, the analysis programs are interrupted and there will be a time gap in the analysis results. (ii) If the analysis programs maintain temporal information such as background models, this information will be lost and have to be rebuilt on the new cloud instances. This will negatively affect the quality of the results after migration.

The proposed resource manager migrates analysis programs from a cloud instance i when its resources are overutilized, i.e. when

$$\text{CPU}_i > \text{CPU}_H \quad \text{or} \quad \text{Mem}_i > \text{Mem}_H, \tag{4.8}$$

where CPU_i and Mem_i are the current CPU and memory utilization of the cloud instance i, and CPU_H and Mem_H are the high thresholds that set an upper bound on the permissiable CPU and memory utilization. In addition, the resource manager considers deallocating a cloud instance when its resources are underutilized, i.e. when

$$\mathrm{CPU}_i < \mathrm{CPU}_L \quad \text{and} \quad \mathrm{Mem}_i < \mathrm{Mem}_L, \tag{4.9}$$

where CPU_L and Mem_L are the low thresholds that set a lower bound on the acceptable CPU and memory utilization.

The following set of migration policies defines which analysis programs the resource manager should migrate from an overutilized cloud instance:

1. Migrate image analysis programs first before migrating video analysis programs because image analysis programs do not keep temporal information across frames.
2. Migrate analysis programs with lower frame rate to reduce disruption.
3. Migrate analysis programs that require more resources so that fewer data streams are needed for migration.
4. Migrate analysis programs that started more recently to prevent disruption of long-running programs.

As shown in Figure 4.2, the proposed resource manager uses the following procedure to monitor and scale cloud resources in order to perform image and video analysis of the big data from network cameras:

1. When a user starts a new analysis program, use the allocation procedure in Section 5.3.3.C to estimate and allocate the appropriate number of cloud instances.
2. Continuously monitor the resource utilization of all the cloud instances.
3. If the resources of a cloud instance are overutilized as defined in (4.8), immediately migrate *some* data streams from the instance. Choose the cameras to

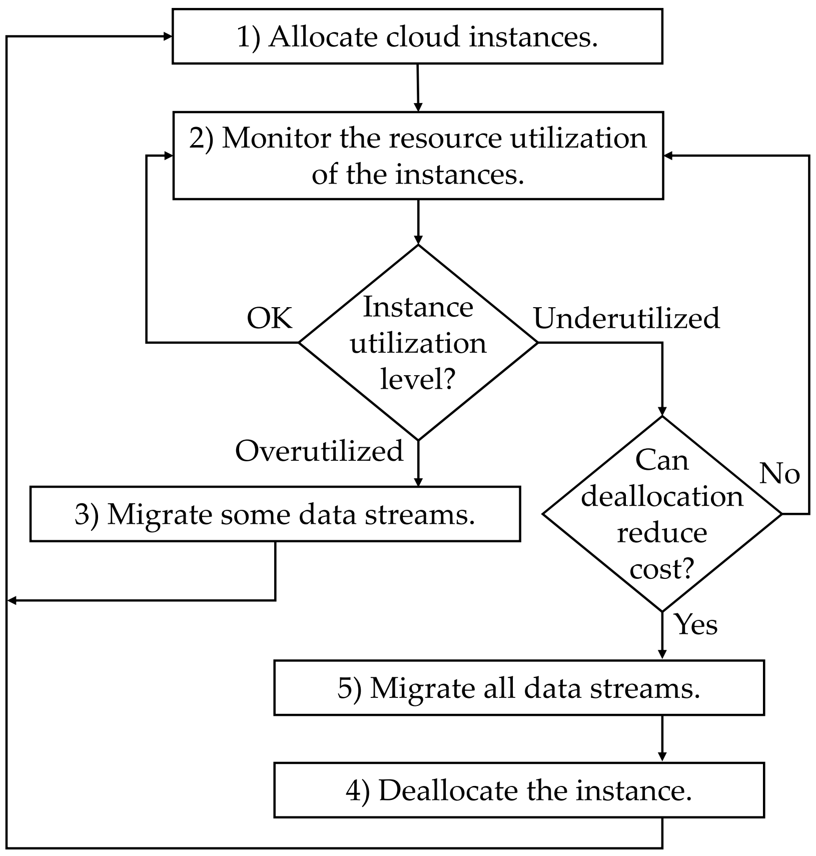

1) Allocate cloud instances.
2) Monitor the resource utilization of the instances.
Instance utilization level?
OK
Underutilized
Overutilized
3) Migrate some data streams.
Can deallocation reduce cost?
No
Yes
5) Migrate all data streams.
4) Deallocate the instance.

Fig. 4.2.: The overall procedure of monitoring and scaling cloud instances.

migrate based on the abovementioned migration policies. Suspend the analysis of the chosen data streams, and use the allocation procedure to allocate new resources.

4. If the resources of a cloud instance are underutilized as defined in (4.9) for a period of time, the instance is a candidate to be deallocated. Use the allocation procedure to estimate the hourly price of the proposed cloud instances if all the analysis programs are migrated from the underutilized instance. Deallocate the instance and migrate its analysis programs if this price is less than the hourly price of the instance.

There is a tradeoff between the analysis cost and the quality of the analysis results. To maintain the quality of the analysis results, the proposed resource manager may incur higher cost. For example, the resource manager considers deallocating only the underutilized cloud instances.

4.4 Experiments

In order to evaluate the proposed resource manager, we conduct experiments using six types of cloud instances and four analysis programs. The cloud instances have different CPU and memory capabilities, and the analysis programs represent different workloads in terms of CPU and memory: image archival, motion estimation, moving objects detection, and human detection.

4.4.1 Experimental Setup

Table 4.1 compares the six Amazon EC2 cloud instance types that are used in the experiments: two general purpose instances (`m3.xlarge` and `m3.2xlarge`), two compute optimized instances (`c4.xlarge` and `c4.2xlarge`), and two memory optimized instances (`r3.xlarge` and `r3.2xlarge`). The processor of the compute optimized instances is Intel Xeon E5-2666 v3 clocked at 2.9 GHz, and it is Intel Xeon E5-2670 v2 clocked at 2.5 GHz for all the other instances.

Table 4.2 shows the analysis programs used in the experiments. All the programs are implemented using OpenCV [58]. The programs represent different workloads in terms of CPU and memory requirements as shown later by the experiments. IA, ME, and MOD are used in the experiments for both image analysis at 0.2 FPS (Frames Per Second) and video analysis at 10 FPS. HD is used for image analysis only because it is very compute intensive and can not be executed at high frame rate.

4.4.2 Resource Requirements and Effective Cost

To estimate the number of streams an instance can analyze as well as the resource requirements of an analysis program, we conduct experiments executing the four

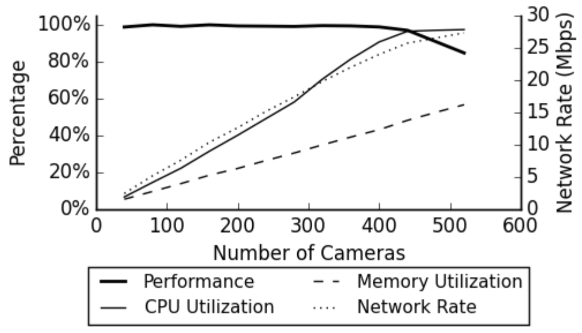
Table 4.2.: The analysis programs used in the experiments.

Name	Abbr.	Description	Results Per Input Image
Image Archival	IA	Downloads images, without further analysis.	The input image
Motion Estimation	ME	Performs background subtraction [53] and estimates the amount of motion as the percentage of foreground pixels.	The foreground mask, the input image, and the amount of motion.
Moving Objects Detection	MOD	Performs background subtraction [54], removes the noise using morphological erosion and dilation, and finds the contours of the foreground mask. Each contour is a moving object.	The output image with the moving objects, the input image, and the number of moving objects.
Human Detection	HD	Detects humans using Histograms of Oriented Gradients [57].	The output image with the detected humans, the input image, and the number of humans.

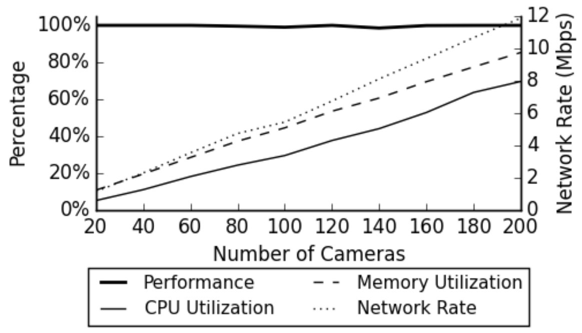
analysis programs in Section 6.3.1 on the six cloud instances in Table 4.1. The experiments monitor the resource utilization as well as the analysis performance as defined in (4.4).

Figure 4.3 shows the resource utilization and the analysis performance of executing different image and video analysis programs using different cloud instances. The figure shows that while increasing the number of cameras increases the resource utilization, the analysis performance can gradually decrease after the CPU resources are used up or suddenly drops after the memory resources are used up. The experiments show that CPU and memory resources are used up faster than network resources, and the CPU and memory utilization should be maintained below 90% in order to satisfy the performance metrics as defined in Section 5.3.3.

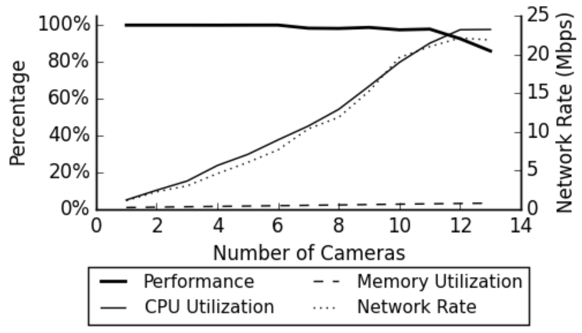
Many factors determine whether the CPU or the memory resources will be the barrier to increase the number of data streams being analyzed: (i) The CPU and memory capabilities of the cloud instances. For example, the same analysis program for moving objects counting uses up the CPU resources faster on the `m3.2xlarge`



(a)



(b)



(c)

Fig. 4.3.: The resource utilization and the analysis performance, as defined in (4.4): (a) Moving objects counting at 0.2 FPS using `m3.2xlarge`. (b) Moving objects counting at 0.2 FPS using `c4.xlarge`. (c) Motion estimation at 10 FPS using an `r3.xlarge` cloud instance. The left vertical axis represents the percentage of the CPU utilization, the memory utilization, and the analysis performance.

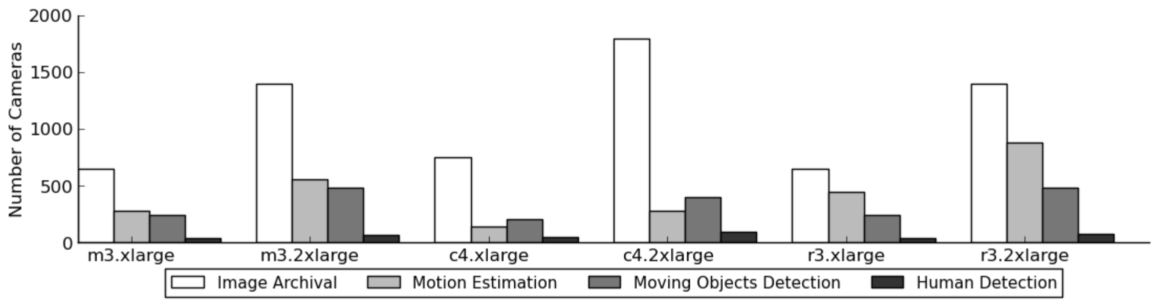
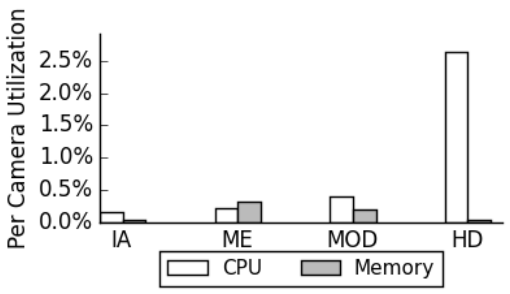


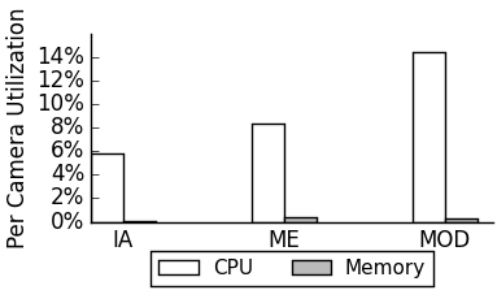
Fig. 4.4.: The maximum numbers of data streams that can be analyzed at 0.2 FPS using different analysis programs and different cloud instances.

cloud instance (30GB memory) as shown in Figure 4.3(a), but uses up the memory resources faster on the `c4.xlarge` cloud instance (7.5GB memory) as shown in Figure 4.3(b). (ii) The resource requirements of the analysis programs. Compute intensive analysis programs such as human detection use up CPU resources faster, and memory intensive analysis programs such as motion estimation use up memory resources faster. (iii) The analysis frame rate. The higher the analysis frame rate is, the higher the CPU requirements of the analysis. Figure 4.3(c) shows that executing the analysis program for motion estimation at 10 FPS uses up the CPU resources much faster, and the memory resources are highly underutilized.

These experiments enable us to estimate the maximum numbers of data streams that can be analyzed using different analysis programs on different cloud instances. Figure 4.4 shows the maximum number of data streams for image analysis at 0.2 FPS, and a similar figure can be shown for video analysis. The experiments also enable us to estimate the per camera CPU and memory utilization as defined in (4.1) and (4.2). Figure 4.5 shows the per camera CPU and memory utilization for different analysis programs using the `m3.xlarge` cloud instance. For image analysis at 0.2 FPS, human detection is the most compute intensive analysis program, and motion estimation is the most memory intensive analysis program. Image archival is the least compute and memory intensive. For video analysis at 10 FPS, CPU resources becomes much more vital than memory resources.

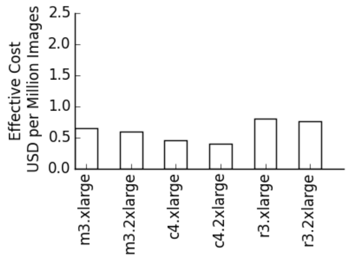


(a)

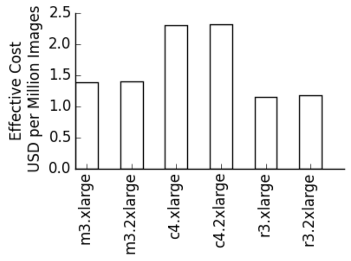


(b)

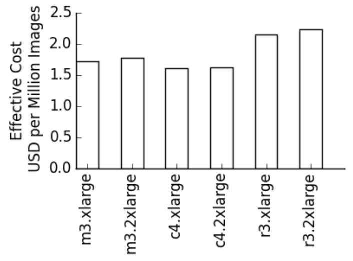
Fig. 4.5.: Per camera CPU and memory utilization for different analysis programs using the `m3.xlarge` cloud instance in the cases of: (i) Image analysis at 0.2 FPS. (ii) Video analysis at 10 FPS.



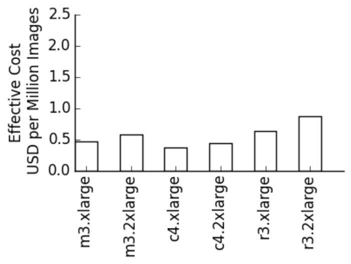
(a)



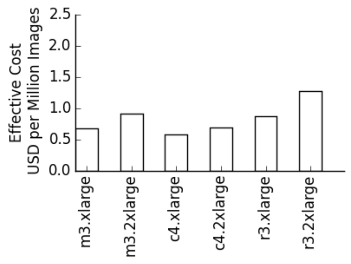
(b)



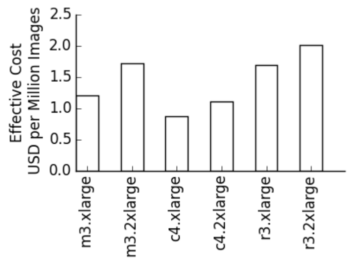
(c)



(d)



(e)



(f)

Fig. 4.6.: The effective cost as defined in (4.6) of different cloud instances for executing different analysis programs: (a-c) at 0.2FPS. (d-f) at 10 FPS. (a, d) Image archival. (b, e) Motion estimation. (c, f) Moving objects counting.

Figure 4.6 and Figure 4.7 show the effective cost of different cloud instances for executing different analysis programs. The figures show the following:

1. *There is no clear winner.* Different cloud instances are more cost-effective than others for some analysis programs. Choosing the right cloud instance for an

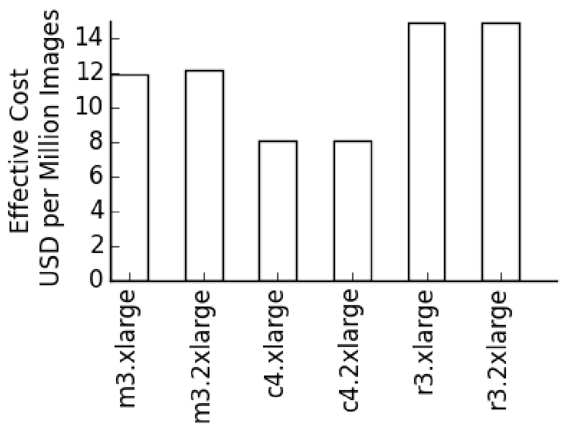


Fig. 4.7.: The effective cost as defined in (4.6) for executing the human detection program at 0.2 FPS.

analysis program can save half on the analysis cost. This observation motivates our research for cost-based resource allocation and management.

2. For image analysis at 0.2 FPS, compute optimized cloud instances (`c4.xlarge` and `c4.2xlarge`) are more cost-effective for moving objects detection and human detection. Memory optimized cloud instances (`r3.xlarge` and `r3.2xlarge`) are more cost-effective for motion estimation.

3. For video analysis at 10 FPS, compute optimized cloud instances are always more cost-effective than the other instances. That's because video analysis consumes CPU resources much more than memory resources as we showed earlier.

4. Although the `xlarge` instances provide half the CPU and memory resources of the `2xlarge` instances for half the price as shown in Table 4.1, the `xlarge` instances are often more cost-effective than the `2xlarge` instances. This recommends using smaller instances instead of larger ones.

Program	Start Time	Duration (Hours)	Cameras	Frame Rate
ME	0:00	4.50	1000	0.2
HD	1:15	4.75	10	0.2
MOD	1:30	4.50	16	10.0

Table 4.3.: The analysis programs of the 6-hour large-scale experiment.

4.4.3 Cloud Resource Allocation and Management

To evaluate the proposed resource manager, we conduct a 6-hour large-scale experiment that uses CAM^2 to analyze the data from 1026 cameras using different analysis programs at different frame rates as shown in Table 4.3. The experiment analyzes 5.5 million images, totalling 260GB data. Figure 4.8 shows sample analysis results.

In this experiment, the resource manager considers a cloud instance overutilized if the utilization is above 90% and underutilized if the utilization is below 40%, and targets a 70% utilization when allocating new resources. A cloud instance has to remain underutilized for 5 minutes before an action is taken by the resource manager. Figure 4.9 shows the CPU utilization of the cloud instances during experiment. The figure does not show the memory utilization because it does not affect any resource management decisions in this experiment. The figure shows the following events:

1. At 0:00, four memory-optimized `r3.xlarge` instances are allocated to handle the memory-intensive ME analysis program.

2. At 0:05, one `r3.xlarge` instance is deallocated after migrating its analysis programs to the other three running instances as shown by the marker A.

3. At 1:15, two of the currently running instances can handle the additional load of the second analysis program as shown by the marker B; as a result, the added analysis cost is zero.

4. At 1:30, four compute-optimized `c4.xlarge` instances are allocated to handle the CPU-intensive MOD program.

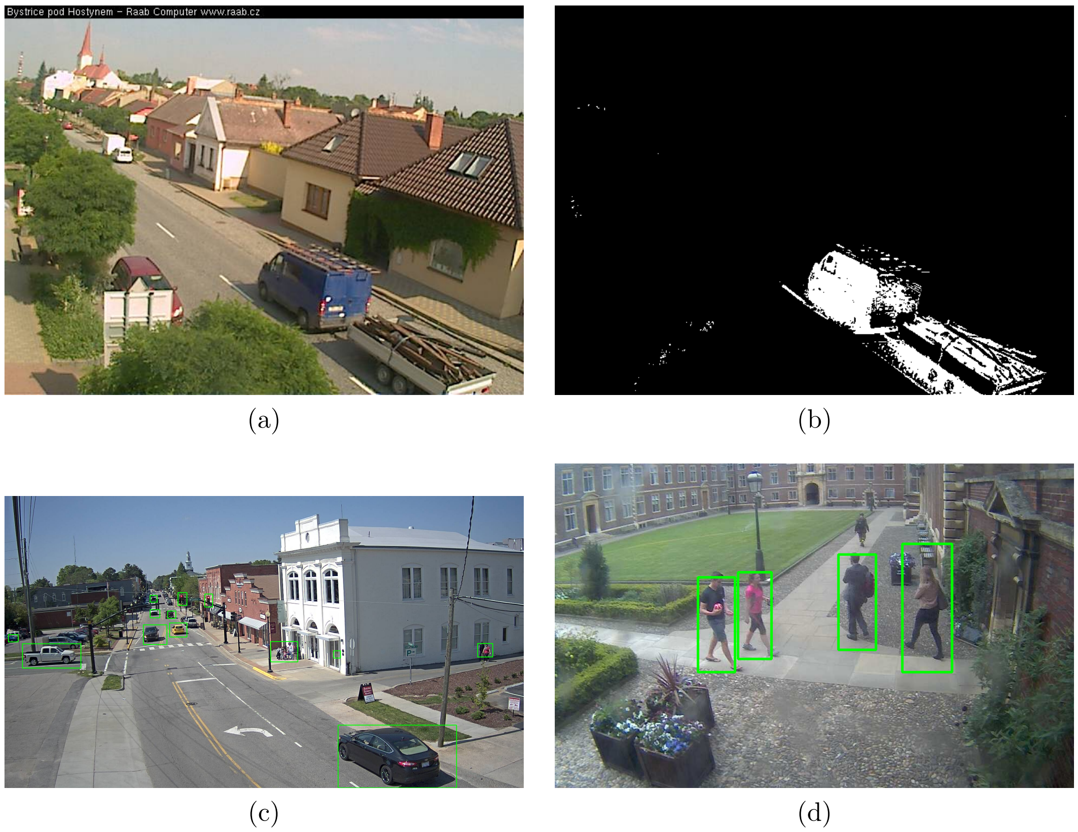


Fig. 4.8.: Sample results of the experiment shown in Table 4.3: (a, b) Motion estimation for a camera in Czech Republic. (a) A sample input image. (b) The corresponding foreground mask. The amount of motion in this image is 5%. (c) Moving objects detection for a camera in the USA. The moving objects are enclosed by green boxes. The image shows eight moving cars, two groups of moving pedestrians, and two traffic lights considered moving due to changing from yellow to red. (d) Human detection for a camera in England. Humans are enclosed by green boxes. The program successfully detects four humans in the image, and misses one.

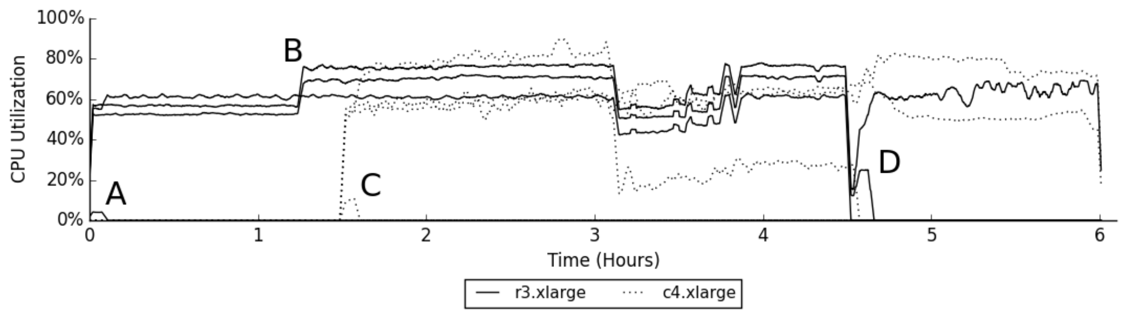


Fig. 4.9.: The CPU utilization of the cloud instances while analyzing the data from 1026 cameras using different analysis programs at different frame rates as shown in Table 4.3.

5. At 1:35, one `c4.xlarge` instance is deallocated after migrating its analysis programs to the other three running `c4.xlarge` instances as shown by the marker C.

6. At 3:10, the CPU utilization of some cloud instances drops, which can be due to unexpected network conditions.

7. At 4:30, the execution of the first analysis program ends, which causes one `r3.xlarge` to be deallocated and the other two `r3.xlarge` instances are underutilized.

8. At 4:35, two of the three underutilized instances are deallocated after migrating their analysis programs to the third instance as shown by the marker D.

9. At 6:00, the execution of all the analysis programs ends.

Based on the lifetime of the cloud instances in Figure 4.9 and their prices, the experiment costs \$12.77. If the proposed resource manager is not used, and the general-purpose `m3.xlarge` instances are used for all the analysis programs, this experiment needs five, one, and five `m3.xlarge` instances to handle the three analysis programs respectively. The overall analysis cost is \$14.63 in this case. This means that the resource manager leads to a 13% reduction in the overall analysis cost.

5. ENHANCED RESOURCE MANAGEMENT

This chapter proposes an enhanced resource manager that improves the first manager (Chapter 4) in many ways. For example, the enhanced manager: (i) is able to handle multiple analysis programs at different frame rates, (ii) considers the desired frame rates and the camera frame sizes while estimating the resource requirements of analyzing the data stream from each camera, (iii) models the resource allocation problem as a 2D vector bin packing problem [4] and solves it using a greedy heuristic algorithm, (iv) avoids conducting test runs on all instances and for different frame rates by modeling the relationship between the frame rate and the resource requirements.

This chapter is organized as follows: Section 6.3.4 shows how the resource manager estimates the resource requirements of analyzing the data stream from each camera. This stage is performed once and used for future executions of the same analysis program. Section 5.2 formulates the resource allocation problem as a vector bin packing problem. Section 5.3 evaluates the resource manager.

5.1 Estimation of Resource Requirements

In order to support a wide range of analysis programs, the resource manager assumes no prior knowledge about the programs. Hence, the resource requirements of the programs are unknown a priori, but can be estimated by conducting a test run. The manager executes each program at the desired frame rate on the data streams from multiple cameras and monitors the CPU and memory utilization during the analysis. This allows the manager to estimate the resource requirements of analyzing a single data stream. The test run is conducted once and used for future executions of the same analysis program. In order to eliminate the need to conduct a test run if users execute the same analysis program at different frame rates, the manager models the relation between the frame rate and the CPU (or memory) requirements as a linear (or constant) relationship. The experiments demonstrate these relationships later.

Camera frame sizes significantly affect resource requirements. Analyzing streams with higher frame sizes requires more resources. The experiments show that the relation between camera frame sizes and resource requirements is almost linear. Hence, the test run is conducted using cameras with the same frame size. The requirements of the cameras with other frame sizes are estimated linearly based on their frame size.

Cloud instance types have different capabilities and hourly costs. The hourly cost of an instance type is proportional to its capabilities. For example, an instance type with 8 cores and 30 GB of memory is twice as expensive as an instance type with 4 cores and 15 GB of memory. Our relevant work [38] [59] [60] shows that smaller instances (i.e., fewer CPU cores and less memory) are more cost-effective than larger ones. Based on these observations, the manager prefers smaller instance types.

The manager uses a single instance to estimate the resource requirements of analyzing the data stream from each camera according to the following procedure:

1. Select a set of cameras with the same frame size to conduct the test run.
2. Execute each given analysis program at the desired frame rate on the data streams from multiple cameras.
3. Monitor the CPU and memory utilization.
4. Estimate the resource requirements of analyzing the data stream from a single camera.
5. For cameras with other frame sizes, estimate the resource requirements linearly based on their frame sizes.

5.2 Resource Allocation Using Vector Bin Packing

We model the resource allocation problem as a 2D vector bin packing problem. In the 2D vector bin packing problem [4], there are objects and bins. Each object has a 2D resource demand. Each bin has a 2D resource constraint and a fixed cost. The problem is to select bins and pack all the objects into these bins such that the overall cost of all the bins is minimized without violating the resource constraints.

Similarly, in the resource allocation problem, there are data streams and instances. Different CPU and memory requirements are needed to analyze each data stream. Each instance has CPU and memory constraints and a fixed hourly cost. The problem is to select instances and assign all the data streams to these instances such that the overall cost of all the instances is minimized without violating the resource constraints.

We use the First Fit by Ordered Deviation (FFOD) heuristic algorithm proposed by Han et al. [4] to solve the 2D vector bin packing problem. The algorithm uses opportunity costs to select new bins and assign objects to bins. The output is the number of required bins, the type of each bin, and the objects assigned to each bin. This maps to the number of required instances, the type of each instance, and the cameras assigned to each instance.

The resource manager allocates cloud instances using the following procedure:

1. Map the resource allocation problem to a 2D vector bin packing problem.

2. Use the heuristic algorithm proposed by Han et al . [4] to solve the 2D vector bin packing problem in order to get the number of required instances, the type of each instance, and the cameras assigned to each instance.

3. Allocate the required instances of the given types if the running instances are not sufficient.

5.3 Experiments

This section describes the experiments used to evaluate the proposed resource manager. Section 6.3.1 explains the setup used in the experiments. Section 5.3.2 evaluates different factors considered by the manager while allocating resources. Section 5.3.3 evaluates the ability of the manager to reduce the cost of the allocated instances while meeting the performance requirements.

5.3.1 Experimental Setup

Table 5.1 shows the different Amazon EC2 [3] instance types used in the experiments. The `m4.xlarge` and `m4.2xlarge` instances are general purpose, i.e., they provide a balance of compute and memory resources. The `c4.xlarge` and `c4.2xlarge` instances are compute optimized with the lowest cost per number of CPU cores. The `r3.xlarge` and `r3.2xlarge` instances are memory optimized with the lowest cost per GB of memory. The experiments do not include disk and network resources since we observe that they are not the bottleneck while analyzing the data from many cameras.

Table 5.1.: The CPU, memory, and hourly cost of the Amazon EC2 instance types (at Oregon) used in the experiments.

Instance	Cores	Memory (GB)	Hourly Cost
`m4.xlarge`	4	16.0	\$0.239
`m4.2xlarge`	8	32.0	\$0.479
`c4.xlarge`	4	7.5	\$0.209
`c4.2xlarge`	8	15.0	\$0.419
`r3.xlarge`	4	30.5	\$0.333
`r3.2xlarge`	8	61.0	\$0.665

In addition to the four analysis programs in Table 4.2, the experiments use a fifth program for Feature Tracking (FT). FT detects [55] and tracks [56] image features with back-tracking for verification, calculates the speed of each feature, and visualizes the tracks of the features according to their speeds. The outputs of the program are: the image with the tracks of the moving features, the input image, the number of features, the number of moving features, and the average speed of the moving features.

5.3.2 Evaluation of the Factors Affecting Resource Management Decisions

Many factors affect resource management decisions as shown in Figure 4.1. This section evaluates the effect of each factor separately.

Resource Requirements of Analysis Programs

Figure 5.1 shows that different analysis programs have different CPU and memory requirements. Some programs are CPU intensive (e.g., HD), and some are memory intensive (e.g., ME). The per camera utilization of each program is estimated according to the resource utilization while analyzing the data streams from multiple cameras. The experiment executes the analysis programs at 0.2 FPS for 5 minutes, and the frame size of all the cameras is 640×480. This experiment demonstrates the need to conduct a test run for any analysis program in order to estimate its resource requirements.

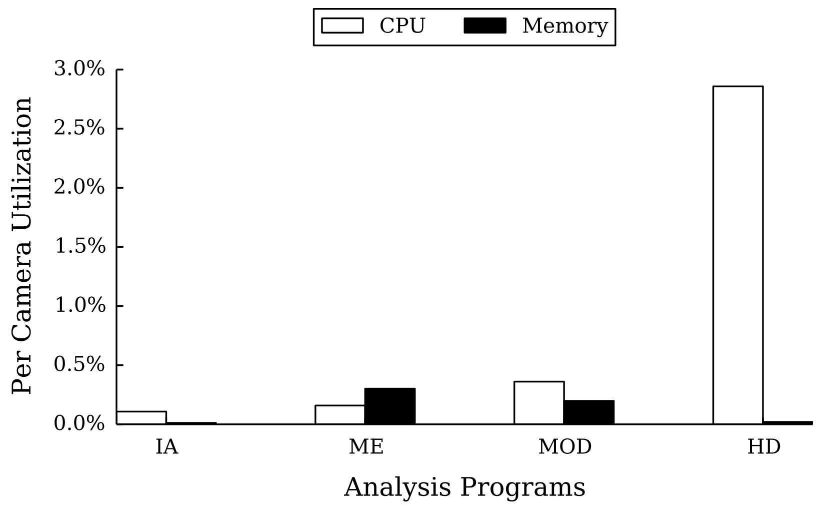


Fig. 5.1.: Per camera CPU and memory utilization for different analysis programs at 0.2 FPS using the `m4.xlarge` instance.

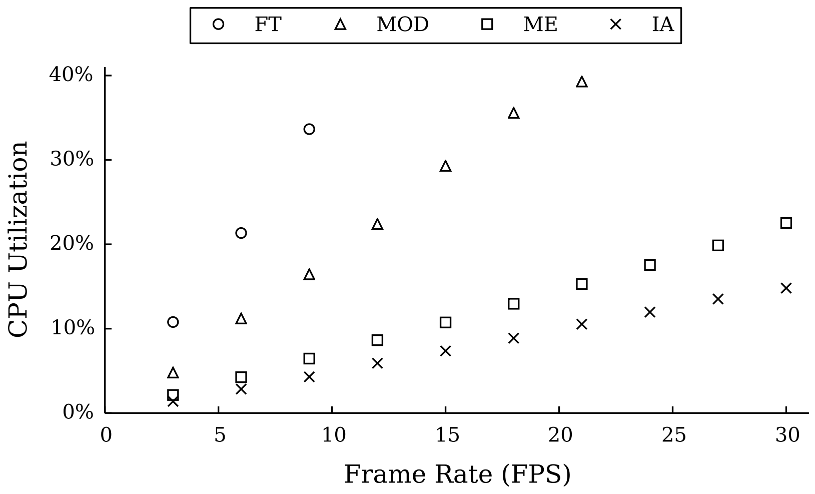
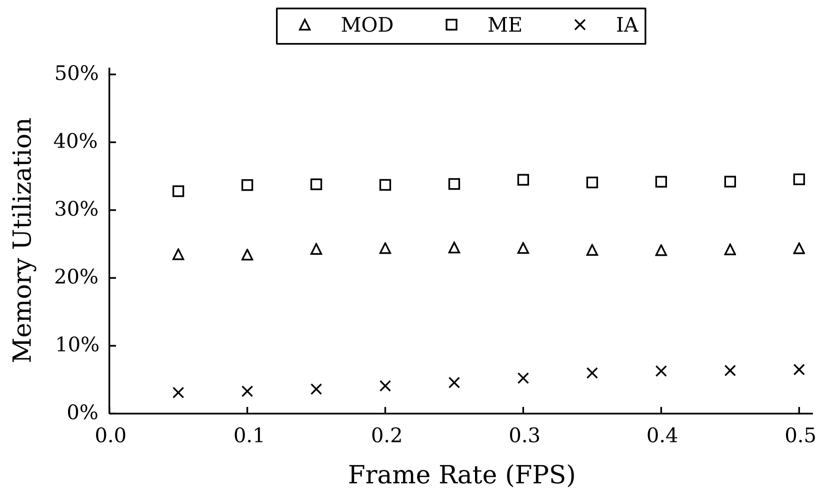


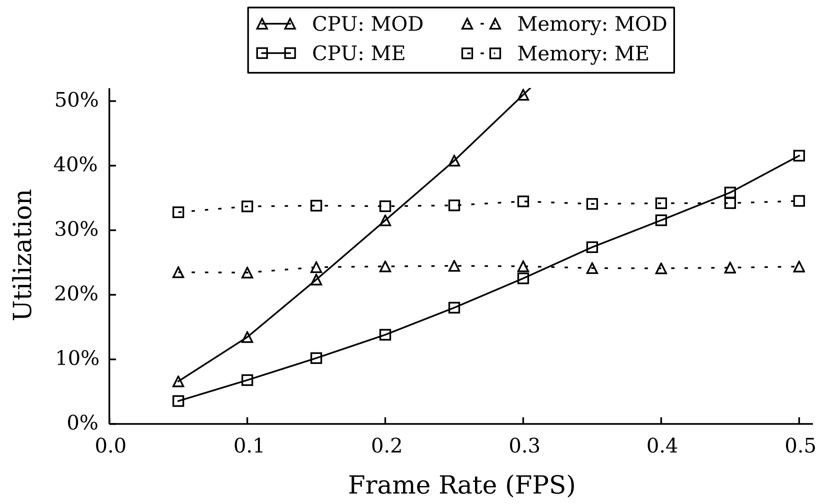
Fig. 5.2.: The effect of the desired frame rate on the CPU utilization of an `m4.xlarge` instance while analyzing an MJPEG stream from a single camera.

Desired Frame Rates Figure 5.2 shows the relationship between the frame rate and the CPU utilization. The experiment analyzes an MJPEG stream from a single camera using different analysis programs and frame rates. The CPU utilization is averaged over 5 minutes, and the frame size of the camera is 704×480. FT and MOD do not meet the performance requirement ($\eta > 90\%$) at frame rates above 9 FPS and 21 FPS respectively, due to their relatively high execution times. This experiment shows that the CPU utilization increases linearly as the frame rate increases.

Figure 5.3(a) shows that the effect of the frame rate on the memory utilization is negligible. The experiment analyzes JPEG streams from 100 cameras using different analysis programs and frame rates. The memory utilization is averaged over 5 minutes, and the frame size of all the cameras is 640×480. These experiments show the frame rate significantly affects the CPU utilization but has negligible effect on the memory utilization. Hence, analysis programs may be CPU intensive at high frame rates and memory intensive at low frame rates as shown in Figure 5.3(b). For example, MOD (and ME) are memory intensive at 0.15 FPS (and 0.4 FPS) and CPU intensive at 0.2 FPS (and 0.45 FPS).



(a) The effect of the frame rate on the memory utilization is negligible.



(b) Analysis programs may become CPU intensive or memory intensive based on the frame rate.

Fig. 5.3.: The effect of the desired frame rate on the CPU and memory utilization of an `m4.xlarge` instance while analyzing image streams from 100 cameras.

These experiments demonstrate the need to consider the effect of the frame rate on *both* the CPU and memory requirements of analysis programs. The proposed resource manager: (i) models the relation between the frame rate and the CPU requirement as a linear relationship, (ii) considers the memory requirement of an analysis program constant for any frame rate, and (iii) models the resource allocation problem as a 2D bin packing problem to handle *both* the CPU and memory requirements.

Camera Frame Sizes Figure 5.4 shows the linear relationship between the camera frame size and the CPU utilization. The experiment analyzes an MJPEG stream from a single camera that supports multiple frame sizes. The experiment executes different analysis programs at 10 FPS. The CPU utilization is averaged over 5 minutes, and the frame sizes of the camera are 320×240, 640×480, and 800×600. Based on this experiment, the resource manager models the relation between the camera frame size and the CPU requirement as a linear relationship. Similarly, we also observe a linear relationship between the camera frame size and the memory requirement.

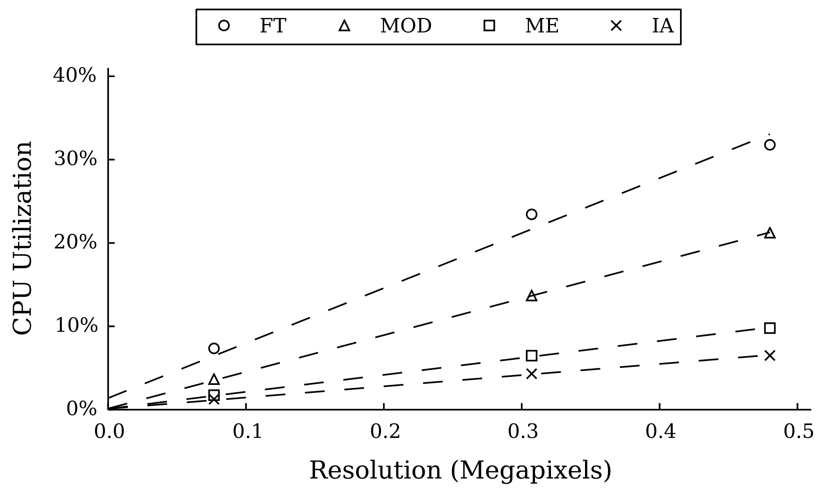
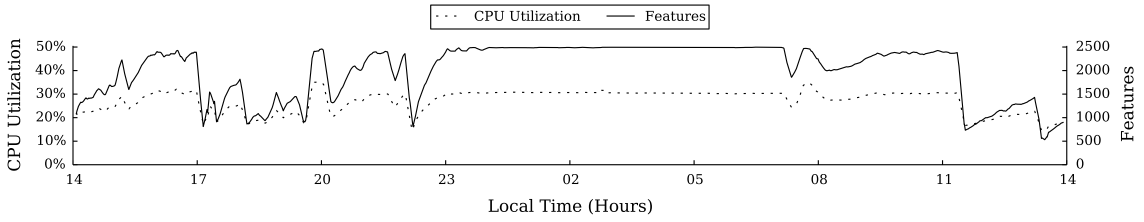
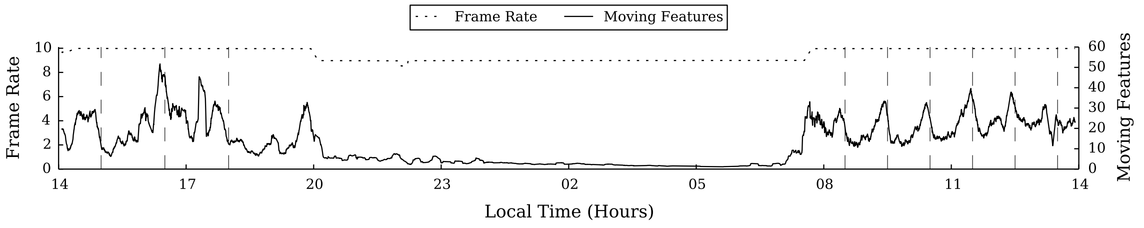


Fig. 5.4.: The effect of the camera frame size on the CPU utilization of an `m4.xlarge` instance while executing different analysis programs at 10 FPS. Dashed lines show the linear approximation of the measures.

Camera Visual Content To evaluate the effect of the visual content on the resource requirements of analysis programs, we perform an experiment that uses FT to detect and track features in an MJPEG stream for 24 hours at 10 FPS. The experiment analyzes 820,000 images. That is more than 25 GB of data from a single 640×480 camera in 24 hours. This experiment uses an `m4.xlarge` instance and monitors the CPU utilization of the instance as well as the number of features and moving features in every image.



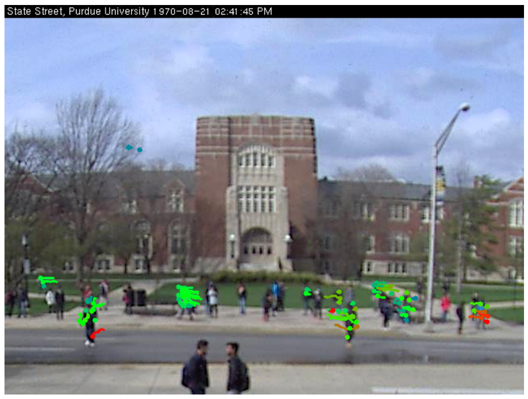
(a) The CPU utilization changes dynamically with the visual content, represented by the total number of tracked features.



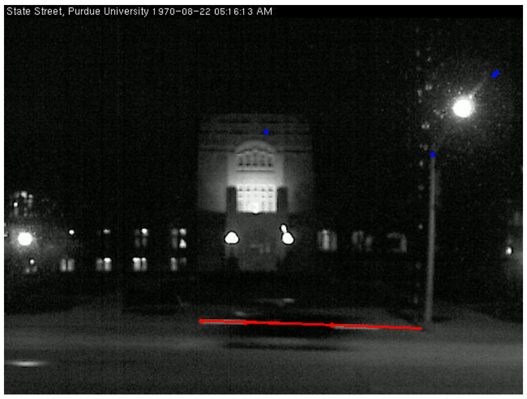
(b) More moving features are detected during the day, especially before the lectures starting times indicated by the vertical dashed lines.

Fig. 5.5.: Using an `m4.xlarge` instance to execute FT on an MJPEG stream from a camera at Purdue University for 24 hours at 10 FPS (820,000 images, 25 GB of data). The lines in (a) and (b) are smoothed using a moving average filter with a 10-minute window.

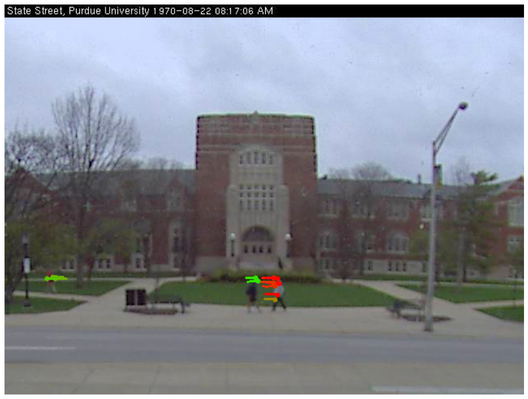
Figure 5.5 shows the results of the experiment. Figure 5.5(a) shows that the CPU requirements of FT change dynamically with the visual content, represented by the total number of tracked features. When the number of features increases (or decreases), FT requires more (or less) CPU resources and the CPU utilization increases (or decreases). During the night, many features are detected due to the noise, but FT limits the number of tracked features to 2,500. Figure 5.5(b) shows that the system is able to maintain the actual frame rate close to the desired frame rate (10 FPS). The overall actual frame rate is 9.5 FPS ($\eta > 90\%$). The figure also shows how the number of moving features changes over time. Figures 5.6 show sample output results. Based on the dynamic resource requirements shown in this experiment, the resource manager monitors the allocated instances, allocates more instances if needed, and deallocates existing instances to reduce the cost if possible.



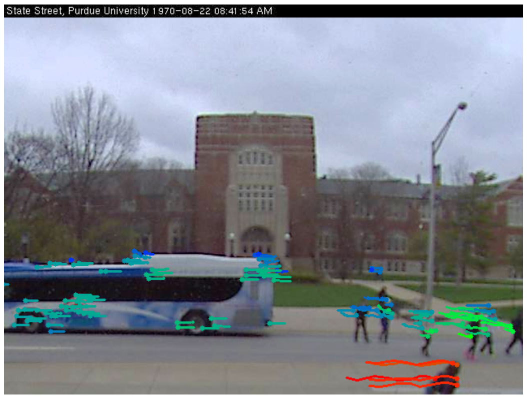
(a) At 16:25 when it is crowded



(b) At 7:00 with a moving car



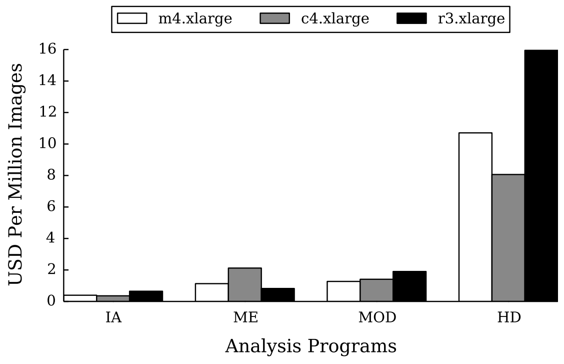
(c) At 10:00 during lectures



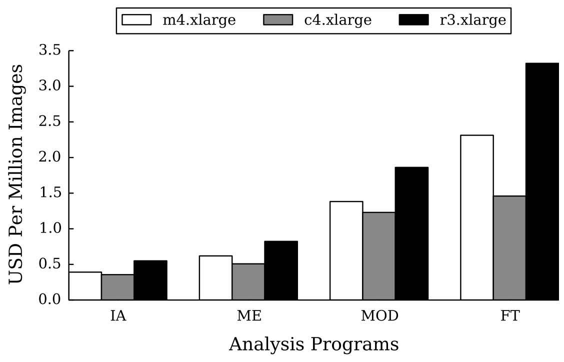
(d) At 10:25 during a lecture break

Fig. 5.6.: Sample output results of executing FT on an MJPEG stream from a camera at Purdue University for 24 hours at 10 FPS. A circle is a moving feature, and a line is its track. The feature color indicates its speed, ranging from blue to red (lowest to highest speeds).

Cloud Instance Types and Costs To measure the cost-effectiveness of different instance types, we estimate the cost of analyzing one million images using different analysis programs. Figure 5.7 shows that different instance types are more cost-effective for different analysis programs. At low frame rates, memory optimized instances (e.g., `r3.xlarge`) are more cost-effective than compute optimized instances (e.g., `c4.xlarge`) for memory intensive analysis programs (e.g., ME). At high frame rates, compute optimized instances are more cost-effective for all analysis programs because the programs become CPU intensive as demonstrated in Section 5.3.2. These experiments show that cost can be reduced significantly by carefully selecting instance types based on the resource requirements of the analysis programs.



(a) 0.2 FPS



(b) 10 FPS

Fig. 5.7.: The cost-effectiveness of different instance types with different analysis programs and frame rates. Lower is better. HD can not be executed at high frame rates due to its long executing time. FT is not executed at low frame rates because features can not be tracked.

5.3.3 Evaluation of Resource Allocation

The Heuristic Algorithm for Vector Bin Packing

The proposed manager models the resource allocation problem as a vector bin packing problem and solves it using the heuristic algorithm proposed by Han et al. [4]. Using 550 test problems, the authors demonstrated that the overall average solution of the heuristic algorithm deviates by about 3% from the optimum (when available) and by about 5.4% from the lower bound when the optimum is not available.

In order to evaluate our use of the heuristic algorithm for resource allocation, we compare the results of both the heuristic algorithm and the exact method proposed by Brandao and Pedroso [51] and provided through VPSolver (Vector Packing Solver, `http://vpsolver.dcc.fc.up.pt/`). It is impractical to use VPSolver for large problems due to long execution times and limitations of the software. Hence, we consider three relatively simple (few programs and few cameras) scenarios as shown in Table 5.2. In each scenario, it is required to execute one or more analysis programs at different frame rates on the data streams from different numbers of cameras. The frame sizes of the cameras include 640×480 and 1280×720.

Table 5.3 shows the types and numbers of instances determined by the manager for each scenario while using both the heuristic algorithm and the exact method. Using the heuristic algorithm for Scenario A (or B) incurs 4.6% (or 5.3%) more cost compared to the exact method. For Scenario C, the manager uses three `c4.xlarge` instances using either of the two methods and the cost is the same. This demonstrates that the results of the heuristic algorithm are close to the exact method in terms of the overall cost.

Table 5.2.: The scenarios used to evaluate the heuristic algorithm for vector bin packing.

Scenario	Program	Frame Rate	Cameras
A	HD	0.5	100
B	FT	15.0	20
	HD	0.5	35
C	ME	0.2	40
	MOD	0.2	10
	HD	0.2	30

Table 5.3.: The overall cost and the types and numbers of instances determined by the manager for the scenarios shown in Table 5.2 using two different algorithms for vector bin packing (heuristic [4] and exact [51]).

Scenario	Method	Instances		Hourly
		`c4.xlarge`	`c4.2xlarge`	Cost
A	Heuristic	23	-	\$4.81
	Exact	18	2	\$4.60
B	Heuristic	20	-	\$4.18
	Exact	17	1	\$3.97
C	Heuristic	3	-	\$0.63
	Exact	3	-	\$0.63

The Resource Allocation Strategy

In order to evaluate the resource allocation strategy adopted by the proposed manager, we compare 5 different strategies as shown in Table 5.4. For a fair comparison, all the strategies benefit from the ability of the manager to estimate the resource requirements of analysis programs and the ability to model the resource allocation problem as a vector bin packing problem and solve it using the heuristic algorithm. Strategies 1, 2, 3, and 5 allow instances to be shared between different analysis programs. We compare the strategies using three different scenarios that are more complex (more programs and more cameras) than the scenarios in Table 5.2, hence can not be solved using VPSolver. With reference to Figure 5.7, the scenarios represent different types of workloads: a CPU intensive scenario, a memory intensive scenario, and a scenario that contains both CPU and memory intensive programs.

Table 5.4.: The strategies used to evaluate resource allocation. The enhanced manager uses ST5.

Abbr.	Resource Allocation Strategy
ST1	Always use `m4.xlarge` instances
ST2	Always use `c4.xlarge` instances
ST3	Always use `r3.xlarge` instances
ST4	Use the most cost-effective instance for each program without sharing instances between programs
ST5	**The enhanced manager:** Reduce the overall cost of the instances and allow sharing them between programs

Scenario 1, as described in Table 5.5, is CPU intensive. Table 5.6 shows the types and numbers of instances determined by each resource allocation strategy to handle this scenario. ST1, ST2, and ST3 use 70 instances because all the `m4.xlarge`, `c4.xlarge`, and `r3.xlarge` instances have the same CPU resources in terms of number of cores. ST4 and ST5 use the compute optimized `c4.xlarge` instances since they are the most cost-effective instances for this CPU intensive scenario. ST4 uses 81 `c4.xlarge` instances because it does not allow instances to be shared between FT and HD. ST5 further reduces the overall cost by allowing instances to be shared between FT and HD. ST5 incurs the same cost as ST2 because ST2 always uses compute optimized instances. This scenario demonstrates the ability of the manager (ST5) to reduce the overall cost by 37% compared with other strategies (i.e., ST3).

Scenario 2, as described in Table 5.7, is memory intensive. Table 5.8 shows the types and numbers of instances determined by each resource allocation strategy to handle this scenario. ST2 requires the most number of instances since each `c4.xlarge`

Table 5.5.: The details of the CPU intensive Scenario 1.

Program	Frame Rate	Cameras	Frame Sizes
FT	15.00	25	640×480
HD	0.50	250	1280×720, 640×480

Table 5.6.: The types and numbers of instances determined using the allocation strategies described in Table 5.4 to handle Scenario 1 shown in Table 5.5. All instances are `xlarge`. Cost savings are relative to the highest cost.

	Instances			Hourly	Cost
	`m4.x`	`c4.x`	`r3.x`	Cost	Savings
ST1	70	-	-	\$16.73	28%
ST2	-	70	-	\$14.63	37%
ST3	-	-	70	\$23.31	0%
ST4	-	81	-	\$16.93	27%
ST5	-	70	-	\$14.63	37%

instance has only 7.5 GB of memory. ST1 requires fewer instances (i.e., 55) since each `m4.xlarge` instance has more memory resources (i.e., 16 GB). ST3 requires the fewest number of instances (i.e., 29) since each `r3.xlarge` has more memory resources (i.e., 30.5 GB). ST4 and ST5 use the memory optimized `r3.xlarge` instances since they are the most cost-effective instances for this memory intensive scenario. ST4 uses 30 `r3.xlarge` instances because it does not allow instances to be shared between ME and MOD. ST5 further reduces the overall cost by allowing instances to be shared between ME and MOD. ST5 incurs the same cost as ST3 because ST3 always uses memory optimized instances. This scenario demonstrates the ability of the manager (ST5) to reduce the overall cost by 60% compared with other strategies (i.e., ST2).

Scenario 3, as described in Table 5.9, contains both CPU and memory intensive programs. Table 5.10 shows the types and numbers of instances determined by each resource allocation strategy to handle this scenario. ST1, ST2, and ST3 use 69, 91,

Table 5.7.: The details of the memory intensive Scenario 2.

Program	Frame Rate	Cameras	Frame Sizes
ME	0.10	5000	640×480
MOD	0.05	3000	1920×1080, 640×480

Table 5.8.: The types and numbers of instances determined using the allocation strategies described in Table 5.4 to handle Scenario 2 shown in Table 5.7. All instances are `xlarge`. Cost savings are relative to the highest cost.

	Instances			Hourly	Cost
	`m4.x`	`c4.x`	`r3.x`	Cost	Savings
ST1	55	-	-	$13.15	46%
ST2	-	117	-	$24.45	0%
ST3	-	-	29	$9.66	60%
ST4	-	-	30	$9.99	59%
ST5	-	-	29	$9.66	60%

Table 5.9.: The details of Scenario 3.

Program	Frame Rate	Cameras	Frame Sizes
ME	0.20	4000	1280×720, 640×480
MOD	0.20	1000	1280×720, 640×480
FT	10.00	10	640×480
HD	0.20	300	1280×720, 640×480

and 57 instances according to the CPU and memory resources of the respective instances. ST4 and ST5 use the most cost-effective instances for each analysis program: `c4.xlarge` for the CPU intensive FT and HD, `r3.xlarge` for the memory intensive ME, and `m4.xlarge` for the more balanced MOD. ST5 uses fewer instances than SR4 by allowing instances to be shared between the programs. This scenario demonstrates

Table 5.10.: The types and numbers of instances determined using the allocation strategies described in Table 5.4 to handle Scenario 3 shown in Table 5.9. All instances are `xlarge`. Cost savings are relative to the highest cost.

	Instances			Hourly	Cost
	`m4.x`	`c4.x`	`r3.x`	Cost	Savings
ST1	69	-	-	\$16.49	13%
ST2	-	91	-	\$19.02	0%
ST3	-	-	57	\$18.98	0%
ST4	11	30	19	\$15.23	20%
ST5	9	30	18	\$14.42	24%

the ability of the manager (ST5) to reduce the overall cost by 24% compared with other strategies (i.e., ST2).

Figure 5.8 compares the overall cost incurred when using each of the 5 strategies to handle each of the 3 scenarios. The figure shows that different strategies are better for differnt scenarios in terms of reducing the overall cost. ST2 and ST5 are the best strategies for the CPU intensive scenario 1. ST3 and ST5 are the best strategies for the memory intensive scenario 2. However, the strategy of the proposed manager (ST5) is always the best and it reduces up to 60% of the cost of other strategies.

Large-Scale Experiment

In this section, we conduct the large-scale experiment specified by Scenario 3 (shown in Table 5.9) for 24 hours. The experiment analyzes more than 97 million images from 5,310 cameras over 24 hours. That is more than 3.3 TB of data. The experiment uses 15 instances and the overall cost is \$188. In this experiment, the manager considers that cloud vendors impose limits on the types and numbers of running instances by each user. For example, Amazon limits the number of running `m4.2xlarge` instances by each user in a single region to 5. The same applies for `c4.2xlarge` and `r3.2xlarge` instances. In this experiment, we consider using the

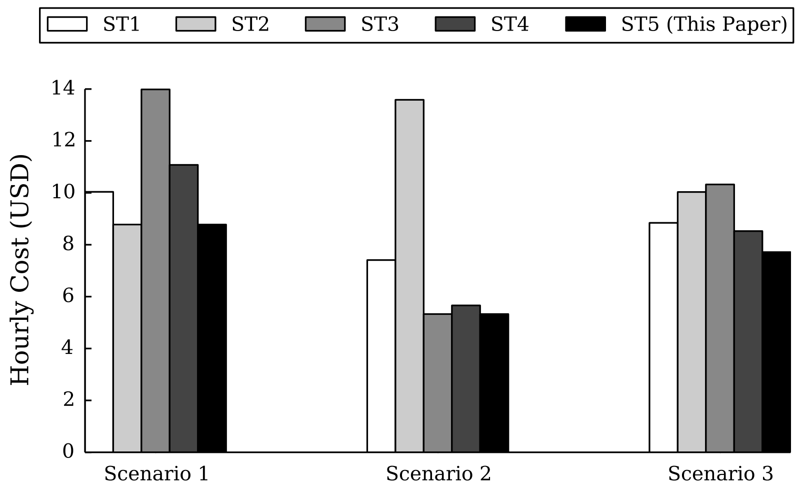
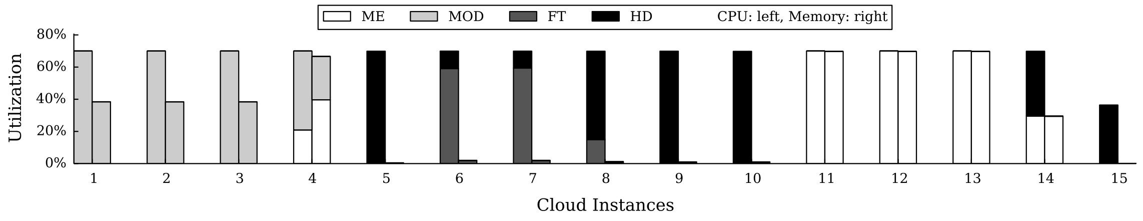


Fig. 5.8.: The overall hourly cost of different scenarios (Table 5.5, Table 5.7, and Table 5.9) using different resource allocation strategies (Table 5.4). ST5 reduces up to 60% of the cost of other strategies.

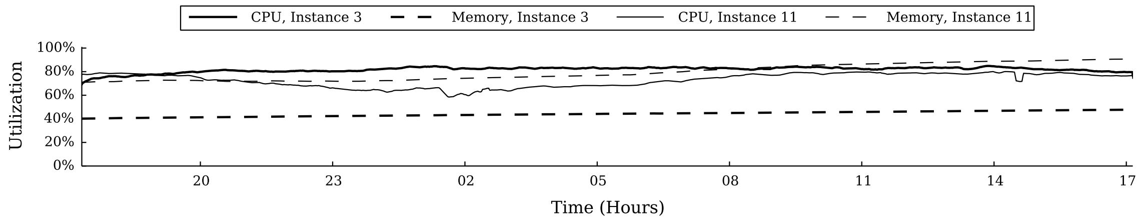
`2xlarge` instances (as opposed to `xlarge`) in order to conduct a large-scale experiment with fewer instances.

Figure 5.9 and Figure 5.10 show the results of the experiment. Figure 5.9(a) shows the estimated CPU and memory utilization of the allocated instances. The manager targets a 70% utilization for both the CPU and memory resources in order to accommodate for the varying resource requirements of the analysis programs. The figure shows, for example, the estimated utilization of instance 1 is 70% for the CPU and 40% for the memory. The figure also shows how the instances are shared between different analysis programs (e.g., ME and MOD in instance 4) so that resources are efficiently utilized and the overall cost is reduced. CPU intensive programs (i.e., FT and HD) mostly use compute optimized instances (i.e., `c4.2xlarge`). ME is memory intensive so it mostly uses memory optimized instances (i.e., `r3.2xlarge`). MOD has a relatively more balanced CPU and memory requirements so it mostly uses general-purpose instances (i.e., `m4.2xlarge`).

Figure 5.9(b) shows the actual CPU and memory utilization of instance 3 and instance 11 over the entire analysis duration. The figure shows that the actual utilization is close to the estimated utilization shown in Figure 5.9(a). The figure also shows that the resource utilization varies over time. In this experiment, the manager successfully meets the performance requirements for all the analysis programs: $\eta = 98\%$ for ME, 94% for MOD, 94% for FT, and 98% for HD. Figure 5.10 shows sample results.



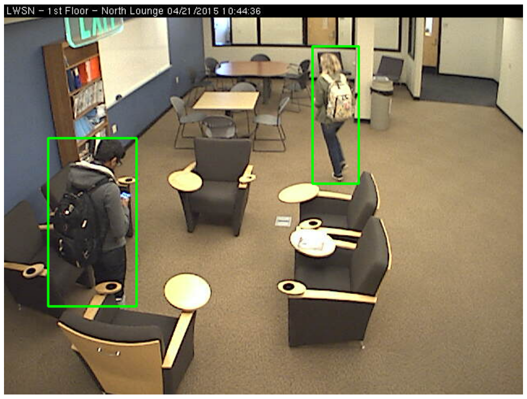
(a) The estimated CPU (left bars) and memory (right bars) utilization of each instance: (1-5) `m4.2xlarge`, (6-10) `c4.2xlarge`, and (11-15) `r3.2xlarge`. The target resource utilization is 70% to accommodate for the varying resource requirements. Bars may be split if multiple analysis programs use the same instance. Each program is executed on the data streams from many cameras.



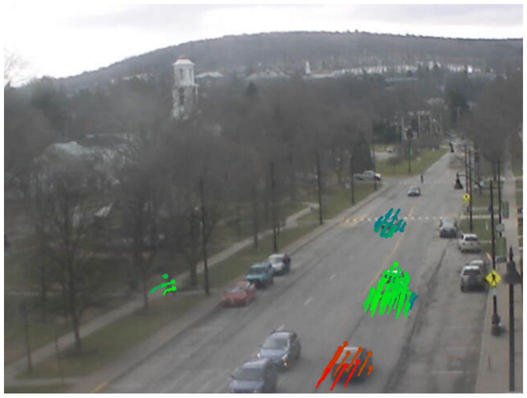
(b) The actual CPU and memory utilization of instance 3 and instance 11 from Figure 5.9(a) over the entire analysis duration. The lines are smoothed using a moving average filter with a 10-minute window.

Fig. 5.9.: Using 15 instances to analyze more than 97 million images (more than 3.3 TB) from 5,310 cameras simultaneously over 24 hours using different analysis programs and frame rates (Scenario 3 in Table 5.5).

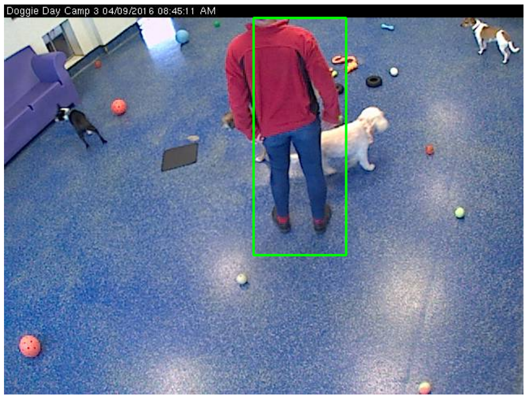
(a) ME: Foreground mask of a moving car



(b) MOD: Two moving objects detected



(c) FT: Tracked features of several cars and pedestrians



(d) HD: One human detected

Fig. 5.10.: Sample output results of using 15 instances to analyze more than 97 million images (more than 3.3 TB) from 5,310 cameras simultaneously over 24 hours using different analysis programs and frame rates (Scenario 3 in Table 6.5). (d) A circle is a moving feature, and a line is its track. The feature color indicates its speed, ranging from blue to red (lowest to highest speeds).

6. MANAGEMENT OF CPU AND GPU RESOURCES

Some cloud instances have GPUs and some instances do not (referred to as GPU instances and non-GPU instances respectively). Using GPUs can accelerate analysis programs and achieve higher frame rates, but incurs additional cost because GPU instances are more expensive. This chapter proposes a resource manager that uses the GPU to achieve frame rates that are not possible using the CPU only. The manager also considers both GPU and non-GPU instances to reduce the overall cost. The chapter is organized as follows: Section 6.1 explains the existing problems and how GPUs can help solving these problems. Section 6.2 describes the proposed resource manager. Section 6.3 evaluates the manager.

6.1 Existing Problems and How GPUs Can Help

The ultimate goal of the resource manager is to reduce the overall cost and meet the performance requirements. Several problems can prevent the manager from achieving its goal:

1. Some analysis programs can not achieve their desired frames due to their long execution times. This problem prevents the manager from meeting the performance requirements ($\eta > 90\%$). For example, Figure 5.2 shows that the CPU alone is not able to execute FT (or MOD) at 12 FPS (or 24 FPS).

2. Executing analysis programs at high frame rates is computationally intensive (and sometimes at low frame rates as shown in Figure 5.1). This causes the CPU to be the bottleneck in the system. For example, Figure 5.9(a) shows that the CPU resources are fully utilized while the memory resources are highly

underutilized for many instances. This prevents the manager from reducing the overall cost by using less instances.

GPUs have been extensively used to accelerate general-purpose applications. This is achieved by offloading compute-intensive portions of the code to the GPU, while the rest of the code is still executed on the CPU. While a CPU has a few powerful cores optimized for sequential processing, a GPU has thousands of smaller cores designed for handling thousands of tasks simultaneously. GPU-based applications often achieve speedups of orders of magnitude compared to CPU-based applications (depending on the capabilities of the CPUs and GPUs, and the applications). Using GPUs in CAM^2 can tackle the two problems mentioned above:

1. GPUs can be used to accelerate analysis programs and achieve high speedups depending on the GPUs and the programs. This enables the resource manager to achieve higher frame rates in order to meet the performance requirements.
2. Using GPUs reduces the workload on the CPU. Since the CPU is usually the bottleneck in the system, fewer instances may be required and the overall cost may be reduced.

6.2 Resource Allocation Using CPU and GPU Resources

The existence of GPU resources impacts the factors and decisions of resource allocation shown in Figure 4.1. Section 6.2.1 and Section 6.2.2 discuss these impacts and show how the new resource manager handles them.

6.2.1 Factors Affecting Resource Allocation Decisions

The proposed resource manager considers the following factors while making allocation decisions:

1. Resource Requirements: The manager considers the following types of resources: CPU, memory, GPU, and GPU memory. Different analysis programs require

Table 6.1.: A hypothetical example for the resource requirements of two analysis programs (P1 and P2).

Resource	P1	P2	P2 using GPU
CPU	5%	50%	10%
Memory	10%	5%	3%
GPU	0%	0%	10%
GPU Memory	0%	0%	7%

different amounts of resources. For example, some programs are memory intensive (e.g. P1 in Table 6.1) while others are CPU intensive (e.g. P2 in Table 6.1). Moreover, some programs can be accelerated using the GPU and their resource requirements change accordingly. For example, executing P2 using the GPU reduces its CPU requirement from 50% to 10% and increases its GPU requirement from 0% to 10%.

The resource manager is designed to be used for a variety of applications. Hence, it does not assume any prior knowledge about the analysis programs' resource requirements. The manager conducts a test run to estimate the resource requirements of each program by monitoring the utilization of resources. The test run is conducted once and the estimations of the resource requirements can be used for future executions of the same program.

2. Desired Frame Rates: The frame rate at which an analysis program is executed significantly affects its resource requirements. Experiments show that the CPU and GPU requirements of an analysis program increase linearly with its frame rate. Using this linear relationship, the resource manager can estimate the resource requirements of an analysis program at different frame rates using a single test run conducted at a particular frame rate. In addition, the frame rate may affect different types of resources differently. For example, increasing the frame rate of a program may increase its CPU requirement, but may have no effect on its memory requirement.

Table 6.2.: The capabilities and the hourly costs of some Amazon EC2 instance types with and without GPUs (at Oregon).

Instance	Cores	Memory (GB)	GPUs	Cost
c4.2xlarge	8	15	-	\$0.419
c4.8xlarge	36	60	-	\$1.675
g2.2xlarge	8	15	1	\$0.650
g2.8xlarge	32	60	4	\$2.600

This causes some analysis programs to be CPU intensive at high frame rates while being memory intensive at low frame rates.

3. Frame Sizes: Different cameras provide streams with different frame sizes (e.g. 640×480 pixels and 1920×1080 pixels). In general, the higher the frame size is, the higher the resource requirements are. The effect of the frame size on the resource requirements of an analysis program depends on the time complexity and the space complexity of the program. Since the resource manager assumes no prior knowledge about analysis programs, the manager conducts a test run for each unique frame size. Fortunately, there are only a few common frame sizes among network cameras.

4. Types and Costs of Cloud Instances: Cloud vendors offer many instances with different capabilities and hourly costs. Table 6.2 shows the capabilities and hourly costs of some Amazon EC2 instance types with and without GPUs. The table shows that some instances do not have GPUs (i.e., c2.2xlarge and c2.8xlarge), the g2.2xlarge instance has a single GPU, and the g2.8xlarge instance has 4 GPUs. The table also shows that GPU instances (i.e., g2.2xlarge and g2.8xlarge) are more expensive than non-GPU instances (i.e., c2.2xlarge and c2.8xlarge). The manager decides the types and number of instances needed to reduce the overall cost while meeting the desired frame rates.

6.2.2 Resource Allocation Using Multiple-Choice Vector Bin Packing

To make the resource allocation decisions shown in Figure 4.1, this manager formulates resource allocation as a multiple-choice vector bin packing problem. In this problem, there are bins and objects. Each bin has a cost and a multidimensional size. Each object may have one of several possible sizes (multiple choices). The goal is to pack all the objects into bins such that: (i) One size is selected for each object. (ii) The overall cost of all the used bins is minimized. (iii) The total size of all the objects in each bin does not exceed its size in any dimension.

Similarly, in the resource allocation problem, there are instances and camera streams. Each instance has an hourly cost and a multidimensional vector representing its resource capabilities (i.e., CPU, memory, GPU, and GPU memory). For example, the vector $[8, 15, 0, 0]$ represents an instance with 8 CPU cores, 15 GB of memory, and no GPUs (i.e., c4.2xlarge). The vector $[8, 15, 1536, 4]$ represents an instance with 8 CPU cores, 15 GB of memory, and a single GPU with 1536 cores and 4 GB of memory (i.e., g2.2xlarge). Each camera stream may have one of two possible resource requirements depending on whether it is executed by the CPU or the GPU. The resource requirements of P2 in Table 6.2 is represented by the vector $[4, 0.75, 0, 0]$ or $[0.8, 0.45, 153.6, 0.28]$ if it is executed by the CPU or the GPU respectively. The goal is to assign all the streams to instances such that: (i) One resource requirement is selected for each stream. This implies deciding if the stream is analyzed by the CPU or the GPU. (ii) The overall cost of all the used instances is minimized. (iii) The total resource requirements of all the streams in each instance do not exceed the instance's resource capabilities in each dimension (i.e., CPU, memory, GPU, and GPU memory). This ensures that all the resources are not overutilized so that the manager can meet the desired frame rates.

If instances with multiple GPUs (e.g. g2.8xlarge) are available, the dimensions and the multiple-choices of the problem change accordingly. For example, the vector $[8, 15, 1536, 4, 1536, 4, 1536, 4, 1536, 4]$ represents an instance with 8 CPU cores, 15 GB of memory, and 4 GPUs each with 1536 cores and 4 GB of memory (i.e., g2.8xlarge). In this case, the vector $[8, 15, 0, 0, 0, 0, 0, 0, 0, 0]$ represents an instance with 8 CPU cores, 15 GB of memory, and no GPUs (i.e., c4.2xlarge). Each camera stream may have one of 5 possible resource requirements depending on whether it is executed by the CPU or one of the 4 GPUs. For example, the resource requirements of P2 in Table 6.2 is represented by the vector $[4, 0.75, 0, 0, 0, 0, 0, 0, 0, 0]$, $[0.8, 0.45, 153.6, 0.28, 0, 0, 0, 0, 0, 0]$, $[0.8, 0.45, 0, 0, 153.6, 0.28, 0, 0, 0, 0]$, $[0.8, 0.45, 0, 0, 0, 0, 153.6, 0.28, 0, 0]$, or $[0.8, 0.45, 0, 0, 0, 0, 0, 0, 153.6, 0.28]$ if it is executed by the CPU, GPU1, GPU2, GPU3, or GPU4 respectively. In general, the dimension of the problem is $2 + 2 \times N$ where N is the maximum number of GPUs in any instance. That is because there are 2 resource types (i.e., CPU and memory) for any instance and 2 more resource types (i.e. GPU and GPU memory) for each added GPU. The number of choices for the resource requirements of each stream is $1 + N$ because the stream can be analyzed either by the CPU or by one of the N GPUs.

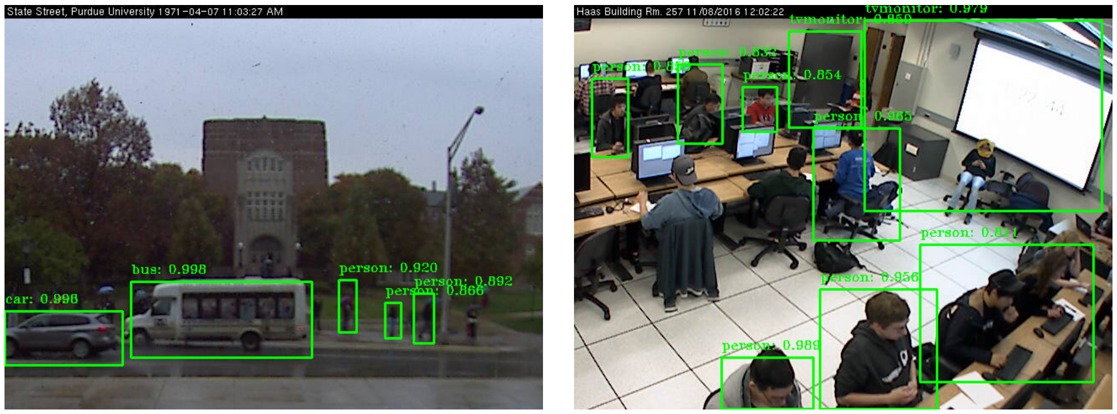
To solve the multiple-choice vector bin packing, the manager uses the exact method proposed by Brandao and Pedroso [51] and provided through VPSolver (Vector Packing Solver, `http://vpsolver.dcc.fc.up.pt/`). The output of the solver is the types and numbers of bins required to pack all the objects, which objects are assigned to each bin, and the selected size of each object. In the resource allocation problem, this maps to the types and numbers of instances required to analyze all the camera streams, which streams are assigned to each instance, and the selected resource requirement of each stream (i.e. which CPU or GPU to analyze the stream).

6.3 Experiments

Section 6.3.1 explains the experimental setup. Section 6.3.2 evaluates the speedup that can be achieved using the GPU. Sections 6.3.3, 6.3.4, and 6.3.5 evaluate the effect of the desired frame rates, analysis programs, and number of cameras on the resources. Section 6.3.6 evaluates the resource allocation strategy of the proposed manager.

6.3.1 Experimental Setup

The experiments use two analysis programs for object detection. The two programs use two convolutional neural networks (VGG-16 [5] and ZF [6]) to detect objects (e.g. persons and cars) in images. The experiments use the Python implementation of the region proposal network proposed by Ren et al. [61] to reduce the execution time of VGG-16 and ZF. Figure 6.1 shows sample outputs. All the experiments use these programs to analyze 640×480 MJPEG streams from network cameras.



(a) VGG-16 (b) ZF

Fig. 6.1.: Sample output results from two network cameras. The objects detected are persons, cars, buses, and TV monitors.

The experiments use a machine with an 8-core Intel Xeon E5-2623 v3 CPU and 32GB of memory. The machine has an NVIDIA K40 GPU with a 12GB of memory. When the GPU is not used, the experiments refer to the machine as a *non-GPU* *instance* and the cost is assumed to be the same as c4.2xlarge as shown in Table 6.2. When the GPU is used, the experiments refer to the machine as *GPU instance* and the cost is assumed to be the same as g2.2xlarge. The resource manager is generic and can be used with different cloud vendors (e.g. Amazon EC2 and Microsoft Azure) with the appropriate changes in instance capabilities and hourly costs. The experiments focus on the CPU and GPU utilization without the memory and GPU memory utilization, but the resource manager is generic and considers all these resource types.

6.3.2 Speedup Achieved Using GPU

The main goal of the resource manager is to meet the desired frame rates of the analysis programs. Using the GPU to accelerate the programs allows the manager to achieve frame rates that are not possible using the CPU only. Figure 6.2 shows the effect of using the GPU on the maximum achievable frame rates of different analysis programs. VGG-16 (or ZF) can be executed at 3.61 (or 9.15) FPS using the GPU, but 0.28 (or 0.56) using the CPU only. This experiment shows that the resource manger can use the GPU to achieve a speedup of around 13 (or 16) for VGG-16 (or ZF).

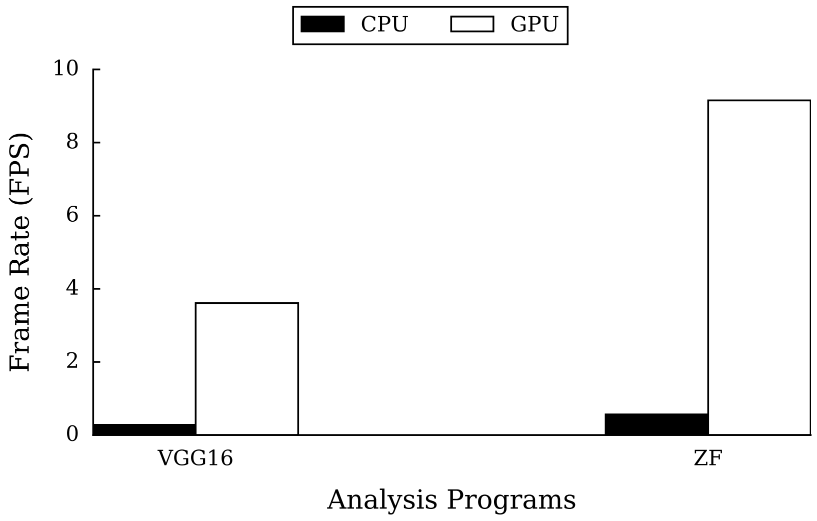


Fig. 6.2.: The effect of using the GPU on the maximum achievable frame rates. Speedup: 13 for VG166 and 16 for ZF.

6.3.3 Effect of the Desired Frame Rates

Desired frame rates significantly affect the resource requirements of analysis programs as well as the analysis performance. Figure 6.3 shows this effect by executing VGG-16 using the GPUa t different frame rates. The figure shows that, at the beginning, the CPU and GPU utilization increase linearly with the frame rate and the performance is 100%. The performance stars to drop gradually after the CPU resources are used up. Since the resource manager aims at maintaining the analysis performance above 90%, the manager allocates cloud instances such that no resource utilization is above 90%.

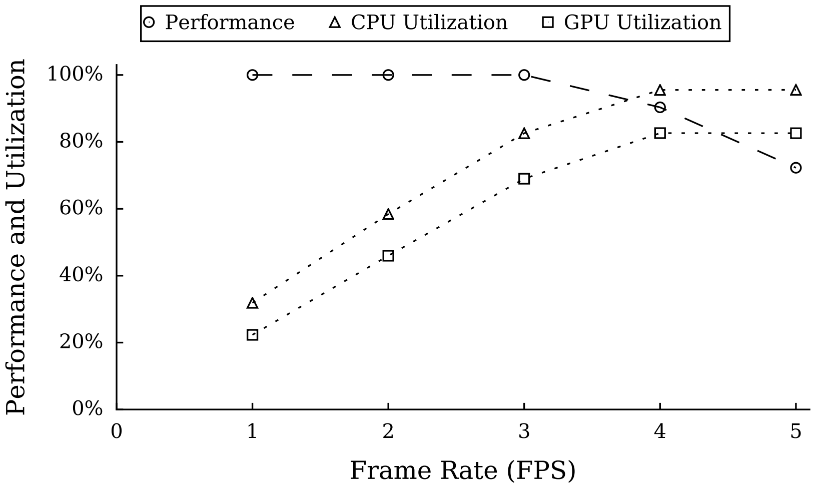


Fig. 6.3.: The effect of the desired frame rate on the resource requirements of VGG-16 as well as the analysis performance.

6.3.4 Resource Requirements of Analysis Programs

Table 6.3 shows the CPU and GPU requirements of VGG-16 and ZF if executed at 0.2 FPS using the CPU only or using the GPU. This shows that for each analysis program, there are two choices of resource requirements depending on whether it is executed by the CPU or the GPU. The manager estimates these resource requirements at different frame rates based on the test run (e.g. Figure 6.2) conducted at a particular frame rate and the linear relationship between the frame rate and the CPU and GPU utilization shown in Figure 6.3.

Table 6.3.: The CPU and GPU requirements of VGG-16 and ZF if executed at 0.2 FPS using the CPU only or using the GPU.

Program	Using CPU		Using GPU	
	CPU	GPU	CPU	GPU
VGG-16	39.4%	-	5.3%	4.6%
ZF	17.8%	-	2.2%	1.2%

6.3.5 Effect of the Number of Cameras

The number of camera streams being analyzed using a single instance affects its resource utilization as well as the analysis performance. Figure 6.4 shows this effect by using the GPU to execute VGG-16 at 2 FPS on the data streams from multiple cameras. The figure shows that, at the beginning, the CPU and GPU utilization increase almost linearly with the number of cameras and the performance is 100%. The performance stars to drop gradually after the CPU and GPU resources are used up. Since the resource manager aims at maintaining the analysis performance above 90%, the manager assigns streams to instances such that no resource utilization is above 90%.

6.3.6 Evaluation of Resource Allocation

To evaluate the resource allocation strategy of the proposed manager, we compare it with two different strategies as shown in Table 6.4. All the strategies benefit from the ability of the manager to estimate the resource requirements of different analysis programs, to formulate the problem as a multiple-choice vector bin packing problem, and to solve it. For ST1 (or ST2), there is a single choice for the resource requirements of each analysis program because only non-GPU (or GPU) instances are considered.

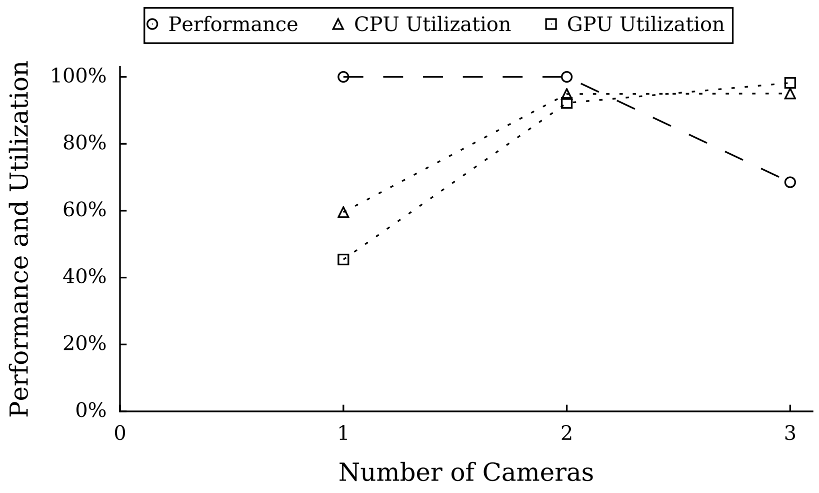


Fig. 6.4.: The effect of the number of camera streams being analyzed (using VGG-16 at 2 FPS) on the resource utilization as well as the analysis performance.

The proposed manager uses ST3 which considers both non-GPU and GPU instances. In this case, two choices of resource requirements exist for each analysis program depending on whether it is executed by the CPU or the GPU.

Table 6.4.: The strategies used to evaluate resource allocation. This manager uses ST3.

Abbr.	Resource Allocation Strategy
ST1	**The Enhanced Manager (Chapter 5):** Always use non-GPU instances
ST2	Always use GPU instances
ST3	**This Manager (Chapter 6):** Use non-GPU and GPU instances to reduce the overall cost of the instances

In order to compare the three resource allocation strategies, we use the three scenarios described in Table 6.5. The table shows the programs, frame rates, and the number of camera streams being analyzed in each scenario. Table 6.6 shows the types and numbers of instances determined by each strategy to handle each scenario:

Table 6.5.: The scenarios used to compare different resource allocation strategies.

Scenario	Program	Frame Rate	Cameras
1	VGG-16	0.25	1
	ZF	0.55	3
2	VGG-16	0.20	1
	ZF	0.50	1
3	VGG-16	0.20	2
	ZF	8.00	10

Table 6.6.: The types and numbers of instances determined by the different allocation strategies in Table 6.4 to handle the different scenarios in Table 6.5.

Scen.	Strategy	Instances		Hourly Cost	Cost Savings
		non-GPU	GPU		
1	ST1	4	-	\$1.676	0%
	ST2	-	1	\$0.650	61%
	ST3	-	1	\$0.650	61%
2	ST1	1	-	\$0.419	36%
	ST2	-	1	\$0.650	0%
	ST3	1	-	\$0.419	36%
3	ST1	Fail	Fail	Fail	Fail
	ST2	-	11	\$7.150	0%
	ST3	1	10	\$6.919	3%

Scenario 1: ST1 uses 4 non-GPU instances to handle the 4 camera streams. That is because a single non-GPU instance can handle only one stream due to the high CPU requirement of VGG-16 at 0.25 FPS (or ZF at 0.55 FPS). ST2 uses a single GPU instance to handle all the 4 streams because the CPU requirement is decreased significantly while using the GPU. ST3 makes the same decisions as ST2 and either of them saves 61% of the overall hourly cost compared with ST1.

Scenario 2: The CPU and GPU requirements of VGG-16 at 0.2 FPS and ZF at 0.5 FPS are relatively low such that a single instance can handle the two given camera streams at the same time. ST1 uses a single non-GPU instance while ST2 uses a single GPU instance. ST3 makes the same decisions as ST1 and either of them saves 36% of the overall hourly cost compared with ST2.

Scenario 3: ST1 fails to execute ZF at 8 FPS since the CPU only can execute ZF at a maximum of 0.56 FPS as shown in Figure 6.2. ST2 uses 10 GPU instances to handle the 10 camera streams of ZF and a single GPU instance to handle both the 2 streams of VGG-16. That is because a single GPU instance can handle only one stream of ZF at 8 FPS due to the high CPU requirement. ST3 considers both GPU and non-GPU instances to reduce the overall hourly cost so it can replace a GPU instance with a non-GPU instance. Hence, ST3 saves 3% of the cost compared with ST2.

These experiments demonstrate that different resource allocation strategies are best in different scenarios according to several factors, such as analysis programs and frame rates. The strategy used by the proposed resource manager considers both GPU and non-GPU instances and always have the lowest cost compared with the other strategies (e.g. 61% cost savings in Scenario 1).

7. CONCLUSION

This dissertation introduces CAM^2, a web-based system that enables users to analyze the real-time visual data from thousands of network cameras simultaneously. CAM^2 can retrieve data from the heterogeneous cameras and execute analysis programs using the cloud. The event-driven API simplifies migrating existing analysis programs to CAM^2. This dissertation also proposes cloud resource managers that reduce the cost for analyzing real-time data streams from thousands of network cameras while meeting the performance requirements. The managers allocate cloud instances based on many factors, including the analysis programs, the desired frame rates, the camera frame sizes, and the types and costs of the instances. The resource managers monitor the allocated instances; they allocate more instances if needed and deallocate existing instances to reduce the cost if possible. The experiments show that the resource managers are able to reduce up to 61% of the overall analysis cost. One experiment analyzes more than 97 million images (3.3 TB of data) from 5,310 cameras simultaneously over 24 hours using 15 instances. Readers interested using CAM^2 can register at `https://cam2.ecn.purdue.edu/` and become users.

REFERENCES

REFERENCES

[1] "Network camera and video analytics market," http://www.marketsandmarkets.com/Market-Reports/visual-communication-market-775.html.

[2] "CAM^2 website," https://cam2.ecn.purdue.edu/.

[3] "Amazon EC2," https://aws.amazon.com/ec2/.

[4] B. T. Han, G. Diehr, and J. S. Cook, "Multiple-type, two-dimensional bin packing problems: Applications and algorithms," *Annals of Operations Research*, vol. 50, no. 1, pp. 239–261, 1994.

[5] K. Simonyan and A. Zisserman, "Very deep convolutional networks for large-scale image recognition," *CoRR*, vol. abs/1409.1556, 2014.

[6] M. D. Zeiler and R. Fergus, "Visualizing and understanding convolutional networks," *CoRR*, vol. abs/1311.2901, 2013.

[7] N. Jacobs, N. Roman, and R. Pless, "Consistent temporal variations in many outdoor scenes," in *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, 2007, pp. 1–6.

[8] S. Srivastava and E. J. Delp, "Video-based real-time surveillance of vehicles," *Journal of Electronic Imaging*, vol. 22, no. 4, p. 041103, 2013.

[9] L. Kratz and K. Nishino, "Anomaly detection in extremely crowded scenes using spatio-temporal motion pattern models," in *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, 2009, pp. 1446–1453.

[10] G. S. Thakur, P. Hui, and A. Helmy, "A framework for realistic vehicular network modeling using planet-scale public webcams," in *Proceedings of the ACM International Workshop on Hot Topics in Planet-scale Measurement*, 2012, pp. 3–8.

[11] J.-F. Lalonde, A. A. Efros, and S. G. Narasimhan, "Webcam clip art: Appearance and illuminant transfer from time-lapse sequences," *ACM Transactions on Graphics*, vol. 28, no. 5, pp. 131:1–131:10, 2009.

[12] Z. Chen, F. Yang, A. Lindner, G. Barrenetxea, and M. Vetterli, "Howis the weather: Automatic inference from images," in *Proceedings of the IEEE International Conference on Image Processing*, 2012, pp. 1853–1856.

[13] N. Jacobs, R. Souvenir, and R. Pless, "Passive vision: The global webcam imaging network," in *Proceedings of the IEEE Applied Imagery Pattern Recognition Workshop*, 2009, pp. 1–8.

[14] "Phenocam by the university of new hampshire," http://phenocam.sr.unh.edu/webcam/.

[15] E. A. Graham, E. C. Riordan, E. M. Yuen, D. Estrin, and P. W. Rundel, "Public internet-connected cameras used as a cross-continental ground-based plant phenology monitoring system," *Global Change Biology*, vol. 16, no. 11, pp. 3014–3023, 2010.

[16] S. Winkler, M. Zessner, E. Saracevic, K. Ruzicka, N. Fleischmann, and U. Wegricht, "Investigative monitoring in the context of detecting anthropogenic impact on an epipotamal river." *Water Science & Technology*, vol. 57, no. 7, 2008.

[17] T. E. Gilmore, F. Birgand, and K. W. Chapman, "Source and magnitude of error in an inexpensive image-based water level measurement system," *Journal of Hydrology*, vol. 496, pp. 178–186, 2013.

[18] L. Goddijn-Murphy, D. Dailloux, M. White, and D. Bowers, "Fundamentals of in situ digital camera methodology for water quality monitoring of coast and ocean," *Sensors*, vol. 9, no. 7, p. 5825, 2009.

[19] K. Ruzicka, O. Gabriel, U. Bletterie, S. Winkler, and M. Zessner, "Cause and effect relationship between foam formation and treated wastewater effluents in a transboundary river," *Physics and Chemistry of the Earth, Parts A/B/C*, vol. 34, no. 89, pp. 565–573, 2009.

[20] K. Hong, M. Voelz, V. Govindaraju, B. Jayaraman, and U. Ramachandran, "A distributed framework for spatio-temporal analysis on large-scale camera networks," in *Proceedings of the IEEE International Conference on Distributed Computing Systems Workshops*, 2013, pp. 309–314.

[21] C. Feris, A. Hampapur, Y. Zhai, R. Bobbitt, L. Brown, D. A. Vaquero, Y.-l. Tian, H. Liu, and M.-T. Sun, "Case study: Ibm smart surveillance system," *Yunqian Ma et al., Intelligent Video Surveillance: System and Technology*, pp. 47–76, 2009.

[22] T. Yu, B. Zhou, Q. Li, R. Liu, W. Wang, and C. Chang, "The service architecture of real-time video analytic system," in *Proceedings of the IEEE International Conference on Service-Oriented Computing and Applications*, 2009, pp. 1–8.

[23] K. Hong, S. Smaldone, J. Shin, D. Lillethun, L. Iftode, and U. Ramachandran, "Target container: A target-centric parallel programming abstraction for video-based surveillance," in *Proceedings of the ACM/IEEE International Conference on Distributed Smart Cameras*, 2011, pp. 1–8.

[24] W. Zhu, C. Luo, J. Wang, and S. Li, "Multimedia cloud computing," *IEEE Signal Processing Magazine*, vol. 28, no. 3, pp. 59–69, 2011.

[25] F. Wang, J. Liu, and M. Chen, "Calms: Cloud-assisted live media streaming for globalized demands with time/region diversities," in *Proceedings of the IEEE International Conference on Computer Communications*, 2012, pp. 199–207.

[26] Y. Wu, C. Wu, B. Li, X. Qiu, and F. Lau, "Cloudmedia: When cloud on demand meets video on demand," in *Proceedings of the International Conference on Distributed Computing Systems*, 2011, pp. 268–277.

[27] M. S. Hossain, M. M. Hassan, M. A. Qurishi, and A. Alghamdi, "Resource allocation for service composition in cloud-based video surveillance platform," in *Proceedings of the IEEE International Conference on Multimedia and Expo Workshops*, 2012, pp. 408–412.

[28] S. Vijayakumar, Q. Zhu, and G. Agrawal, "Dynamic resource provisioning for data streaming applications in a cloud environment," in *Proceedings of the IEEE International Conference on Cloud Computing Technology and Science*, 2010, pp. 441–448.

[29] J. Fung and S. Mann, "Using graphics devices in reverse: Gpu-based image processing and computer vision," in *IEEE International Conference on Multimedia and Expo*, June 2008, pp. 9–12.

[30] C. Oh, S. Yi, and Y. Yi, "Real-time face detection in full hd images exploiting both embedded cpu and gpu," in *IEEE International Conference on Multimedia and Expo*, June 2015, pp. 1–6.

[31] C. Y. Lee, Y. C. Lin, C. L. Wu, C. H. Chang, Y. M. Tsao, and S. Y. Chien, "Multi-pass and frame parallel algorithms of motion estimation in h.264/avc for generic gpu," in *IEEE International Conference on Multimedia and Expo*, July 2007, pp. 1603–1606.

[32] M. Alcoverro, A. Lpez-Mndez, J. R. Casas, and M. Pards, "A real-time body tracking system for smart rooms," in *IEEE International Conference on Multimedia and Expo*, July 2011, pp. 1–6.

[33] A. S. Kaseb, E. Berry, Y. Koh, A. Mohan, W. Chen, H. Li, Y.-H. Lu, and E. J. Delp, "A system for large-scale analysis of distributed cameras," in *Proceedings of the IEEE Global Conference on Signal and Information Processing*, 2014, pp. 340–344.

[34] A. S. Kaseb, E. Berry, E. Rozolis, K. McNulty, S. Bontrager, Y. Koh, Y.-H. Lu, and E. J. Delp, "An interactive web-based system using cloud for large-scale visual analytics," in *Proceedings of the Imaging and Multimedia Analytics in a Web and Mobile World*, 2015.

[35] A. S. Kaseb, Y. Koh, E. Berry, K. McNulty, Y.-H. Lu, and E. J. Delp, "Multimedia content creation using global network cameras: The making of cam2," in *Proceedings of the IEEE Global Conference on Signal and Information Processing*, 2015.

[36] A. S. Kaseb, W. Chen, G. Gingade, and Y.-H. Lu, "Worldview and route planning using live public cameras," in *Proceedings of the Imaging and Multimedia Analytics in a Web and Mobile World*, 2015.

[37] T. Hacker and Y.-H. Lu, "An instructional cloud-based testbed for image and video analytics," in *Proceedings of the IEEE International Conference on Cloud Computing Technology and Science*, 2014, pp. 859–862.

[38] A. S. Kaseb, A. Mohan, and Y.-H. Lu, "Cloud resource management for image and video analysis of big data from network cameras," in *Proceedings of the International Conference on Cloud Computing and Big Data*, 2015.

[39] A. Mohan, A. S. Kaseb, Y.-H. Lu, and T. J. Hacker, "Adaptive resource management for analyzing video streams from globally distributed network cameras," *IEEE Transactions on Cloud Computing*, Submitted.

[40] A. S. Kaseb, A. Mohan, Y. Koh, and Y.-H. Lu, "Cloud resource manager for analyzing big visual data from network cameras," *IEEE Transactions on Cloud Computing*, In the revision process.

[41] A. S. Kaseb, B. Fu, A. Mohan, Y.-H. Lu, and A. Reibman, "Analyzing real-time multimedia content from network cameras using cpus and gpus in the cloud," in *Proceedings of the IEEE International Conference on Multimedia and Expo*, To be submitted.

[42] D. S. Johnson, "Fast algorithms for bin packing," *Journal of Computer and System Sciences*, vol. 8, no. 3, pp. 272–314, 1974.

[43] M. R. Garey and D. S. Johnson, *Computers and Intractability: A Guide to the Theory of NP-Completeness.* W. H. Freeman & Co., 1979.

[44] D. K. Friesen and M. A. Langston, "Variable sized bin packing," *SIAM journal on computing*, vol. 15, no. 1, pp. 222–230, 1986.

[45] J. Kang and S. Park, "Algorithms for the variable sized bin packing problem," *European Journal of Operational Research*, vol. 147, no. 2, pp. 365–372, 2003.

[46] T. G. Crainic, G. Perboli, W. Rei, and R. Tadei, "Efficient lower bounds and heuristics for the variable cost and size bin packing problem," *Computers & Operations Research*, vol. 38, no. 11, pp. 1474–1482, 2011.

[47] M. Stillwell, D. Schanzenbach, F. Vivien, and H. Casanova, "Resource allocation algorithms for virtualized service hosting platforms," *Journal of Parallel and Distributed Computing*, vol. 70, no. 9, pp. 962–974, 2010.

[48] F. C. Spieksma, "A branch-and-bound algorithm for the two-dimensional vector packing problem," *Computers & operations research*, vol. 21, no. 1, pp. 19–25, 1994.

[49] C. Chekuri and S. Khanna, "On multidimensional packing problems," *SIAM journal on computing*, vol. 33, no. 4, pp. 837–851, 2004.

[50] B. Patt-Shamir and D. Rawitz, "Vector bin packing with multiple-choice," *Discrete Applied Mathematics*, vol. 160, no. 1011, pp. 1591–1600, 2012.

[51] F. Brandao and J. P. Pedroso, "Bin packing and related problems: General arc-flow formulation with graph compression," *Computers & Operations Research*, vol. 69, pp. 56 – 67, 2016.

[52] M. Gabay and S. Zaourar, "Vector bin packing with heterogeneous bins: application to the machine reassignment problem," *Annals of Operations Research*, pp. 1–34, 2015.

[53] P. KaewTraKulPong and R. Bowden, "An improved adaptive background mixture model for real-time tracking with shadow detection," in *Video-Based Surveillance Systems.* Springer US, 2002, pp. 135–144.

[54] Z. Zivkovic, "Improved adaptive gaussian mixture model for background subtraction," in *Proceedings of the International Conference on Pattern Recognition*, vol. 2, 2004, pp. 28–31.

[55] J. Shi and C. Tomasi, "Good features to track," in *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, 1994, pp. 593–600.

[56] J.-Y. Bouguet, "Pyramidal implementation of the affine lucas kanade feature tracker description of the algorithm," *Intel Corporation*, vol. 5, no. 1-10, p. 4, 2001.

[57] N. Dalal and B. Triggs, "Histograms of oriented gradients for human detection," in *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, vol. 1, 2005, pp. 886–893.

[58] G. Bradski, "The OpenCV library," *Dr. Dobb's Journal of Software Tools*, vol. 25, no. 11, pp. 120, 122–125, 2000.

[59] W. Chen, Y.-H. Lu, and T. J. Hacker, "Adaptive cloud resource allocation for analysing many video streams," in *Proceedings of the IEEE International Conference on Cloud Computing Technology and Science*, 2015.

[60] W. Chen, A. Mohan, Y.-H. Lu, T. Hacker, W. T. Ooi, and E. J. Delp, "Analysis of large-scale distributed cameras using the cloud," *IEEE Cloud Computing*, vol. 2, no. 5, pp. 54–62, 2015.

[61] S. Ren, K. He, R. Girshick, and J. Sun, "Faster r-cnn: Towards real-time object detection with region proposal networks," in *Advances in Neural Information Processing Systems*, C. Cortes, N. D. Lawrence, D. D. Lee, M. Sugiyama, and R. Garnett, Eds. Curran Associates, Inc., 2015, pp. 91–99.

VITA

VITA

Ahmed S. Kaseb has been a Ph.D. student and a Graduate Research Assistant in the School of Electrical and Computer Engineering at Purdue University. He obtained the M.S. and B.E. degrees in computer engineering from Cairo University in 2013 and 2010 respectively. He is conducting research in Big Data, Cloud Computing, and Computer Vision in order to enable cost-effective large-scale real-time analysis of image and video streams from worldwide distributed network cameras. He received the best paper award in the International Conference on Cloud Computing and Big Data 2015. His team received second prize in the Schurz Innovation Challenge in April 2014. He received the RCA Zworykin Scholarship from Purdue in 2013.