IEEE International Conference on Cloud Computing Technology and Science (CloudCom) 2015

Adaptive Cloud Resource Allocation for Analysing Many Video Streams

Wenyi Chen, Yung-Hsiang Lu, Thomas J. Hacker
Purdue University, West Lafayette, IN, USA
{chen345, yunglu, tjhacker}@purdue.edu

***Abstract*—The visual data generated by network cameras can be valuable for a wide range of scientific studies such as weather, wildlife, and traffic. The resource demands for analysis of the data may fluctuate significantly for some of these studies (for example, seasonal or during only rush hours). Cloud computing's pay-per-use can be a preferred solution for analysing large amounts of data from these network cameras. There are few studies focusing on how to allocate cloud resources to analyse many video streams from network cameras. Existing autoscaling and load balancing methods are inapplicable due to the nature of the workloads. This paper presents a solution to manage cloud resources analysing multiple video streams. When an analysis program is launched, the resource manager adjusts the resource allocation by one of two methods: gradually scaling up the number of instances or predicting the needed number of instances. Then, the resource manager monitors the utilization and scales the resources in response to the analysis program's loads. The proposed resource manager has been implemented using Microsoft Azure and demonstrated that the system can adaptively adjust resource to cater to the workload. We explore the trade-offs inherent to both methods. Gradually scaling up can reach more efficient resource allocation, but may take multiple trials. The prediction method is faster in making allocation decisions, but may need additional adjustment. We also investigate the impact of the types of virtual machines and find that smaller sizes are usually better in terms of performance-cost ratios.**

I. Introduction

Globally deployed network cameras provide valuable information for understanding the world [6], for example, traffic, wildlife, and weather. Some of these studies can span in time from seasonal effects to only a few hours per day. For these studies, pay-per-use cloud computing can be a preferred solution for provisioning the computational and storage resources for analysis. These studies may store data for offline analysis, or retrieve live data for online analysis. In both scenarios, it is necessary to allocate sufficient resources to meet the desired performance requirements. For video streams, high frame rates are one of the most important requirements. Many analysis programs (such as counting moving vehicles on streets) rely on the availability of high frame rates. Many techniques for dynamic resource management have been developed and widely used, for example, the Amazon Web Services auto-scaling feature [2]. However, these techniques serve only web applications. Research [1] [14] [5] on streaming applications with cloud instances usually focuses on single data stream.

In recent years, cloud computing has become a popular option for processing large amounts of data and web services. Cloud computing is flexible in resource allocation and customers can pay based on the usage of computing resources. Meanwhile, many scientific studies use streaming data for observation and analysis. For example, Winkler et al. [15] use webcams to detect foam to measure water quality. Purser and Thomsen [12] suggest using cameras to monitor the impacts on coral reefs affected by drilling for gas and oil. These projects have something in common: the amount of streaming data is large and the needs for computational resources are heavy. The analysis may last for days or months, but the required frame rate may not be very high. Some other studies, such as tracking objects, observing animals, and monitoring volcanos [7], [11], [13] require high frame rates. Because of the wide range of the amount of computational resource needs, cloud computing can be a good option for analysing the streaming data.

Using cloud computing to analyse streaming data requires the solution to some challenging problems: streaming data are highly unpredictable in data rates because of the content, the network condition, the execution time of the analysis programs, etc. The system should be able to cater to quick changes of the flow of the streaming data and adjust resource accordingly. Also, the resource allocation depends on the kind of data streams and the analysis programs. Thus, an adaptive solution must be developed for determining the types and the amounts of resources. Finally, the streaming process may last a very long time, for example, an entire season, depending on the users' requirements. Even though cloud computing can be an inexpensive option for occasional resource needs, when an analysis program executes for an extended period of time, the cost can accumulate quickly. The solution presented in this paper considers how to meet the performance requirements while keeping the costs low.

This paper presents two solutions for resource allocation for analysing large-scale streaming data. Both solutions start with test runs to gather information about the streams and the analysis programs. The first solution increments the number of cloud instances (virtual machines, VMs) gradually until finding the appropriate number of VMs. The second solution uses the performance of one single VM to predict the number of VMs needed. During execution, both solutions monitor the VMs' utilization and adaptively provision resource for meeting performance requirements and reducing cost over a long period of time (several days). We evaluate the solutions and the adaptive way using 286 network cameras located across North America streaming images and running image analysis programs. We also compare the performance-cost

difference of different types of VMs and discover that VMs with fewer cores is more cost-effective.

The main contributions of this paper include: (1) this is one of the first solving the problem of dynamic resource allocation for analysing large-scale streaming visual data; (2) this paper uses real streaming data to evaluate the system and provides concrete data for the performance-cost trade-off and the proposed solution has been implemented using VMs from a cloud vendor; (3) this paper presents easy-to-implement solutions and guidelines for choosing the number and the type of VMs.

Experiments demonstrate that our system can maintain the required frame rate for analysing hundreds of video streams for several days at a low cost.

II. Related Work

Resource management for cloud services is not a new problem. In fact, some cloud vendors provide resource management feature for the users. For example, Amazon Web Services (AWS) provides an auto-scaling feature [2] that allows automatically scaling up or down the number of VMs to cater to changes of network traffic. However, this feature is for web services, not for large-scale streaming analysis. For web services, each request has its own session with a relatively short lifespan. Thus, the number of sessions or threads changes frequently. Resource management strategies in this case mainly focus on the threads management and distribution of threads. For large scale streaming data, the number of session is fixed, and the session may last for many days depending on the application. The network data rates (Mbps) may not change much, but the content of the streams may change substantially, resulting in changing execution time. The auto-scaling feature cannot take those factors into consideration.

Cloud computing is widely used for streaming applications serving video on demand. Alasaad et. al. [1] investigate the prediction of resource allocation for streaming services to keep the cost low. Their strategy is based on the non-linear pricing policy of Amazon EC2. Vijaykumar et. al. [14] design a self-adaptive system to allocate enough CPU resources to make sure the processing rate matches the arrival rate of a single stream. Their system aims to meet the real-time processing performance when data arrival rate abruptly changes. De Cicco et al. [5] develop a resource controller for adaptive video streaming. Their system serves only video streaming without any processing. Also, their work does not consider the cost of the system.

All of the prior work considers a single stream or only a few streams. Moreover, these studies use only simulation to evaluate their designs or strategies. This paper presents our two solutions to allocate appropriate amounts of resources and to maintain performance while keeping the costs low. Our solutions allow users to analyse hundreds of video streams and to run the analysis programs for multiple days. In this paper, we evaluate our system using 286 network cameras and run the system for three days. Our previous work developed a web-based system for analysing large-scale distributed cameras [8] [9]. The existing system has no adaptive resource management. The solutions presented in this paper will be integrated into the system.

III. Resource Management

A. Problem Description

The goal of resource management is to maintain the required performance while making the cost as low as possible. To achieve this goal, we need a way to quantitatively measure performance. Performance metrics can change for different applications and purposes. For this paper, we use the number of frames that can be processed per second to represent performance. The processing time is specified by the user and the cost is determined by the number of VMs and the unit price of the VM. The unit price of a VM is mainly determined by its number of virtual CPUs (vCPUs), the amount of memory, and the geographical location. The main challenges are what type of VMs to allocate, where to allocate them, and how many VMs to allocate. In our previous work [4], we quantitatively measured the effect of the type of VMs. The result can be seen in Figure 1. The experiment is done by running a background subtraction program on many network cameras and recording the frame rates.

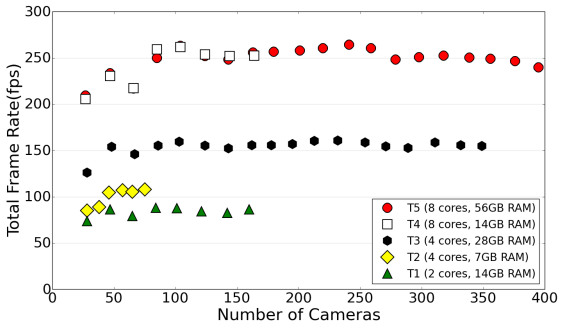


Fig. 1. The effects of types of VMs on the total frame rates [4]. The numbers of cores affect the frame rates and the amounts of memory affect the maximum number of cameras whose data can be processed by a VM. When the total frame rate remains unchanged and more cameras are added, the frame rate per camera decreases.

In this figure, the x-axis represents the number of cameras and the y-axis represents the total frame rate. We use five different types of VMs, denoted as T1, T2, T3, T4, and T5. Their numbers of cores and amounts of memory are specified in the legend. From the figure, we can see the effect of the number of cameras, the number of cores and the amount of memory on the measured frame rate. When the number of cameras is large, the frame rate stays constant. This means the VM resources are saturated and increasing the number of cameras will not raise the total frame rate. This is called the *saturation frame rate* in this paper. When heavy computation is involved, the VM will reach the saturation frame rate quickly (with fewer than 30 cameras). The saturation frame

rate is directly related to the number of cores of the VM. For example, T4 and T5 have the same saturation frame rate, although T5's memory capacity is much larger. This experimental result suggests that the amount of memory does not affect the total frame rate. Instead, the memory capacity determines the maximum number of cameras that can be processed at the same time. For example, T4 runs out of memory at 160 cameras and T5 can process the data from 400 cameras.

To summarize, the frame rate is affected by the number of cameras, the number of cores and the amount of memory. Once a VM reaches the saturation frame rate, the number of cameras have negligible effect on the frame rate. If a VM has enough memory, then the most important factor is the number of cores. The rest of this paper focuses on determining the VMs' numbers of cores and the numbers of VMs, while reducing the costs.

B. Problem Formulation

The total frame rate, F, is the summation of the frame rates of all VMs. Our primary goal is to maintain F larger than the required frame rate specified by the user, F_r. Let f represents the frame rate of a single VM. As discussed above, f is a function of the number of cameras, the number of cores and the amount of memory. Furthermore, when the number of cameras is large, it does not affect the frame rate because the frame rate reaches the saturation frame rate. Also, the amount of memory only determines the maximum number of cameras that can be processed at the same time. Because our system deals with a large number of cameras, the frame rate almost always reaches the saturation frame rate. Thus, if we assume the memory is sufficient for analysis, f is a function of the number of cores of the VM. We denote the number of cores of a single VM as c, then we can denote this function as $g(c)$. We denote the number of VMs we used as N. Then, the relationship among f, F and F_r can be expressed as:

$$f = g(c) \tag{1}$$

$$F = \sum_{i=1}^{N} f_i \tag{2}$$

$$F \geq F_r \tag{3}$$

The total cost, P, depends on the pricing policy of the cloud vendor. In this paper, we will use the pricing policy of Microsoft Azure. In this pricing policy, the cost is proportional to the total number of cores, which is $N \times c$. This means using many small instances has the same cost as using a single large instance as long as the total number of cores is the same. Our problem can be formulated as:

minimize P

$F \geq F_r$

given that

$P \propto N \times c$

and restrictions from equation (1) - (3)

Looking at Figure 1 closely, we can find that f is not proportional to c. Comparing the data points of T5 and T3, we can see that the relationship between c and f is sublinear. When c is doubled, f is less than doubled. This means when c increases, the f-c ratio decreases. The same observation can be obtained by comparing data points of T3 and T1. This suggests that using many small instances may yield better performance than using a single large instance. In order to verify this, we try different type of VMs for the system in evaluation section and find that indeed small instances have better performance. Since the instances are of the same type, and the frame rate of each instance is usually the saturation frame rate, the frame rate from each VM is approximately the same. Then, if the type of VM is determined, our problem becomes finding N. In order to find N that is large enough to reach required frame rate while as small as possible, we have developed different strategies.

TABLE I
SYMBOLS USED IN THE PAPER

Symbol	Meaning
N	number of VM
c	number of cores per VM
f	frame rate for a VM
F	total frame rate
F_r	required frame rate
P	total cost
$g(c)$	the function whose input is the number of cores and the output is the frame rate

C. Test Run and Workload Balance

We assume no prior knowledge about the analysis programs. Hence, our methods obtain essential parameters about the programs by executing them first as "test runs". We run the programs on one VM for a short period of time to determine the average frame rate in this period. Another problem is to balance the workload for each VM. Although the cameras are different in many aspects, including network condition, resolution and content, when the number of cameras becomes large, the differences smooth out. Thus, when assigning cameras to VMs, we randomly pick cameras from the camera pool and assign the same number of cameras to each VM.

Even if we have some knowledge about the cameras, random assignment is still an efficient way for camera distribution. Since the content and network condition of the camera is impossible to predict, the only information we can obtain from a network cameras is the resolution of the camera and the round-trip time (RTT) between the camera and the VM instance. In order to examine the effect of those two factors, we sort 286 IP cameras by their RTT in the ascending order. Then, we use the shortest 10%, the medium 10% and the longest 10% to perform test runs. We do the same for resolution. The results are shown in Table II

TABLE II
FRAME RATE BY CHOOSING CAMERAS BY DIFFERENT PARAMETERS

RTT	Total frame rate(fps)	Resolution	Total frame rate(fps)
shortest 10%	28.66	smallest 10%	58.36
medium 10%	24.61	medium 10%	36.72
longest 10%	26.63	largest 10%	8.94

From Table II, we can see that RTT has negligible effect on the frame rate. This is because RTT mainly reflects the latency of the network, but when heavy analysis is involved, the network is not the bottleneck of the frame rate. The resolution has significant effect on the frame rate. A higher resolution leads to a lower frame rate. This is because a high resolution stream requires more time to process, thus the workload is heavier compared with a low resolution stream. Obtaining the resolution of the cameras is time-consuming. It takes about 15 minutes for us to obtain the resolution of 286 cameras. This is because we need to establish and terminate the connection with each camera. Also, because of unreliable nature of public network cameras, only 181 cameras responded when we try to obtain resolutions and 113 cameras responded when we try to obtain RTT. The relationship between resolution and execution time depends on the analysis program. For some programs, the relationship may be polynomial. For some other programs, the relationship may be exponential. Without any knowledge of the analysis program, we can not relate resolution to the execution time directly. In that case, using resolution for workload distribution is not a plausible solution.

Our system contains two phases. In the first phase we determine how many VMs to allocate in the beginning. In the second phase, we use the adaptive system to monitor the frame rate periodically and use the information as feedback to adjust resource allocation. We will explain the strategies and design of the two phases next.

D. Linear Increment Method

The first method used for the first phase is a linear increment method. First, we launch one VM, and perform a test run, and examine the frame rate of the test run. If the frame rate is lower than the required frame rate, we launch one more VM, and perform another test run. Otherwise, we use the number of VM used in last test run as the number of required VMs. We repeat this process until the frame rate reaches the requirement. In the first few test runs of the method, cameras per VM may be too high and the program will crash. This will result in a frame rate of zero. In that case, the system will continue launching more instances until a valid frame rate is obtained. This method is called *linear increasing* because the number of VMs is incremented linearly. By gradually launching VMs and measuring the total frame rate, this method can find the suitable number of VMs. This method is resilient to the abrupt changes in network conditions or content of cameras during the test runs. Even some test runs gives incorrect results, this method can still find the suitable number of VMs. However, the downside of this method is that the time for finding N is proportional to the number of VMs needed. Thus, it may take many trials to find N. Each trial involves launching a new VM, which may take a few minutes. This means we need more time before the system can start real analysis of visual data.

E. Predictive Method

The second method for the first phase uses a predictive model. This method is based on our observations. Because of the heavy computation of image analysis programs and the large number of cameras, the frame rate from each VM is approximately the same at saturation frame rate. Thus, equation (2) can be converted to $F = N \times f$. In this case, if we can predict f (the saturation frame rate), then we can calculate N using F_r as the total frame rate. In order to find f, we need to use the VM to process the data from a certain number of cameras. Because of the memory limit of the VM, we can not process the data from too many cameras for the test run. Otherwise, the VM would simply crash without gleaning useful information. In contrast, if we use too few cameras, then the instance will not reach the saturation frame rate. To avoid these situations, we choose the number of cameras so that the utilization of VMs is high but the VM will not crash. For this paper, we use 30 cameras for the test run. We choose the cameras randomly from the camera pool. After the test run, we use the measured frame rate as a prediction of f. Then, N can be calculated by:

$$N = \left\lceil \frac{F_r}{f} \right\rceil \tag{4}$$

We take the ceiling to ensure that the frame rate is higher than the required frame rate. The advantage of this method is that the time to find N is constant, thus the time for testing is shorter than the linear increment method. Also, since only one test run on one VM is needed, the cost is smaller than the linear increment method. We launch all of the VMs in parallel, thus the time for launching VMs is much shorter than the linear increment method. The disadvantage of this method is that the prediction may be wrong and we will not get the correct number of VMs needed. This would happen if the cameras we used for test run are heavily biased. For example, if the program is tracking motions in the cameras, and we happen to select some cameras with no or little motion at the time of testing, then the frame rate from testing will be higher than average, thus the prediction of f may be incorrect. Also, this method may overestimate N. This would happen if the computation is light or the number of camera is small. In this case, the f we get from the test run may be smaller than the saturation frame rate, and then make N bigger than necessary. These disadvantages can all be solved by the adaptive allocation explained next. Table III summarizes and compare the advantages and disadvantages of the two methods.

F. Adaptive Resource Allocation

The above two methods determine the initial number of necessary VMs. Depending on the analysis program and the

TABLE III
ADVATANGES AND DISADVANTAGES OF THE TWO METHODS

Method	Advantages	Disadavantages
Linear increment	less sensitive to changes of network condition	takes more time and cost more
Predictive	fast and cheap	the output may be wrong

content of the data, the frame rate may change over time. Consider an analysis program counting the number of people on a street of a business district. The street are crowded during business hours and almost empty after business hours. We further assume that the program's execution time increases when more people are detected (this is common for many people detection methods). In this case, the program's execution time is longer during daytime and shorter at night. In other words, the frame rates are lower at daytime and higher at night. Our solution is designed for running analysis programs over long periods of time. Thus, our resource manager has to consider the changes in programs' execution time. The number of VMs would need to be changed to make the system adaptive. Also, as mentioned above, the predictive method may overestimate or underestimate the number of VMs needed, thus our system needs to be able to detect these and adjust accordingly. After we determine N using either method mentioned above, we partition the camera list so each VM can get approximately the same number of cameras. After a period of time, we examine the achieved frame rate. If the duration is too short, it may interfere with the analysis a lot and reduce the efficiency of the whole system. If the duration is too long, the system will react slowly to the changes in cameras or network conditions. In this paper, we set the duration to 20 minutes to balance the above two factors. If the frame rate is so high that we can achieve the required frame rate with fewer instances, we reduce the number of VMs. If the frame rate is below the required frame rate, we launch more VMs. If the number of VMs changes, we randomly re-assign cameras to the VMs to make sure the workload is balanced. This ensures the system would always use the appropriate number of VMs depending on the current condition.

Adaptive resource allocation must solve two important problems: (1) determine the thresholds for terminating or launching VMs and (2) determine the number of VMs to launch or to terminate. The answer for (1) can be divided into two parts: threshold for launching VMs and threshold for terminating VMs. The threshold for launching is simpler, if the frame rate falls below the required frame rate, we should launch more VMs. The threshold for terminating is more complex. The threshold should be the frame rate that is high enough so that the frame rate is still above the required frame rate even after we terminate some instances. The answer for (2) is also tricky. A simple solution is to terminate only one or launch only one VM at one time and eventually we will get to the right number of VMs. This would make the system react slowly to abrupt changes. Thus, we need to determine the number of VMs to launch or terminate.

Our solution solves the two problems together. We do not calculate thresholds or determine how many VMs to launch or to terminate. We calculate the number of VMs needed based on the most recent frame rate. Then, if the number is different from the number we have right now, we adjust VMs. In order to calculate how many VMs are needed, we reuse the thinking in predictive method and assume that the total frame rate, F, is proportional to the number of VMs. Thus, we can calculate f first:

$$f = \frac{F}{N} \tag{5}$$

Then, we can determine the number of VMs needed using Equation (4). If the number of VMs needed is different from the number of VMs we have, then we know we need to adjust the number of VMs.

When the number of VMs needs to be changed, we need to re-assign cameras to VMs. Here we randomly assigned cameras each time to VMs. When re-assignment occurs, we need to pause the analysis and may need to migrate processed data to another VM. During migration, streaming data may be lost. Migrating data streams between VMs without disruption is our future work.

By combining the strategies for determining N with the adaptive system, we develop an automated system that can allocate VMs and dynamically adjust the number of VMs to maintain performance and use as few as VMs possible for a very long time. Reducing the number of VMs used can reduce the total cost. Thus our system can keep the cost low.

IV. Evaluation

To evaluate our strategies, we conduct experiments on real streaming data. We first justify our choice of VM type by comparing the performance of different types of VMs. Then, we evaluate the performance of the system in different phases.

A. Experiment Environment

The evaluation uses CAM^2 [8] [9] because it is designed to analyse the data from thousands of network cameras. The camera database in CAM^2 contains more than 70,000 cameras around the world. In this paper, we use 286 AXIS network cameras as the sources for streaming visual data. All the cameras are located in North America. Motion JPEG is used for retrieving data from the cameras. Motion JPEG encoded each frame of the video as an independent JPEG images with no inter-frame compression. Motion JPEG produces more data than the H.264, but it reduces the effect of lost and corrupted frames. All of the cameras can provide data at least 10 frames per second (fps). This frame rate is larger than the requirement of many analysis programs. When the processing rate is smaller than the incoming rate of the streams, no data is buffered. Hence, when the workload is heavy, data will be dropped instead of storing it in memory. Public network cameras are not reliable all the time. They can go offline

without any warning. Thus, although we feed the IP address of 286 cameras to our system, we find that not all of the cameras are responsive over the course of the experiments. Consequently, some these cameras' data are not processed in experiments. Even within one experiment, the number of cameras that are online can change unexpectedly. However, the change of the number of cameras within one experiment is usually relatively small (only 1 or 2 cameras). Thus, the number of cameras over time varies little as number of VMs changes in each experiment. Since the change of the number of cameras is very small compared to the total number of cameras, those changed cameras do not have significant effect on the total frame rate and the number of VMs used.

For the VMs, we use instances from Microsoft Azure. We create a VM image that includes all required programs and libraries. The image allows all instances to have the same software packages. The VMs' configurations, including geographical locations (West US), network bandwidth, disk storage and etc., are configured to be the same. The only parameters that are different are the number of cores and the amount of memory, which we control for our experiments.

For the analysis programs, we use three commonly used image analysis programs from OpenCV, including background subtraction (BGS) [7], Scale-Invariant Feature Transform (SIFT) [10] and Speeded Up Robust Features (SURF) [3]. BGS is an algorithm that builds a background model for a series of consecutive images, and derives the foreground image by subtracting the background model from the original video or images. BGS is commonly used for pre-processing of object detection and tracking to remove the effect of background. SIFT is used to detect and describe local features in images. SURF is also a local feature detector and descriptor and it is partially inspired by SIFT. SIFT and SURF are commonly used in computer vision tasks like object recognition, object tracking, and 3D reconstructions. All the algorithm implementations are provided by OpenCV. We choose these programs because these programs are commonly used, and includes intermediate to heavy computation.

B. Comparison of Different Instance Types

This section explains our choice of VM types. Smaller instances have better performance-cost ratio. We run BGS using the linear increment method with different types of VMs, and let the system run for about 2 hours. The results are shown in Figure 2.

In Figure 2, there are three sets of markers, each set of markers represents one experiment. The x-axis is the time since the beginning of the execution, and the y-axis is the total frame rate. For each set of markers, there are two phases. In the first phase, test phase, the system seeks to determine N by performing test runs. In this experiment, we use the linear increment method, thus we can see the frame rate rises gradually, as more instances are launched. For example, in the 2-core curve in Figure 2, the test phase lasts from 1 minute to approximately 50 minutes. The second phase is the steady phase, in which the real processing happens. In this phase, the

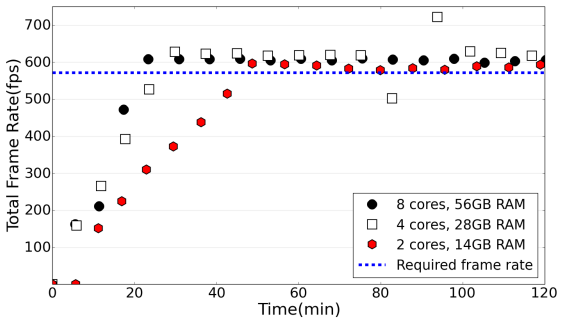


Fig. 2. Measured frame rate of varying VM types using the linear increment method.

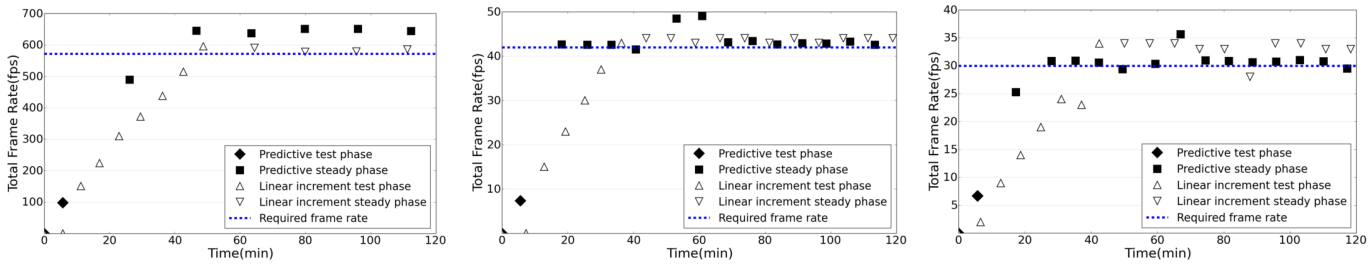
frame rate is stable. Then we monitor the frame rate every 20 minutes, and adjust the number of VMs to cater to the changes of the frame rate. The dashed line represents the users' required frame rate. Here, the required frame rate is 2fps per camera, 572fps in total.

From Figure 2, we can see that all different types of VMs can achieve the required frame rate and provide similar performance. However, when we look at the time and cost in test run phase, they are quite different. The time and cost of the two phases are shown in Table IV.

TABLE IV
TIME AND COST USING DIFFERENT VM TYPES

	VM type		
	2 cores	4 cores	8 cores
Time in the test phase (min)	49	30	23
Number of VMs needed	8	5	4
Cost per hour in steady phase ($)	2.0	2.5	4.0

In Table IV, we can see that when larger instances are used, the time consumed for the test run phase is shorter and the number of VMs needed is smaller. Thus the total test run time and time for launching VMs is shorter. Another factor is the cost per hour in the steady state. When larger instances are used, the cost is higher in the steady state. If we use 2-core VMs, we need 8 VMs for a total of 16 cores. If we use 4-core VMs, we need 5 VMs for a total of 20 cores. This means when using 4-core VM, we need four more cores to achieve the same frame rate compared to using 2-core VMs. When using 8-core VMs, the difference is larger. We need 4 VMs and with a total of 32 cores. We use the price list from Microsoft Azure to calculate the cost per hour in the steady state. When using 4-core VMs, we pay 25% more than using 2-core VMs. When using 8-core VMs, we pay 100% more. Thus, we conclude that indeed smaller instances are better in terms of performance-cost ratio for the considered cases. Virtualization of VMs on real machines is a complicated problem and larger instances required more complex implementation. We hypothesize complex virtualization reduce the efficiency of the instance. Thus, large instances have lower performance-cost ratio. Based on our experiments, we use 2-core VMs for the following experiments.



(a) (b) (c)

Fig. 3. The comparison of different methods using different applications. (a) BGS, (b) SIFT, and (c) SURF. Both methods can achieve the required frame rate eventually, but predictive method takes less time to reach the steady state. However, predictive method's first prediction may be wrong sometimes

C. Evaluation of the Test Phase

After we justify our choice of VM, we can evaluate our system. Since there are two phases in the system, we evaluate the performance of the system in the test phase first. We have two different strategies: *linear increment method* and *predictive method*. We run BGS, SIFT, and SURF with both strategies for about 2 hours. The VM has 2 cores and 14GB RAM. For different applications, we set different required frame rates because the execution time of the applications are different. Computation for BGS is lighter than SIFT and SURF, so we set the required frame rates of BGS to 2 fps per camera. The required frame rate for SIFT and SURF are set to 0.15 and 0.1 fps per camera. The frame rates are chosen so that the number of VMs is not so small that the experiments are not representative of real applications, nor so large that the system can not handle the load. The results are shown in Figure 3.

Here we use different shapes of markers to distinguish the test phase and the steady phase. For the linear increment method, it launches one VM at one time, thus we can see the frame rate increasing linearly. The test run phase ends when the data rate is higher than the required frame rate. After the test run phase, the system transitions into the steady phase, in which the data rate does not frequently change, but can adaptively adjust the number of VMs when needed. For the predictive method, the test run phase is very short because it only involves one test run in all cases. After that, the system will launch all needed VMs. That is why the frame rate abruptly increases after the first marker. Then, it reaches the steady phase.

From Figure 3, we can see the predictive method takes far less time to transition into the steady phase. Table V shows the length of the test run phase for different applications and methods. When using the predictive method, the time required for reaching the steady phase is almost constant, and it does not depend on the required frame rate or the application. This is because in all cases, the system only requires one test run. For the linear increment method, it takes a relatively long time. The linear increment method may take 7 to 8 times of the time compared with the predictive method in these experiments. This is because the run time is proportional to the number of VMs needed. Thus, it depends on the required frame rate, type of VM and applications.

Less time also means less money. We calculate the cost of each method in the test phase. Price policies are different from vendor to vendor. For example, Microsoft Azure charges for VMs by the minute while Amazon EC2 charges for VMs by the hour. For this paper, we use prices from Microsoft Azure because we used Azure instances and the pricing policies for most vendors are similar. The results are also shown in Table V. We can see the difference in cost is even larger than the difference in time. This is because for the linear increment method, the test runs use multiple VMs while the predictive method always use only one. It is clear the predictive method takes less time and costs less.

TABLE V
TIME AND COST IN THE TEST PHASE FOR DIFFERENT APPLICATIONS AND METHODS

Application	The Test Phase Duration (min)		Cost ($)	
	Linear Increment	Predictive	Linear Increment	Predictive
BGS	49	6	0.9375	0.025
SIFT	36	5	0.5	0.021
SURF	42	6	0.7	0.025

The disadvantage of predictive method is that sometimes the prediction is not correct. From Figure 3(a)(c), we can see that the second marker on the predictive method curve is still below the required frame rate. When we do test runs for the predictive method, we use the results from a portion of the cameras to predict the whole set of cameras. If the cameras chosen cannot represent the whole set, then the prediction would be wrong. However, because of the adaptivity of the system, the system will meet the required frame rate after the first time it gets actual total frame rate from the VMs. The linear increment method is resilient to the unpredictable disruptions in network condition or the cameras during test runs. In Figure 3(c), the frame rate drops after the sixth test run for the linear increment method. There may be some abrupt changes in the network condition or the cameras. In that particular test run, even though the number of VMs is larger than before, the frame rate is lower. If the results of this test run is used for predictive method, the prediction would be much larger than

necessary. With linear increment method, the system simply keeps scaling up and can still reach the right number of VMs at the end of the test phase.

D. Evaluation of the Steady Phase

The next feature we evaluate is the adaptivity of the system. We run SIFT using the predictive method and let the system run for over 3 days in order to observe the behavior of the system for a long period of time. We set the required frame rate to be 0.15 frames per second per camera, thus the total frame rate is 42 fps. The results are shown in figure 4.

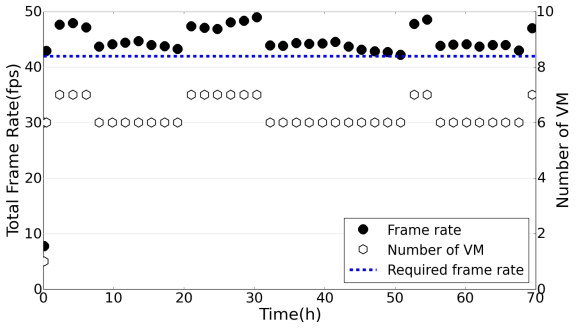


Fig. 4. The results for testing adaptivity of the system. The dashed line shows the frame rate threshold we set. The black markers shows the actual frame rate. The white markers shows the number of VMs used

In Figure 4, we can see that the system can adjust the the number of VMs over time. Whenever the frame rate falls below the required frame rate, the system launches new instances so that the frame rate increases. Whenever the frame rate is so high that fewer instances can achieve the required frame rate, the system can terminate instances to save money while maintaining the required frame rate. The total cost in for the three days is $57. Without the adaptive system, if we use 6 VMs all the time, then the frame rate can not reach required frame rate for 33% of the test time. If we use 7 VMs all the time, the total cost is $63, which is 10.5% more than using the adaptive system.

The changes are somewhat periodic. This is because SIFT extracts features from streams. For the selected cameras, there are more objects in the scene in the day than in the night. Thus, SIFT needs more processing time in the day than in the night. That means, we need more VMs in the days to meet the required frame rate. With this pattern, we can see that the adaptive system is indeed necessary for long time processing and our design can satisfactorily accomplish the task.

V. Conclusion

In this paper, we present an automated system for resource allocation for analysing a large amount of streaming data. We first find that instances with fewer cores are better in performance-cost ratio. Then, we propose two strategies for resource allocation: linear increment method and predictive method. We evaluate and compare the two strategies and find that the predictive method has advantages in execution time and cost, while the linear increment method is resilient to abrupt changes in netowrk condition or content of cameras during the test runs. We develop an adaptive system to dynamically adjust resource to cater to the change of streaming data. We use real network cameras to evaluate our design and conclude that our system works for large amount of streams for a long time.

Acknowledgment

The authors would like to thank Ahmed S. Kaseb, Youngsol Koh, Kyle McNulty, Everett Berry, and the undergraduate research team for their contributions building the CAM2 system.

The authors would like to thank Amazon EC2 and Microsoft Azure for providing the virtual machines used for the experiments in this paper.

This project is supported in part by the US National Science Foundation grants 1535108 and 0958487. Any opinions, findings, and conclusions or recommendations expressed in this presentation are those of the speaker and do not necessarily reflect the views of the National Science Foundation.

References

[1] A. Alasaad, K. Shafiee, S. Gopalakrishnan, and V. Leung. Prediction-based resource allocation in clouds for media streaming applications. In *Globecom Workshops (GC Wkshps), 2012 IEEE*, pages 753–757, 2012.

[2] Amazon. AWS Auto Scaling. http://aws.amazon.com/autoscaling/.

[3] H. Bay, T. Tuytelaars, and L. V. Gool. Surf: Speeded up robust features. In *Computer Vision - ECCV 2006*, pages 404–417, 2006.

[4] W. Chen, A. Mohan, Y.-H. Lu, T. Hacker, W. T. Ooi, and E. J. Delp. Analysis of large-scale distributed cameras using cloud. *IEEE Cloud Computing*, Accepted.

[5] L. De Cicco, S. Mascolo, and D. Calamita. A resource allocation controller for cloud-based adaptive video streaming. In *Communications Workshops (ICC), 2013 IEEE International Conference on*, pages 723–727, 2013.

[6] N. Hemsoth. On the api for harvesting global camera networks. The Platform, http://www.theplatform.net/2015/04/02/an-api-for-harvesting-global-camera-networks/, April 2 2015.

[7] P. KaewTraKulPong and R. Bowden. An improved adaptive background mixture model for realtime tracking with shadow detection. In *Video-Based Surveillance Systems*, 2002.

[8] 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 *IEEE Global Conference on Signal and Information Processing*, 2014.

[9] 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 for large-scale analysis of distributed cameras. In *Imaging and Multimedia Analytics in a Web and Mobile World*, 2015.

[10] D. Lowe. Object recognition from local scale-invariant features. In *Computer Vision, 1999. The Proceedings of the Seventh IEEE International Conference on*, volume 2, pages 1150 – 1157, 1999.

[11] S. N. Z. Park. The Giant Panda Cam. http://nationalzoo.si.edu/animals/webcams/giant-panda.cfm.

[12] A. Purser and L. Thomsen. Monitoring strategies for drill cutting discharge in the vicinity of cold-water coral ecosystems. In *Marine Pollution Bulletin*, volume 64, pages 2309–2316, 2012.

[13] N. P. Service. Hawai'i Volcanoes National Park. http://www.nature.nps.gov/air/webcams/parks/havocam/havocam.cfm.

[14] S. Vijayakumar, Q. Zhu, and G. Agrawal. Dynamic resource provisioning for data streaming applications in a cloud environment. In *Cloud Computing Technology and Science (CloudCom), 2010 IEEE Second International Conference on*, pages 441–448, 2010.

[15] S. Winkler, M. Zessner, E. Saracevic, K. Ruzicka, N. Fleischmann, and U. Wegricht. Cause and effect relationship between foam formation and treated wastewater effluents in a transboundary river. In *Physics and Chemistry of the Earth*, volume 34, pages 565–573, 2009.