Cloud Resource Management for Analyzing Big Real-Time Visual Data from Network Cameras

Ahmed S. Kaseb, Anup Mohan, Youngsol Koh, and Yung-Hsiang Lu

Abstract—Thousands of network cameras stream real-time visual data for 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 the large quantities of data requires significant amounts of resources. These resources can be obtained from cloud vendors offering cloud instances (referred to as *instances* in this paper) with different capabilities and hourly costs. It is a challenging problem to manage cloud resources to reduce the cost for analyzing the big real-time visual data from network cameras while meeting the performance requirements. That is because the problem is affected by many factors related to the analysis programs, the cameras, and the instances. This paper proposes a cloud resource manager (referred to as *manager* in this paper) that aims at solving this problem. The manager estimates the resource requirements of analyzing the data stream from each camera, formulates the resource allocation problem as a 2D vector bin packing problem, and solves it using a heuristic algorithm. The resource manager monitors the allocated instances; it allocates more instances if needed and deallocates existing instances to reduce the cost if possible. The experiments show that the resource manager is able to reduce up to 60 percent of the overall cost. The experiments use multiple analysis programs, such as moving objects detection, feature tracking, and human detection. One experiment analyzes more than 97 million images (3.3 TB of data) from 5,310 cameras simultaneously over 24 hours using 15 Amazon EC2 instances costing $188.

Index Terms—Resource management, cloud computing, big data, computer vision, network cameras

1 Introduction

Over the past years, millions of network cameras have been deployed around the world. It is estimated that more than 28 million network cameras will be shipped in 2017, growing at a rate of 27 percent from 2012 to 2017 [1]. Su et al. [2] estimated that 10 million network cameras can produce 16,200 TB of data everyday at 1.5 Tbps. This results in an unprecedented amount of big real-time visual data. Among these millions of cameras, our team has discovered more than 120,000 *public* worldwide distributed network cameras deployed by departments of transportation, universities, individuals, etc. Retrieving one image per minute from each of these public cameras results in more than 4 TB of data everyday (assuming 40 KB per image). Moreover, some cameras provide high frame rates (e.g., 30 Frames Per Second), and some cameras have very high resolutions (e.g., $5{,}184 \times 3{,}456 \approx 18$ MP). These public cameras are streaming real-time visual data for a variety of environments as shown in Fig. 1. The visual data can be analyzed in *real-time* for many applications (e.g., traffic monitoring and surveillance).

- *A. S. Kaseb is with the Computer Engineering Department, Faculty of Engineering, Cairo University, Giza 12613, Egypt. E-mail: akaseb@eng.cu.edu.eg.*
- *A. Mohan is with Intel Corporation, Santa Clara, CA 95054. E-mail: anup.mohan@intel.com.*
- *Y. Koh and Y.-H. Lu are with the School of Electrical and Computer Engineering, Purdue University, West Lafayette, IN 47907. E-mail: {koh0, yunglu}@purdue.edu.*

Manuscript received 26 May 2016; revised 30 May 2017; accepted 11 June 2017. Date of publication 27 June 2017; date of current version 29 Nov. 2019. (Corresponding author: Ahmed S. Kaseb.) Recommended for acceptance by B. He. Digital Object Identifier no. 10.1109/TCC.2017.2720665

In order to analyze the data streams from thousands of cameras simultaneously, there is a need for a system that is able to manage significant amounts of resources (CPU, memory, etc.). These resources may be obtained using the cloud. Cloud vendors offer instances with different capabilities in terms of CPU, memory, network, etc. The instances are priced in a pay-as-you-go manner, i.e., each instance has an hourly cost and users pay when the instance is used. It is a challenging problem to manage cloud resources in order to reduce the cost for analyzing real-time data streams from thousands of cameras while meeting the performance requirements. This problem can be divided into 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. These decisions are affected by many factors, such as the analysis programs, the desired frame rates, the resolutions of the cameras, the visual content from the cameras, the types and costs of the instances, etc.

This paper proposes a resource manager that aims at reducing the overall cost of analyzing real-time data streams from network cameras while meeting the performance requirements. The overall cost is defined as the summation of the hourly costs of the used instances. The manager: (i) estimates the resource requirements for analyzing the data stream from each camera, (ii) formulates the resource allocation problem as a 2D vector bin packing

(a) Highway (b) Shopping Mall

(c) University Atrium (d) Coast

Fig. 1. Sample images from public network cameras.

problem and solves it using a heuristic algorithm, (iii) monitors the allocated cloud instances to ensure that they are not overutilized or underutilized, (iv) allocates more instances if needed and deallocates existing instances to reduce the cost if possible, and (v) migrates data streams among instances to meet the performance requirements and reduce the overall cost.

To evaluate the proposed resource manager, we integrate it into CAM^2 [3] (Continuous Analysis of Many CAMeras). CAM^2 is a cloud-based system for the large-scale analysis of the data streams from network cameras. CAM^2 has a public website [4] (http://cam2project.net) and provides access to more than 120,000 worldwide distributed network cameras. The experiments evaluate the effects of many factors (e.g., camera resolutions and instance types) on the resource management decisions. The experiments use five analysis programs to represent different workloads: image archival, motion estimation, moving objects detection, features tracking, and human detection. The experiments show that the manager is able to reduce up to 60 percent 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 Amazon EC2 [5] instances costing $188.

This paper has the following contributions:

A) It proposes a cloud resource manager that reduces the cost of analyzing real-time data streams from network cameras while meeting the performance requirements.

B) The manager 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 resolutions of the cameras, (iv) the types and costs of the cloud instances, (v) and the utilization of the running instances. The experiments evaluate the effects of these factors.

C) The manager formulates the resource allocation problem as a 2D vector bin packing problem. The experiments show the ability of the manager to reduce up to 60 percent of the overall cost.

D) The manager is integrated into CAM^2, which is, to our knowledge, the first open system for analyzing real-time big visual data from network cameras [3].

2 Related Work

2.1 Network Cameras: Sources and Analysis

There are many sources for the visual data from network cameras. Archive of Many Outdoor Scenes (AMOS) [6] 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. 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 snapshots from thousands of public cameras.

Network cameras have been used for many applications, such as surveillance [7] and weather detection [8]. Many systems have been built to analyze the visual data from network cameras. Hong et al. [9] proposed a distributed framework for spatio-temporal analysis applications on large-scale camera networks. Target Container [10] is a framework enabling users to track multiple targets in a camera network. Our system 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, and can be used for a wide range of applications.

2.2 Cloud Resource Management

Many studies have been conducted on managing cloud resources efficiently. Sun et al. [11] presented a survey of the resource management research optimizing resource utilization in data centers. They described the resource overprovisioning problem in current data centers. They also discussed different virtual machine-based, physical machine-based, and application-based resource management mechanisms. Several studies considered pushing computations to network edges in order to enhance the scalability and reduce the delay [12], [13], but this approach is not applicable in our case because we do not own or control the thousands of worldwide distributed cameras and their local networks. Thus, it is not possible moving computations to the edges.

Sharma et al. [14] proposed a system that reduces the infrastructure cost for cloud applications using integer linear programming. The system determines the types and numbers of instances based on the total number of requests, not the resource requirements of the individual requests. The system uses a dispatcher to balance the load across all application replicas. While this approach is very useful for web applications, it is not applicable in our case because the resource requirements of analyzing different data streams are different depending on the analysis program, the frame rate, and the resolution. Moreover, images from the same stream can not be arbitrarily distributed to different instances because a single data stream has to be analyzed using the same instance. That is because the instance can use historical information (e.g., background model) about the stream to detect temporal events (e.g., motion).

TABLE 1
Comparison of Our Resource Management Work

Criteria	[17]	[18]	[19]	This Paper
Cameras				
Allocate more resources for higher resolution cameras				✓
Evaluate the effect of the resolutions on the resource requirements				✓
Evaluate the effect of the camera visual content on the resource requirements				✓
Consider the cameras' locations			✓	
Analysis Programs				
Propose an allocation procedure that can handle multiple analysis programs at different frame rates				✓
Evaluate the effect of the frame rates on the resource requirements				✓
Cloud Instances				
Consider memory resources	✓			✓
Use different instance families (general purpose, compute optimized, and memory optimized)	✓			✓
Reuse running instances for newly launched analysis	✓			✓
Consider the instances' locations			✓	
Resource Monitoring and Scaling				
Scale up or down based on the actual analysis frame rate		✓		
Migrate streams among instances	✓			✓

Hossain et al. [15] aimed at reducing the number of physical machines required to host a set of virtual machines used for video surveillance services. Their solution used a linear programming formulation that does not consider that different physical machines may have different costs. Their solution is evaluated using simulations. In contrast, this paper aims at reducing the overall cost of cloud instances required to analyze a set of data streams for a variety of applications. Our solution uses a vector bin packing formulation that considers that different instances may have different costs. Our solution is evaluated using extensive experiments on Amazon EC2.

Vijayakumar et al. [16] presented a heuristic resource provisioning algorithm for data streaming applications in the cloud. They assume that: (i) Applications are CPU intensive. (ii) CPU cycles for a particular virtual machine can be changed at runtime. (iii) The cost of a virtual machine is proportional to its allocated CPU cycles. This solution is not applicable in our case because: (i) Analysis programs can be CPU intensive or memory intensive. (ii) A cloud instance of a particular type has fixed capabilities, e.g., number of cores and memory size. (iii) A cloud instance of a particular type has a fixed hourly cost. Our solution considers these factors.

Our previous work proposed resource managers reducing the cost of analyzing image and video streams from network cameras [17], [18], [19]. Anup et al. [19] formulated the resource allocation problem as a 1D variable sized bin packing problem, but this approach is not applicable in our case because it only considers CPU resources and ignores memory resources. As shown in Table 1, this paper extends our previous work on resource management in many ways. For example, it considers the heterogeneity of the camera resolutions and allocates more resources for higher resolution cameras. It also proposes an allocation procedure that can handle multiple analysis programs at different frame rates. It evaluates the effects of the camera resolutions, camera visual content, and analysis frame rates on the resource requirements. In addition, this paper conducts the largest experiment analyzing more than 97 million images (3.3 TB of data) from 5,310 cameras over 24 hours, compared with our previous experiment [17] analyzing 5.5 million images (260 GB of data) from 1,026 cameras over 6 hours.

Amazon EC2 [5] provides three related services to our proposed resource manager: (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. These services are not applicable in our case because the manager needs to: (i) monitor both the CPU and memory utilization, (ii) launch instances of different types based on the existing workloads such that the overall cost is reduced, and (iii) balance the load among the instances based on their resource utilization.

2.3 Bin Packing

In the classical bin packing problem, the goal is to pack objects of different sizes into the minimum number of unit-size bins, such that the total size of the objects in any bin does not exceed one [20]. This problem is NP-hard [21]. Several generalizations have been studied: (i) Variable Sized Bin Packing [22] allows bins to have different sizes and minimizes the overall size of the used bins. (ii) Vector Bin Packing [23], [24] uses multidimensional vectors for the size of each object or bin. The overall size of the objects in any bin in any dimension must not exceed one. (iii) Vector Bin Packing with Heterogeneous Bins [25] allows bins to have different sizes and costs. The objective is to minimize the overall cost of the used bins.

Among the different methods explained in Sections 2.2 and 2.3 (i.e., integer linear programming, variable sized bin packing, Amazon services, heuristics, etc.), we formulate the resource allocation problem as a 2D vector bin packing problem with heterogeneous bins because of several reasons: (i) Bin packing problems are known to be used for job assignment and resource allocation. (ii) 2D bin packing is used to handle the 2 resource types (i.e., CPU and memory). (iii) The heterogeneity of the bins allows the manager to support different types of cloud instances with different capabilities and costs. (iv) Bin packing problems are extensively studied in literature and existing solutions can be used. We solve the 2D vector bin packing problem using the heuristic algorithm proposed by Han et al. [25] because of

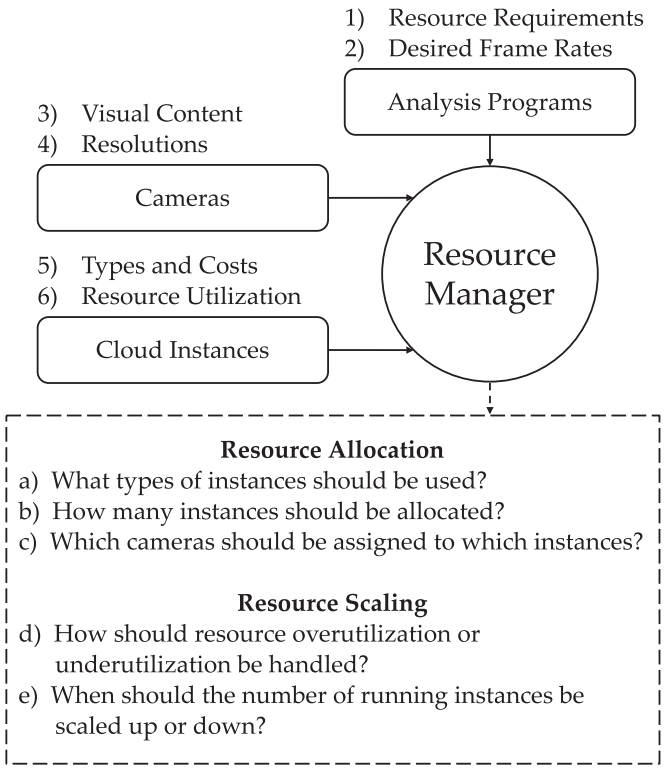


Fig. 2. The factors (1-6) affecting resource management decisions (a-e). The goal of the resource manager is to meet the performance requirements while reducing the overall cost of using the cloud.

its simplicity, speed, and accuracy. The heuristic algorithm can be 1,000 times faster than other algorithms [25]. The average solution of the heuristic algorithm deviates by about 3 percent from the optimum (when available) and by about 5.4 percent from the lower bound when the optimum is not available [25]. Alternative algorithms can be integrated into the manager without affecting its design. Instead, this would affect its speed and accuracy.

2.4 Image and Video Analysis

To evaluate the proposed manager, this paper uses existing image and video analysis techniques for background subtraction [26], [27], feature detection [28], feature tracking [29], and human detection [30]. The paper uses the corresponding OpenCV [31] implementation of each technique. This paper does not introduce new image or video analysis techniques.

3 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 1,000 traffic cameras at 10 Frames Per Second (FPS). This program requires a 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 a 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 paper investigates how to achieve this goal given that the two programs have different resource requirements and desired frame rates, the cameras have different resolutions and visual content, and the cloud instances have different types and costs.

TABLE 2
The CPU, Memory, and Hourly Cost of Different Amazon EC2 Instance Types (in Oregon)

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

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 Fig. 2, 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 a 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 unchanged. 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) *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.
4) *Resolutions*: Cameras may have different resolutions, e.g., 640×480. Analyzing streams from higher resolution cameras requires more CPU and memory resources than lower resolution cameras.
5) *Cloud Instance Types and Costs*: Cloud vendors offer dozens of instance types with different capabilities and hourly costs. For example, Table 2 shows different Amazon EC2 [5] instance types. 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 overall cost is defined as the summation of the hourly costs of the used instances. The overall cost 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. Scaling up or down the number of running instances and migrating data streams among instances may be necessary to maintain the utilization within acceptable levels (i.e., neither underutilized nor overutilized).

4 The Cloud Resource Manager

This section explains how the manager allocates and scales cloud resources in order to reduce the cost of analyzing real-time data streams from thousands of network cameras simultaneously. The manager aims at meeting the performance requirements defined in Section 4.1 and uses a two-stage resource allocation process:

1) Estimating the resource requirements of analyzing the data stream from each camera as shown in Section 4.2. This stage is performed once and used for future executions of the same analysis program.
2) Allocating cloud instances in order to reduce the overall cost as explained in Section 4.3. The overall cost is defined as the summation of the hourly costs of the used instances.

Section 4.4 shows how the resource manager monitors and scales up or down the number of running instances.

4.1 Analysis Performance Requirements

The goal of the resource manager is to execute multiple analysis programs on the data streams from different *sets of cameras* at different *desired frame rates*. Hence, we define the performance as the ratio between the actual frame rate and the desired frame rate. For a single camera c, the performance is denoted by η_c and can be calculated as

$$\eta_c = \frac{f_c}{f}, \tag{1}$$

where f_c is the actual analysis frame rate for the camera c, and f is the desired analysis frame rate. For all the cameras, the overall performance is denoted by η ($\in [0, 1]$) and can be calculated as

$$\eta = \frac{1}{N}\sum_{c=1}^{N}\frac{f_c}{f}, \tag{2}$$

where N is the total number of cameras.

The performance is affected by many factors. Some factors can not be controlled by the resource manager, such as unexpected network conditions, or cameras being accessed by other users. One factor that can be controlled by the manager is the resource utilization. If the resources are overutilized, the performance decreases significantly. The goal of the resource manager is to maintain the performance above 90 percent. Our previous experiments [17] show that maintaining the CPU and memory utilization below 90 percent leads to performance above 90 percent. Hence, the goal of the resource manager becomes allocating cloud instances such that CPU and memory utilization of any instance are below 90 percent.

4.2 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 number of data streams used in the test run is experimentally determined. 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 relationship between the frame rate and the CPU (or memory) requirements as a linear (or constant) relationship. Our experiments demonstrate these relationships later.

Camera resolutions significantly affect resource requirements. Analyzing streams from higher resolution cameras requires more resources than lower resolution cameras. The effect of the camera resolutions on the resource requirements depends on the time complexity and the space complexity of the analysis program. Because the manager assumes no prior knowledge about analysis programs, the manager repeats the test run for each unique resolution. Fortunately, there are only a few common resolutions among network cameras, such as 640×480.

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 previous work [17], [18], [32] 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 resolution 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.

TABLE 3
The Analogy Between the 2D Vector Bin Packing Problem and the Resource Allocation Problem

2D Vector Bin Packing	Resource Allocation
Each bin has a 2D size and a fixed cost.	Each instance has CPU and memory constraints and a fixed hourly cost.
Each object has a 2D size.	The analysis of each data stream requires different CPU and memory requirements.
Goal: pack all the objects into bins such that the overall cost of the bins is minimized	Goal: assign all the data streams to instances such that the overall cost of the instances is minimized

5. If there are cameras with other resolutions, repeat the test run for each unique resolution.

4.3 Resource Allocation Using Vector Bin Packing

We formulate the resource allocation problem as a 2D vector bin packing problem. Table 3 summarizes the analogy between the two problems. In the 2D vector bin packing problem [25], there are bins and objects. Each bin has a 2D size and a fixed cost. Each object has a 2D size. The goal is to select bins and pack all the objects into these bins such that the overall cost of the bins is minimized without violating the size constraints of the bins. Similarly, in the resource allocation problem, there are cloud instances and data streams. Each instance has CPU and memory constraints and a fixed hourly cost. The analysis of each data stream requires different CPU and memory requirements. The goal is to select instances and assign all the data streams to these instances such that the overall cost of the instances is minimized without violating the resource constraints of the instances.

We use the First Fit by Ordered Deviation (FFOD) heuristic algorithm proposed by Han et al. [25] to solve the 2D vector bin packing problem. The output of the algorithm 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. [25] 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.

4.4 Resource Monitoring and Scaling

It is necessary to continuously monitor resource utilization while analyzing data streams from network cameras. That is because the resource requirements may change for many reasons, such as: (i) The requirements depend on the visual content which may vary over time. (ii) The achieved camera frame rate may vary due to concurrent access from multiple users, network conditions, etc. The manager monitors the CPU and memory utilization of the allocated cloud instances to ensure that they are not overutilized (i.e., CPU or memory utilization above 90 percent) or underutilized (i.e., CPU and memory utilization below 50 percent).

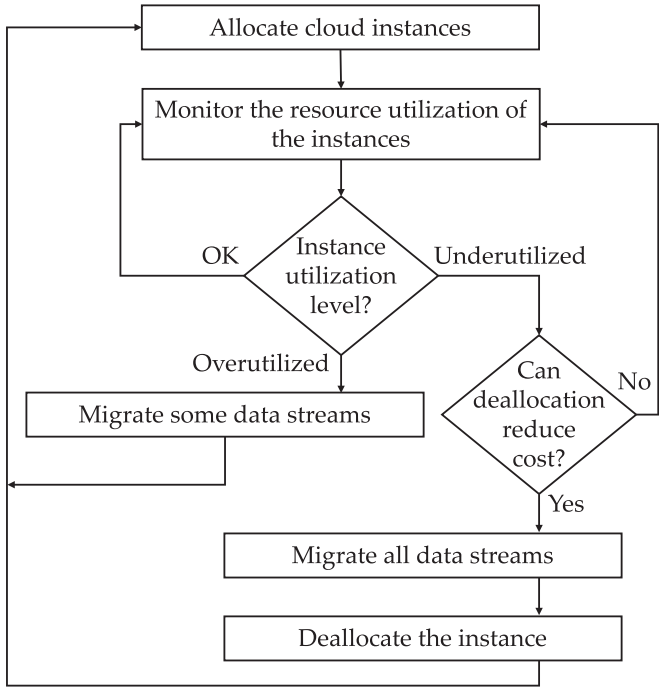


Fig. 3. The overall procedure of monitoring and scaling cloud instances.

Migrating data streams among instances is essential to maintain their utilization within acceptable levels. This migration negatively affects the analysis results' quality because: (i) There is a time gap between terminating the analysis of the data stream on the first instance and restarting the analysis on the second instance. (ii) If the analysis programs maintain temporal information such as background models, this information is lost and has to be rebuilt on the new instances. To reduce these effects, the resource manager considers deallocating only the underutilized instances. For the overutilized instances, the manager prefers migrating:

1) Streams whose images are analyzed independently so that no temporal information is lost. The manager needs users to indicate if their programs analyze images independently or not.
2) Streams being analyzed at low frame rate to reduce the effect of the migration time gap.
3) Streams being analyzed using resource intensive programs so that fewer streams are migrated.
4) Streams with high resolutions since they are more resource intensive and fewer streams are migrated.
5) Streams whose analysis started more recently not to disrupt long-running analysis.

Fig. 3 shows how the resource manager allocates more instances if needed and deallocates existing instances to reduce the cost if possible. After the manager allocates instances, it continuously monitors their resource utilization. If an instance is overutilized, the manager immediately migrates

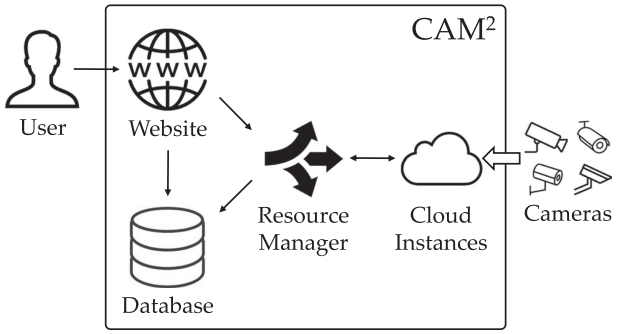


Fig. 4. The architecture of CAM^2.

some data streams from this instance. If an instance is underutilized for a period of time, and if deallocating it can reduce the overall cost, the manager migrates all the data streams currently being analyzed by this instance and deallocates it. In both the overutilization and underutilization cases, the resource manager allocates resources for the migrated data streams as described in Section 4.3.

5 Experiments

This section describes the experiments used to evaluate the proposed resource manager. Section 5.1 describes the CAM^2 system into which the manager is integrated. Section 5.2 explains the setup used in the experiments. Section 5.3 evaluates different factors considered by the manager while allocating resources. Section 5.4 evaluates the ability of the manager to reduce the cost of the allocated instances while meeting the performance requirements.

5.1 CAM^2 Analyzing the Data from Network Cameras

To evaluate the proposed resource manager, we integrate it into CAM^2. CAM^2 is a cloud-based distributed system for the large-scale analysis of the data from network cameras [3]. The goal of CAM^2 is to enable users to execute their analysis programs on the real-time images or videos from more than 120,000 online cameras. CAM^2 allows users to execute analysis programs for a variety of applications, such as counting people, detecting moving objects, tracking features, detecting weather conditions, etc.

Fig. 4 shows the architecture of CAM^2 with the following main components:

1) The website is the portal for users to select cameras and upload analysis programs.
2) The database maintains the information about the cameras, such as data formats and resolutions.
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 handles the heterogeneity of the cameras and the underlying infrastructure. CAM^2 is scalable as the visual data does not pass through a central server. Instead, the data is retrieved directly by the cloud instances as shown in Fig. 4. The website of CAM^2 enables users to: (i) view recent snapshots from the cameras through an interactive world map; (ii) select the cameras to analyze their data streams; (iii) set the execution parameters, such as the analysis duration and the frame rate; (iv) upload and execute CAM^2-compatible analysis programs; and (v) download the analysis results. Our previous work [4] provided more details on how to use the website and the API of CAM^2. The API enables analysis programs to read the input images from the cameras and save the analysis results. The API provides a uniform way to deal with all the cameras and requires only slight changes to the existing analysis programs.

CAM^2 provides access to more than 120,000 network cameras. The heterogeneity of the cameras presents many challenges for CAM^2 and its resource manager. The cameras are heterogeneous in many ways:

Types. Some cameras have public IP addresses and can provide real-time image or video streams. These cameras are referred to as IP cameras. Other cameras are available only through websites providing recent snapshots periodically.

Brands and Data Formats. Each brand (e.g., Axis, Panasonic) has a different way to retrieve the data and supports different data formats (e.g., JPEG, 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.

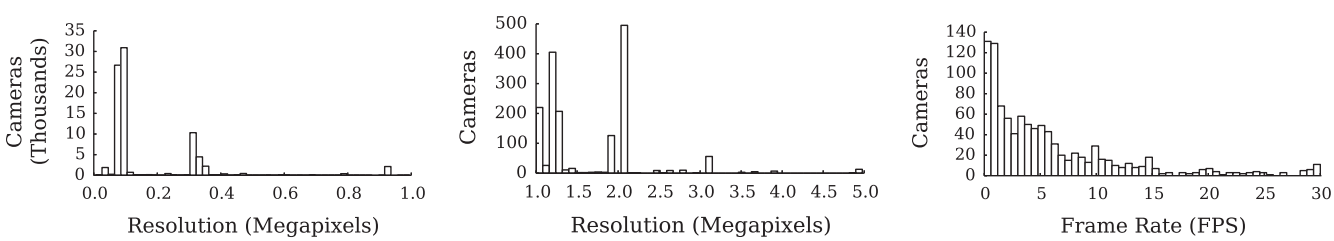
Resolutions. Figs. 5a and 5b show histograms for the camera resolutions below and above 1 Megapixels respectively. More than 1,500 cameras have resolutions above 1 Megapixels.

Frame Rates. Fig. 5c shows a histogram for the MJPEG frame rates of more than 900 IP cameras.

Visual Content. Fig. 1 shows the heterogeneity of the visual content from the cameras in CAM^2. The cameras provide a variety of scenes (e.g., highways, shopping malls) and can be used for a variety of applications.

5.2 Experimental Setup

Table 2 shows the Amazon EC2 [5] instance types used in the experiments. Table 4 shows the analysis programs used in the experiments. All the programs are implemented



(a) Resolutions Below 1 Megapixels. (b) Resolutions Above 1 Megapixels. (c) Frame Rates of more than 900 IP cameras from an instance in Oregon, USA.

Fig. 5. The heterogeneity of the camera resolutions and frame rates in CAM^2.

TABLE 4
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 [26] and estimates the amount of motion (i.e., the percentage of foreground pixels).	The foreground mask, the input image, and the amount of motion.
Moving Objects Detection	MOD	Performs background subtraction [27], 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.
Feature Tracking	FT	Detects [28] and tracks [29] image features with backtracking for verification, calculates the speed of each feature, and visualizes the tracks of the features according to their speeds.	The output 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.
Human Detection	HD	Detects humans using Histograms of Oriented Gradients [30].	The output image with the detected humans, the input image, and the number of humans.

FT is not executed at low frame rates because features can not be tracked. HD can not be executed at high frame rates due to its long execution time.

using OpenCV [31]. The programs represent different workloads in terms of CPU and memory requirements as shown later by the experiments. All the experiments meet the performance requirements as defined in Section 4.1. Unless otherwise specified, the measurements of the CPU and memory utilization are averaged over 5 minutes, and the resolution of the cameras is 640 × 480. 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 [17].

5.3 Evaluation of the Factors Affecting Resource Management Decisions

Many factors affect resource management decisions as shown in Fig. 2. This section evaluates the effect of each factor separately.

5.3.1 Resource Requirements of Analysis Programs

Fig. 6 shows that different analysis programs have different CPU and memory requirements at 0.2 FPS. 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. This experiment demonstrates the need to conduct a test run for any analysis program in order to estimate its resource requirements.

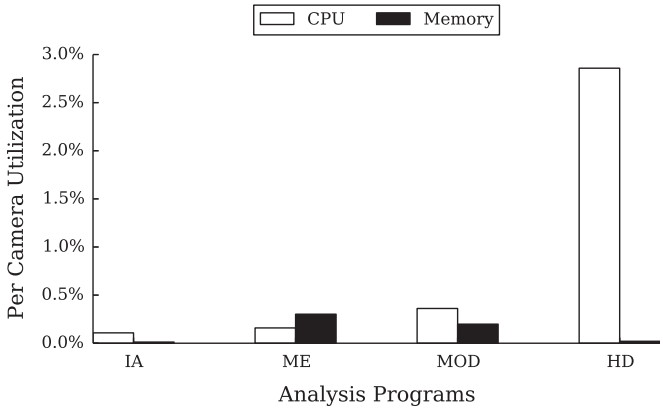


Fig. 6. Per camera CPU and memory utilization for different analysis programs at 0.2 FPS using the `m4.xlarge` instance. FT is not included as features can not be tracked at 0.2 FPS.

5.3.2 Desired Frame Rates

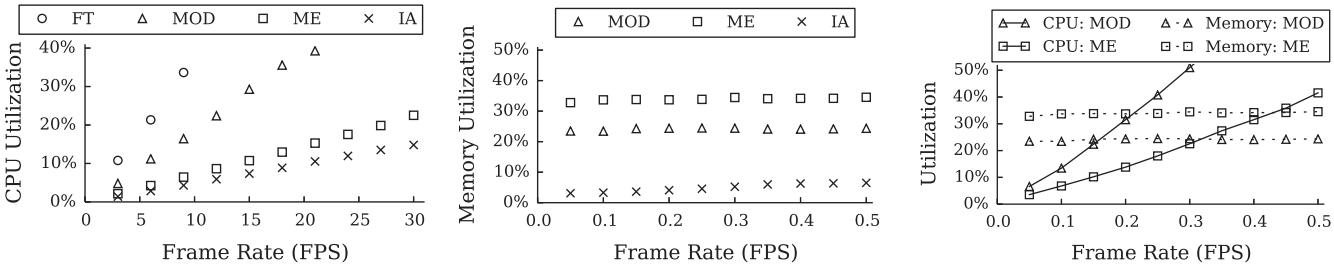
Fig. 7a shows the relationship between the frame rate and the CPU utilization. The experiment analyzes an MJPEG stream from a single camera using different analysis programs and frame rates. The resolution of the camera is 704 × 480. FT and MOD do not meet the performance requirement ($\eta > 90$ percent) 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.

Fig. 7b 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. These experiments show that 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 Fig. 7c. 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).

These experiments demonstrate the need to consider the effect of the frame rate on *both* the CPU and memory requirements of analysis programs. Thus, 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) formulates the resource allocation problem as a 2D bin packing problem to handle *both* the CPU and memory requirements.

5.3.3 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

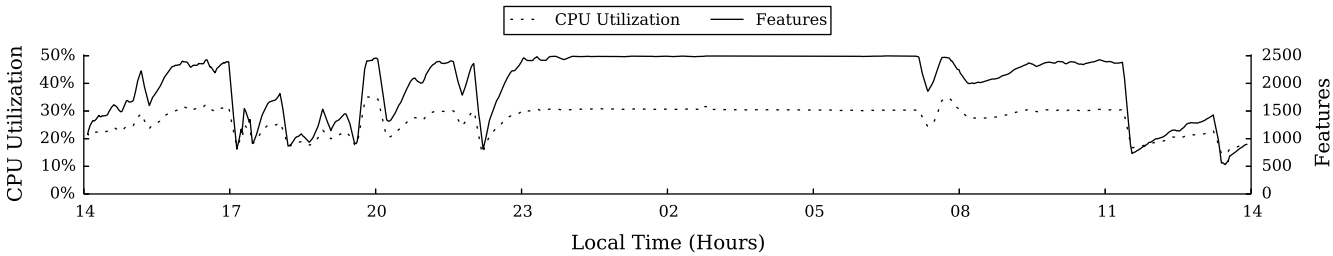


(a) MJPEG Stream from 1 Camera (b) JPEG Streams from 100 Cameras (c) JPEG Streams from 100 Cameras

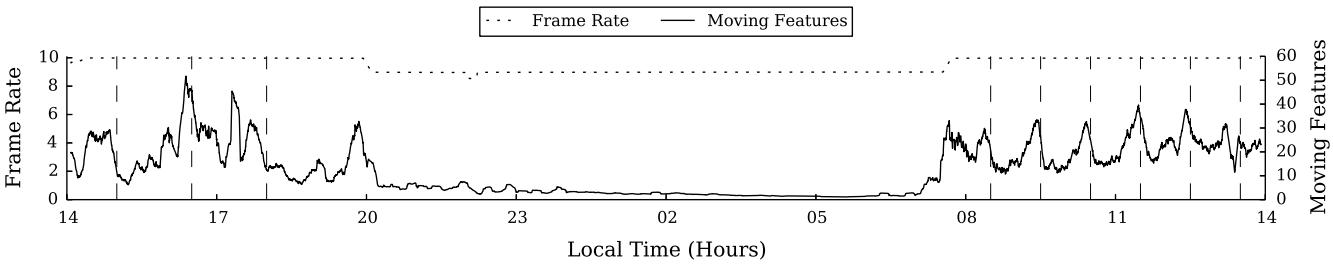
Fig. 7. The effect of the desired frame rate on both the CPU and memory utilization of an `m4.xlarge` instance. FT is not included in (b-c) as features can not be tracked at low frame rates. HD is not included in (a) as it can not be executed at high frame rates due to its long execution time. HD is not included in (b-c) as it is more CPU intensive, so it can not be executed on 100 camera streams using a single instance.

stream for 24 hours at 10 FPS. The experiment analyzes 820,000 images, more than 25 GB of data from a single camera. 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.

Fig. 8 shows the results of the experiment. Fig. 8a 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 noise but they are static, hence the number of moving features is not affected. FT also limits the number of tracked features to 2,500 to limit the execution time of the program. Fig. 8b 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 ($\eta > 90$ percent). The figure also shows that more moving features are detected during the day, especially before the lectures' starting times indicated by the vertical dashed lines. Figs. 8c, 8d, 8e, and 8f show sample output results.



(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. The system is able to meet the performance requirement by maintaining the actual frame rate close to the desired frame rate (10 FPS)

(c) At 16:25 when it is crowded (d) At 7:00 with a moving car (e) At 10:00 during lectures (f) At 10:25 during a lecture break

Fig. 8. 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) on March 24th 2016. The lines in (a) and (b) are smoothed using a moving average filter with a 10-minute window. (c-f) Sample output results. 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).

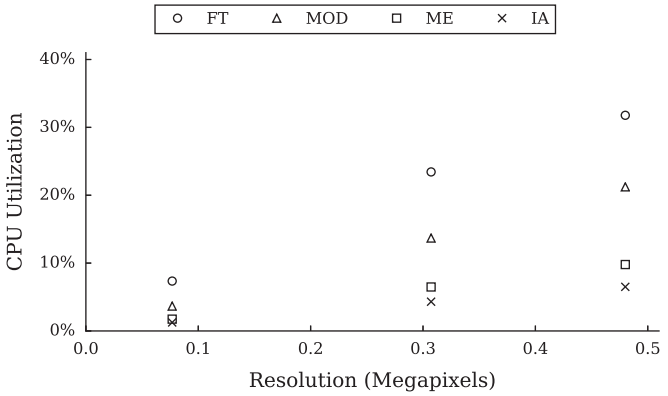


Fig. 9. The effect of the camera resolution on the CPU utilization of an `m4.xlarge` instance while executing different analysis programs at 10 FPS. HD is not included as it can not be executed at 10 FPS due to its long execution time.

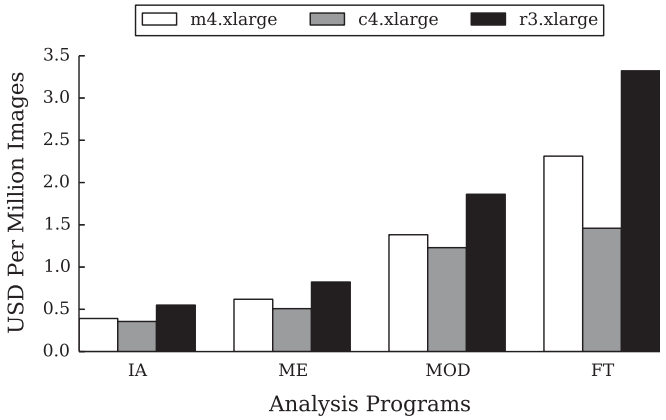


Fig. 11. Cost-effectiveness of different instance types with different analysis programs at 10 FPS. Lower is better. HD is not included as it can not be executed at 10 FPS due to its long execution time.

5.3.4 Camera Resolutions

Fig. 9 shows the effect of the camera resolution on the CPU requirement of different analysis programs. The experiment analyzes a single MJPEG stream from a single camera that supports multiple resolutions. The resolutions of the camera are 320 × 240, 640 × 480, and 800 × 600. The experiment executes different analysis programs at 10 FPS. This experiment demonstrates the need to conduct a test run for each unique resolution because the resource requirements are significantly different.

5.3.5 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. Figs. 10 and 11 show that different instance types are more cost-effective for different analysis programs. At low frame rates, as shown in Fig. 10, 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) and vice versa. At high frame rates, as shown in Fig. 11, 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.

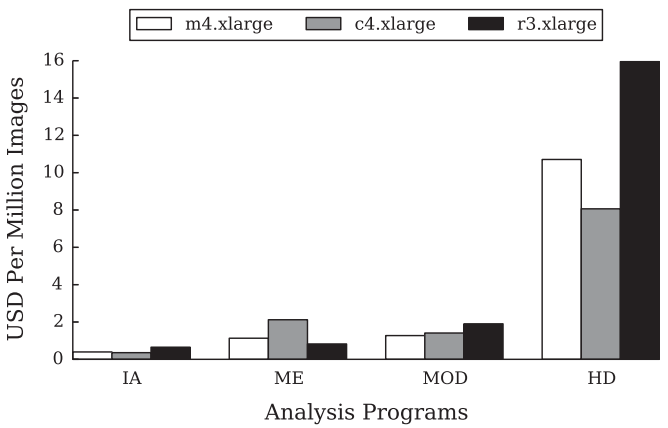


Fig. 10. Cost-effectiveness of different instance types with different analysis programs at 0.2 FPS. Lower is better. FT is not included in as features can not be tracked at 0.2 FPS.

5.4 Evaluation of Resource Allocation

5.4.1 The Heuristic Algorithm for Vector Bin Packing

The proposed manager formulates the resource allocation problem as a vector bin packing problem and solves it using the heuristic algorithm proposed by Han et al. [25]. Using 550 test problems, Han et al. [25] demonstrated that the average solution of the algorithm deviates by about 3 percent from the optimum (when available) and by about 5.4 percent from the lower bound when the optimum is not available.

In order to evaluate using the heuristic algorithm for resource allocation, we compare the results of both the heuristic algorithm and the exact method proposed by Brandao and Pedroso [33] 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. We consider the three scenarios shown in Table 5. 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 resolutions of the cameras include 640 × 480 and 1,280 × 720.

Table 6 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 percent

TABLE 5
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 6
The Overall Cost and the Types and Numbers of Instances Determined by the Manager for the Scenarios Shown in Table 5 Using Two Different Algorithms for Vector Bin Packing (Heuristic [25] and Exact [33])

Scenario	Method	Instances		Hourly Cost
		c4.xlarge	c4.2xlarge	
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

TABLE 7
The Strategies Used to Evaluate Resource Allocation

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	**This Paper:** Reduce the overall cost of the instances and allow sharing them between programs

The manager in this paper uses ST5.

TABLE 8
The Details of the CPU Intensive Scenario 1

Program	Frame Rate	Cameras	Resolutions
FT	15.00	25	640 × 480
HD	0.50	250	1,280 × 720, 640 × 480

(or 5.3 percent) 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.

5.4.2 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 7. 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 formulate 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, hence can not be solved using VPSolver. 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 9
The Types and Numbers of Instances Determined Using the Allocation Strategies Described in Table 7 to Handle Scenario 1 Shown in Table 8

	Instances			Hourly Cost	Cost Savings
	m4.x	c4.x	r3.x		
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%

All instances are xlarge. Cost savings are relative to the highest cost.

TABLE 10
The Details of the Memory Intensive Scenario 2

Program	Frame Rate	Cameras	Resolutions
ME	0.10	5,000	640×480
MOD	0.05	3,000	1920×1080, 640×480

TABLE 11
The Types and Numbers of Instances Determined Using the Allocation Strategies Described in Table 7 to Handle Scenario 2 Shown in Table 10

	Instances			Hourly Cost	Cost Savings
	m4.x	c4.x	r3.x		
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%

All instances are xlarge. Cost savings are relative to the highest cost.

Scenario 1, as described in Table 8, is CPU intensive. Table 9 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 percent compared with other strategies (i.e., ST3).

Scenario 2, as described in Table 10, is memory intensive. Table 11 shows the types and numbers of instances determined by each resource allocation strategy to handle this scenario. ST2 requires the highest number of instances since each c4.xlarge 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

TABLE 12
The Details of Scenario 3

Program	Frame Rate	Cameras	Resolutions
ME	0.20	4,000	1,280 × 720, 640 × 480
MOD	0.20	1,000	1,280 × 720, 640 × 480
FT	10.00	10	640 × 480
HD	0.20	300	1,280 × 720, 640 × 480

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 percent compared with other strategies (i.e., ST2).

Scenario 3, as described in Table 12, contains both CPU and memory intensive programs. Table 13 shows the types and numbers of instances determined by each resource allocation strategy to handle this scenario. ST1, ST2, and ST3 use 69, 91, 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 the ability of the manager (ST5) to reduce the overall cost by 24 percent compared with other strategies (i.e., ST2).

Fig. 12 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 different 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 percent of the cost of other strategies.

5.4.3 Large-Scale Experiment

In this section, we conduct the large-scale experiment specified by Scenario 3 (shown in Table 12) 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 resource 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 five. The same limit applies for `c4.2xlarge` and `r3.2xlarge` instances. In this experiment, we consider using the `2xlarge` instances (as opposed to `xlarge`) in order to conduct a large-scale experiment with fewer instances.

TABLE 13
The Types and Numbers of Instances Determined Using the Allocation Strategies Described in Table 7 to Handle Scenario 3 Shown in Table 12

	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%

All instances are `xlarge`. Cost savings are relative to the highest cost.

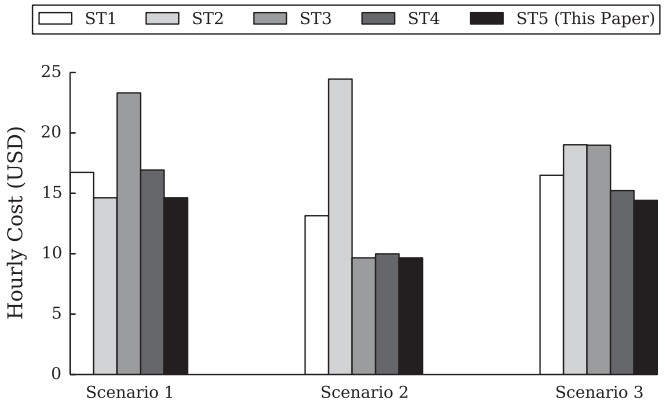
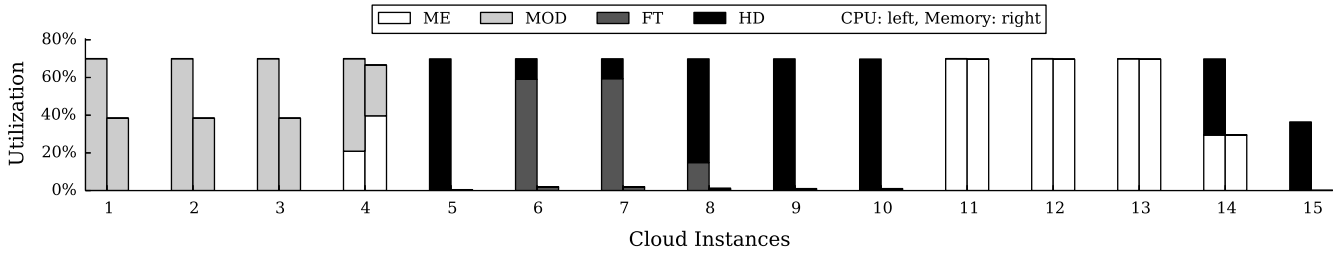


Fig. 12. The overall hourly cost of different scenarios (Tables 8, 10, and 12) using different resource allocation strategies (Table 7). This paper (i.e., ST5) reduces up to 60 percent (i.e., (24.45-9.66)/24.45 in Scenario 2) of the cost of other strategies.

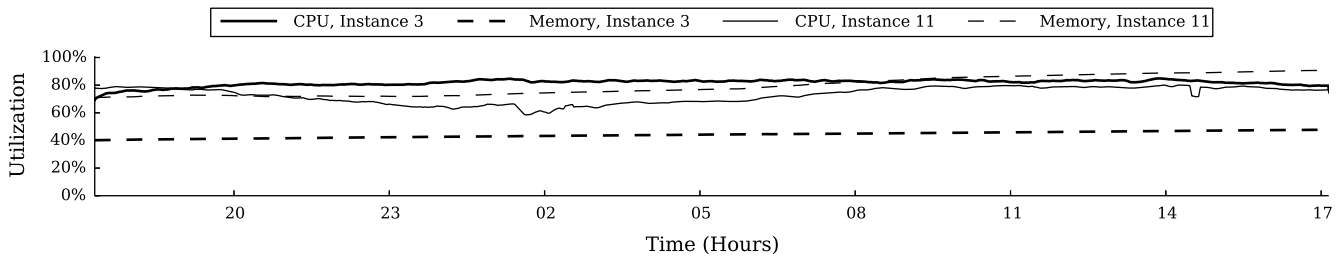
Fig. 13 shows the results of the experiment. Fig. 13a shows the estimated CPU and memory utilization of the allocated instances for each analysis program. The manager targets a 70 percent utilization for both the CPU and memory resources. That is less than the 90 percent overutilization threshold so that instances do not get overutilized easily due to the varying resource requirements of the analysis programs. The figure shows, for example, the utilization of instance 1 is 70 percent for the CPU and 40 percent 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`).

Fig. 13b 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 Fig. 13a. 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: $\eta = 98$ percent for ME, 94 percent for MOD, 94 percent for FT, and 98 percent for HD. Figs. 13c, 13d, and 13e show sample results.

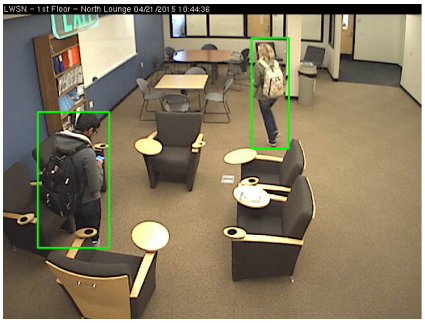
Our previous work [17] evaluated the ability of the resource manager to scale the number of running instances up or down. The experiments showed that the manager



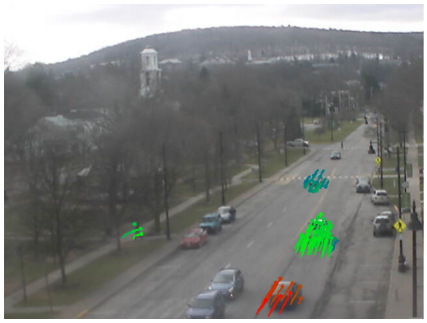
(a) The estimated CPU (left bars) and memory (right bars) utilization of each instance: (1-5) `m4.2xlarge`, (6-10) `c4.2xlarge`, and (11-15) `r3.2xlarge`. The target resource utilization is 70% to accommodate for the varying resource requirements. Bars may be split if multiple analysis programs use the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



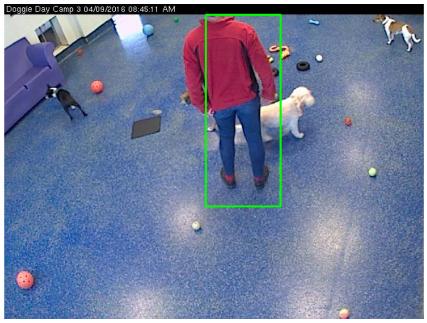
(b) The actual CPU and memory utilization of instance 3 and instance 11 from Figure 13(a) over the entire analysis duration. The lines are smoothed using a moving average filter with a 10-minute window.



(c) MOD: Two moving objects detected



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



(e) HD: One human detected

Fig. 13. 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 12). (c-e) Sample output results. (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).

reduced the cost by 13 percent due to being able to scale down the number of running instances when they are underutilized and to reuse running instances for new analysis programs. This paper improves our previous work in many ways, such as proposing a procedure that can allocate resources for multiple analysis programs at different frame rates and can consider the heterogeneity of the camera resolutions. The paper also evaluates the effects of the camera resolutions, camera visual content, and analysis frame rates on the resource requirements. In addition, this paper conducts the largest experiment analyzing more than 97 million images (3.3 TB of data) from 5,310 cameras over 24 hours.

6 Conclusion

This paper proposes a cloud resource manager that reduces the cost for analyzing real-time data streams from thousands of network cameras while meeting the performance requirements. The manager allocates cloud instances based on many factors, including the analysis programs, the desired frame rates, the camera resolutions, and the types and costs of the instances. The manager formulates the resource allocation problem as a 2D vector bin packing problem and solves it using a heuristic algorithm. The resource manager monitors the allocated instances; it allocates more instances if needed and deallocates existing instances to reduce the cost if possible. The experiments show that the resource manager is able to reduce up to 60 percent 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.

Acknowledgments

The authors would like to thank Amazon for providing the cloud infrastructure, and the organizations that provide the camera data. A complete list of the data sources is available at https://cam2.ecn.purdue.edu/acknowledgements. This project is supported in part by US National Science Foundation ACI-1535108, CNS-0958487, and OISE-1427808. Any opinions, findings, and conclusions or recommendations expressed in this material are those of the authors and do not necessarily reflect the views of the sponsors.

REFERENCES

[1] Network camera and video analytics market. (2012). [Online]. Available: http://www.marketsandmarkets.com/Market-Reports/visual-communication-market-775. html

[2] W.-T. Su, Y.-H. Lu, and A. S. Kaseb, "Harvest the information from multimedia big data in global camera networks," in *Proc. IEEE Int. Conf. Multimedia Big Data*, 2015, pp. 184–191.

[3] A. S. Kaseb, et al., "A system for large-scale analysis of distributed cameras," in *Proc. IEEE Global Conf. Signal Inf. Process.*, 2014, pp. 340–344.

[4] A. S. Kaseb, et al., "An interactive web-based system using cloud for large-scale visual analytics," in *Proc. Imag. Multimedia Analytics A Web Mobile World*, 2015, Art. no. 94080E.

[5] Amazon EC2. [Online]. Available: https://aws.amazon.com/ec2/, last accessed in 2017.

[6] N. Jacobs, N. Roman, and R. Pless, "Consistent temporal variations in many outdoor scenes," in *Proc. IEEE Conf. Comput. Vis. Pattern Recognit.*, 2007, pp. 1–6.

[7] S. Srivastava and E. J. Delp, "Video-based real-time surveillance of vehicles," *J. Electron. Imaging*, vol. 22, no. 4, 2013, Art. no. 041103.

[8] Z. Chen, F. Yang, A. Lindner, G. Barrenetxea, and M. Vetterli, "Howis the weather: Automatic inference from images," in *Proc. IEEE Int. Conf. Image Process.*, 2012, pp. 1853–1856.

[9] K. Hong, M. Voelz, V. Govindaraju, B. Jayaraman, and U. Ramachandran, "A distributed framework for spatio-temporal analysis on large-scale camera networks," in *Proc. IEEE Int. Conf. Distrib. Comput. Syst. Workshops*, 2013, pp. 309–314.

[10] 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 *Proc. ACM/IEEE Int. Conf. Distrib. Smart Cameras*, 2011, pp. 1–8.

[11] X. Sun, N. Ansari, and R. Wang, "Optimizing resource utilization of a data center," *IEEE Commun. Surveys Tuts.*, vol. 18, no. 4, pp. 2822–2846, Oct.–Dec. 2016.

[12] X. Sun and N. Ansari, "PRIMAL: Profit maximization avatar placement for mobile edge computing," in *Proc. IEEE Int. Conf. Commun.*, 2016, pp. 1–6.

[13] X. Sun and N. Ansari, "EdgeIoT: Mobile edge computing for the internet of things," *IEEE Commun. Mag.*, vol. 54, no. 12, pp. 22–29, Dec. 2016.

[14] U. Sharma, P. Shenoy, S. Sahu, and A. Shaikh, "A cost-aware elasticity provisioning system for the cloud," in *Proc. Int. Conf. Distrib. Comput. Syst.*, Jun. 2011, pp. 559–570.

[15] M. S. Hossain, M. M. Hassan, M. A. Qurishi, and A. Alghamdi, "Resource allocation for service composition in cloud-based video surveillance platform," in *Proc. IEEE Int. Conf. Multimedia Expo Workshops*, 2012, pp. 408–412.

[16] S. Vijayakumar, Q. Zhu, and G. Agrawal, "Dynamic resource provisioning for data streaming applications in a cloud environment," in *Proc. IEEE Int. Conf. Cloud Comput. Technol. Sci.*, 2010, pp. 441–448.

[17] A. S. Kaseb, A. Mohan, and Y.-H. Lu, "Cloud resource management for image and video analysis of big data from network cameras," in *Proc. Int. Conf. Cloud Comput. Big Data*, 2015, pp. 287–294.

[18] W. Chen, Y.-H. Lu, and T. J. Hacker, "Adaptive cloud resource allocation for analysing many video streams," in *Proc. IEEE Int. Conf. Cloud Comput. Technol. Sci.*, 2015, pp. 17–24.

[19] A. Mohan, A. S. Kaseb, Y. H. Lu, and T. J. Hacker, "Location based cloud resource management for analyzing real-time videos from globally distributed network cameras," in *Proc. IEEE Int. Conf. Cloud Comput. Technol. Sci.*, 2016, pp. 176–183.

[20] D. S. Johnson, "Fast algorithms for bin packing," *J. Comput. Syst. Sci.*, vol. 8, no. 3, pp. 272–314, 1974.

[21] M. R. Garey and D. S. Johnson, *Computers and Intractability: A Guide to the Theory of NP-Completeness*. San Francisco, CA, USA: Freeman, 1979.

[22] D. K. Friesen and M. A. Langston, "Variable sized bin packing," *SIAM J. Comput.*, vol. 15, no. 1, pp. 222–230, 1986.

[23] F. C. Spieksma, "A branch-and-bound algorithm for the two-dimensional vector packing problem," *Comput. Operations Res.*, vol. 21, no. 1, pp. 19–25, 1994.

[24] C. Chekuri and S. Khanna, "On multidimensional packing problems," *SIAM J. Comput.*, vol. 33, no. 4, pp. 837–851, 2004.

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

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

[27] Z. Zivkovic, "Improved adaptive gaussian mixture model for background subtraction," in *Proc. Int. Conf. Pattern Recognit.*, 2004, pp. 28–31.

[28] J. Shi and C. Tomasi, "Good features to track," in *Proc. IEEE Conf. Comput. Vis. Pattern Recognit.*, 1994, pp. 593–600.

[29] J.-Y. Bouguet, "Pyramidal implementation of the affine Lucas Kanade feature tracker description of the algorithm," *Intel Corporation*, vol. 5, no. 1–10, 2001, Art. no. 4.

[30] N. Dalal and B. Triggs, "Histograms of oriented gradients for human detection," in *Proc. IEEE Conf. Comput. Vis. Pattern Recognit.*, 2005, pp. 886–893.

[31] G. Bradski, "The OpenCV library," *Dr. Dobb's J. Softw. Tools*, vol. 25, no. 11, pp. 120, 122–125, 2000.

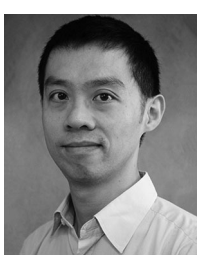
[32] W. Chen, A. Mohan, Y.-H. Lu, T. Hacker, W. T. Ooi, and E. J. Delp, "Analysis of large-scale distributed cameras using the cloud," *IEEE Cloud Comput.*, vol. 2, no. 5, pp. 54–62, Sep./Oct. 2015.

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

Ahmed S. Kaseb received the BE and MS degrees in computer engineering from Cairo University, in 2010 and 2013, respectively, and the PhD degree in computer engineering from Purdue University, in 2016. He is an assistant professor of computer engineering in the Faculty of Engineering, Cairo University. He is conducting research in big data, cloud computing, and computer vision. He received the best paper award in the International Conference on Cloud Computing and Big Data 2015.

Anup Mohan received the PhD degree from the School of Electrical and Computer Engineering, Purdue University, in 2017. His research interests include large-scale video analysis, cloud computing, and big data analysis. He is currently working with Intel Corporation, Santa Clara.

Youngsol Koh received the BS degree in electrical engineering from Purdue University, West Lafayette, in 2012. He is working toward the MS degree in computer engineering from Purdue University. His research interests include big data and cloud computing to enable large-scale archived image and video processing in cost effective ways.



Yung-Hsiang Lu received the PhD degree from Stanford University. He is a professor in the School of Electrical and Computer Engineering, Purdue University. He is an ACM distinguished scientist and ACM distinguished speaker. He is the lead investigator of a software system for analyzing thousands of network cameras worldwide. This cloud-based system can retrieve and analyze real-time visual data. More than 200 people have signed up as users. He is the lead organizer of the first Low-Power Image Recognition Challenge, in 2015.