Container based Video Surveillance Cloud Service with Fine-Grained Resource Provisioning

Haitao Zhang, Huadong Ma, Guangping Fu, Xianda Yang, Zhe Jiang, Yangyang Gao
Beijing Key Lab of Intelligent Telecomm. Software and Multimedia,
Beijing University of Posts and Telecomm., Beijing, China
Email: {zht, mhd}@bupt.edu.cn; {krystism, yxda2014, zhe.jiang90}@gmail.com; gyyzyp@163.com

***Abstract*—Many current video surveillance cloud platforms are built based on Virtual Machine (VM) technology that usually induces the loss of service performance and brings some challenges in the service management agility. In addition, the efficient resource provisioning for the heterogeneous video services is also a challenging issue in such a dynamic and shared cloud environment. In this paper, we firstly design a novel video surveillance cloud platform that employs the lightweight container technology and is defined according to the ITU standards. Our platform can provide a flexible and reconfigurable video microservice management environment with the high service capacity. Secondly, we propose a predictive fine-grained resource provisioning approach that can periodically predict the future workload and perform the proactive resource supply for the video microservices in the cloud. Our approach utilizes the service similarity matching and the time-series nearest neighbor regression to efficiently predict the future resource requirements, and dynamically optimizes the usage of resources based on predictive results while ensuring quality of service. Finally, we implement the proposed platform, and conduct the extensive experiments. The experimental results indicate that the proposed solution provides the higher service deployment density, accurately predicts the resource demands and significantly improves the resource utilization.**

***Keywords*-Cloud computing, container, video surveillance, resource prediction, resource provisioning**

I. Introduction

Intelligent Video Surveillance System (IVSS) are increasingly used to provide intelligent video analysis and automatic event-based real-time alerts, such as intrusion detection, traffic flow analysis, and traffic violation detection, to enhance city management [1] [2]. Intelligent surveillance video content understanding are data-intensive and computation-intensive by nature. During the last few years, the combination of intelligent video analysis technology and cloud computing is widely accepted as a promising solution to deal with the challenges of large scale surveillance video analysis [3] [4] [5]. The cloud-based video surveillance is emerging as a noteworthy technology which can provide a set of media services (e.g., forwarding, transcoding, analysis) executed in a cloud platform. The cloud computing platform can provide a flexible stack of powerful virtual resources like CPU, memory, storage and network bandwidth to manage the video surveillance services in a cost-efficient mode. The elastic resource provisioning is well suited to scale out or scale in the resources at runtime in response to the changing needs of the continuous video surveillance services.

Current video surveillance cloud is based on VM technologies. The media processing applications can run in parallel across multiple VMs. However, the VM-based cloud platform will induce the loss of the speed and performance of video surveillance applications, and brings some challenges in an agile application management environment [6]. Container technology is much more efficient than VM, and can produce a more lightweight and more agile virtualization computing resources. A recent literature [7] shows that Docker containers consume only 6% of CPU and 16% of memory comparing to a corresponding KVM based VM, and moreover, its boot time is only 60% and its reboot time is only 2% of the corresponding VMs. So container technology will be a very promising alternative for the cloud-based video surveillance platform.

In the video surveillance cloud, every video application may have different resource requirements with each other, and the physical servers may have the heterogeneous resources configuration. In addition, each video application often experiences the highly variable workload that can cause the dynamic resource usage pattern. Therefore, inappropriate virtulization resource allocation in this complex cloud environment may result in resource waste and application performance degradation. The widely accepted strategy is to timely adjust the resource provisioning according to the actual demands of applications, and a predictive resource behavior model should be established previously [8] [9] [10]. The recent literatures proposed to use the fine-grained resources as the basic service units, and gave some resource prediction models and resource scheduling approaches for cloud applications [11] [9]. But they usually do not consider the architecture and the service features of the video surveillance cloud into their solutions. So the previous approaches cannot be directly incorporated in the current video surveillance cloud platform.

In this paper, we focus on the efficient resource utilization and service management issues in video surveillance cloud. The specifics of our contributions are described as follows.

2159-6190/16 $31.00 © 2016 IEEE
DOI 10.1109/CLOUD.2016.103

Firstly, we design a novel container-based video surveillance cloud platform that is integrated with the standard video surveillance system. The system function components are defined according to International Telecommunications Union (ITU) standards [12]. Our platform can agilely manipulate the video surveillance microservices with the configurable underlying resources, and can provide the higher service capacity and resource efficiency by using the lightweight virtualization technology.

Secondly, we propose a prediction-based elastic resource provisioning approach, and it includes two main operations that are respectively the resource prediction and the resource allocation. We employ a two-phase prediction mode, which utilizes the service similarity matching and the time-series nearest neighbor regression based on the video service features and the historical resource usage data. Our resource prediction algorithm is cost-efficient and can accurately predict the resource requirements of the video processing task under the dynamic workloads. Furthermore, based on the results of the resource prediction, our previously proposed resource allocation method is used to dynamically adjust the cloud resource supply for each video microservice at a fine granularity.

Finally, we implement the proposed video surveillance cloud platform based on Docker technology and develop several typical video surveillance microservices. The experimental results show that our cloud platform has a higher service deployment density than that of the VM-based cloud platform. In addition, as compared to the current widely used methods, our resource provisioning approach provides a more cost-efficient and accurate resource behavior prediction, and further improves the flexibility and resource utilization.

The rest of the paper is organized as follows. Section II describes the architecture and workflow of the video surveillance cloud. We propose a resource prediction approach in Section III. Section IV introduces a resource allocation approach. Section V presents the platform implementation and the experimental results. Section VI outlines the related work. We conclude the paper in Section VII.

II. Container based Video Surveillance Cloud

A. System Architecture

Our video surveillance cloud cloud platform is built upon Docker container virtualization technology. It is fully integrated with the video surveillance system that complies with ITU standards [12], and is powered by a scalable computing pool to meet the demand of video services. Figure 1 illustrates the architecture of the proposed video surveillance cloud. As shown in the figure, the system is divided into several components according to their relatively independent functions. The function of each component is presented as follows.

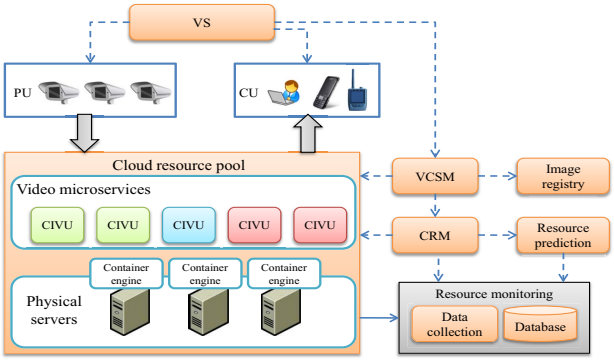


Figure 1. Video surveillance cloud architecture.

1) Cloud resource pool: Cloud resource pool is the core part for the video surveillance cloud and is mainly consisted of many physical machines. Each physical machine is configured with a Docker container engine which is responsible for the creation and destruction of CIVU. The main function of cloud resource pool is used for the provisioning of the actual resources and the creation of the video microservices.

2) PU: Premises Unit (PU). In the cloud-based video surveillance system, PU is typically the IP camera which collects the surveillance video data from the monitoring region. PU can send the live video stream to other system components through broadband networks.

3) CU: Customer Unit (CU). CU provides the integral video surveillance service interface for users. As a highly customizable software, CU can be implemented in different platforms such as PC, and mobile phone. In cloud-based video surveillance system, it initiates the workflow request, receives and presents the instant video processing results from the cloud platform.

4) VS: Visual Surveillance (VS) system is a centralized video surveillance management unit, which is used to remotely capture video and present it to the end user in a user-friendly manner, based on a managed broadband network with ensured quality, security and reliability. In addition, VS can send the video service requests to VCSM for utilizing the cloud resources to efficiently process the surveillance video.

5) VCSM: Video Cloud Service Manager (VCSM) is responsible for managing the video microservices, such as video forwarding, video transcoding, and object tracking. When the video surveillance system sends a video processing request, VCSM will trigger CRM to configure the cloud resources and control the container engine to generate the CIVU which has the corresponding video processing ability. It manages the containers launched from their Docker images stored in the image registry component. The main tasks of VCSM include: (1) Receive and analyze the task request from VS. (2) Manage a set of CIVU. (3) Collect the video processing results from CIVU and send the results

back to VS or CU.

6) CIVU: Containerized Intelligent Video Unit (CIVU) is a container instance which incorporates a specific video processing program to complete the assigned tasks, such as video forwarding, video transcoding, and object tracking. Each CIVU in the cloud resource pool has a unique ID. The configuration of each CIVU is diverse, and depends on the needs of the specific video processing task. Based on the Docker containers, each CIVU can be launched in a subsecond, and we can pack a lot of CIVU onto a physical machine and get the more effective scalability.

7) CRM: Cloud Resource Manager (CRM) is mainly responsible for the scheduling of physical resources. When we need to deploy a CIVU in the cloud resource pool, we can call CRM to assign the suitable amount of resources on the appropriate physical servers. Because the workload of each CIVU and the running state of each server are time-varying, we need to reschedule the resource for the running CIVU to improve the resource utilization. The resource scheduling is performed according to the current resource state and the resource prediction results obtained from the resource prediction component. The cloud resource can be released by CRM once the CIVU is destroyed.

8) Resource monitoring: Resource monitoring component is used to realize the real-time monitoring of resource usage for each running CIVU. It mainly consists of a data collection submodule and a database. The data collection submodule monitors each running CIVU, and can collect the resource usage data such as CPU utilization, memory utilization, and network I/O state. The resource usage data is stored in the database, and can be retrieved by other system components through the service interfaces.

9) Resource prediction: Resource prediction component is used to accurately estimate the future resource requirement of each running CIVU according to the history data stored in the resource monitoring database and the video service related data such as service type, video resolution, and video stream bit rate. For maximizing the resource utilization while ensuring QoS, CRM uses the prediction results to reschedule the cloud resources for each CIVU. The resource prediction algorithm used in this component will be proposed in the next section.

10) Image registry: Image registry is an online image repository which stores all video microservice images. Each image corresponds to a special video processing task. By combining with the online version control system GitHub and the continuous integration tool Jenkins (which are not shown in Figure 1), it makes our system realize the automatic building for the video microservice program.

B. Deployment, Destruction, and Migration of Services

In this subsection, we focus on the main workflows of the container based video surveillance cloud platform, which include service deployment, service destruction and service migration.

1) Service Deployment: CU initiates a video surveillance service request. VS receives the service request from CU and forwards it to VCSM. Then, VCSM sends the resource request to CRM. When CRM receives the resource request from VCSM, it will traverse all the physical servers and choose the appropriate host according to the current resource state and the service request. After the successful resource scheduling, CRM sends a confirmation message to the VCSM. After obtaining the cloud resource, VCSM will request the container engine in the current scheduled host to pull the corresponding image from the image registry and then create and start a CIVU which provides the requested video service. After the CIVU is started, the video process program in CIVU will run automatically.

2) Service Destruction: Once the video processing task of CIVU ends or the user terminates the video processing task, VS sends the ID of the corresponding CIVU to VCSM, and requests for destructing the CIVU. After VCSM receives the request, it will call CRM to destruct the corresponding CIVU and release the occupied resources through communicating with the container engines.

3) Service Migration: When a CIVU runs, the resource usage data of the CIVU is recorded by the resource monitoring component. By utilizing the historical resource data stored in the database, the resource prediction component periodically predicts the resource requirements of the CIVU in the next time period. CRM periodically requests for the prediction results from the resource prediction component. Once the predicted demand quantity of one resource (such as CPU) is below a given minimum threshold or above a given maximum threshold, CRM will reschedule the appropriate cloud resource for the currently running CIVU according to the resource prediction result and the current cloud resource running state. After the successful resource rescheduling, CRM will request VCSM to destruct the CIVU deployed in the current container engine and redeploy a new CIVU with the same function in the container engine hosted in the rescheduled cloud resource.

III. Resource Prediction

In this section, we present our resource prediction approach that is used to predict the future resource requirement needed by each CIVU with time-variable workload.

A. Problem Formulation

In the cloud platform, a media processing task performed by CIVU i is expressed by $x_i = (a_i^1, a_i^2, ..., a_i^n)$ where a_i^k is the k-th attribute of the task i ($k \in \{1, 2, ..., n\}$). Each media processing task includes n attributes which can be the surveillance video resolution, the video stream bit rate, and the type of task like video forwarding, video transcoding, and object detection.

The resource usage data of CIVU i is collected and stored in the database, and is defined as $S_i(t) = \{S_i^c(t), S_i^m(t), S_i^{tx}(t), S_i^{rx}(t)\}$ where $S_i^c(\cdot)$, $S_i^m(\cdot)$, $S_i^{tx}(\cdot)$, and $S_i^{rx}(\cdot)$ are respectively the resource usage amount of CPU, memory, uplink bandwidth and downlink bandwidth. For example, the CPU resource usage amount of CIVU i is expressed by a recorded sequence $S_i^c(t) = \{s_i^c(1), s_i^c(2), ..., s_i^c(t)\}$ until the current time instant t. Because the resource prediction process for each type of resource is the same, we will use the CPU workload prediction as an example to introduce our approach.

The cloud platform can dynamically reschedule the resource according to the resource prediction results for each CIVU. For reducing the overhead of the frequent resource rescheduling, the resource prediction approach should predict the future resource requirement for a specific period of time. In this paper, the objective of our resource prediction is to predict the resource workload $\{s_i^c(t+1), ..., s_i^c(t+p)\}$ of CIVU i based on $S_i^c(t)$. Our resource prediction approach is performed periodically in every p time units.

B. Proposed Approach

Our approach includes two phases which are respectively the initial resource requirement prediction and the continuous resource requirement prediction.

Initial prediction phase. In the service deployment process, VCSM requires CRM to allocate the cloud resource for CIVU i, and we do not have any resource usage data of i. The initial resource amount can be determined by user, and is estimated by the past experience. But this method is very inaccurate, and may lead to serious the waste of resources or the performance degradation. In this paper, we propose another method to estimate the initial resource requirement of a CIVU. Assume that we have the resource usage data of many other CIVUs R, and each $j \in R$ may be the current running CIVU in the cloud resource pool or the previously ending CIVU. For accurately estimating the initial workload of CIVU i, we measure the task similarity between CIVU i and each CIVU j, and then use the resource usage data of similar task to calculate the initial resource allocation value.

Firstly, we normalize each attribute value $a_.^k$ of the media processing task into $[0, 1]$, and measure the task similarity between i and j by using the normalized attribute vector $\overline{x}_. = (\overline{a}_.^1, ...\overline{a}_.^k, ..., \overline{a}_.^n)$. The most similar task $j'(\in R)$ with i is calculated by

$$j' = \arg\min_{j \in R} \|\overline{x}_i - \overline{x}_j\|_2 \tag{1}$$

where $\|\cdot\|_2$ is the L^2 norm operator.

Secondly, we choose the peak resource usage $\widehat{s}_{j'}^c(t) = \max\{s_{j'}^c(t-l), ..., s_{j'}^c(t)\}$ among the resource usage value of the last l time units of j' as the initial resource allocation value for i.

Continuous prediction phase. The load of IDCs usually shows some pattern, and it either exhibits periodical nature or is orderly during some period [8]. In the process of media task execution, we can use the historical resource usage data to predict the resource requirement. We propose a Time-Series Nearest Neighbor Regression (TSNNR) algorithm which utilizes the similarity of the historical time series data trends to predict resource requirement.

Step 1. We needs to find the useful past time series data that has the similar pattern with the most recent time series data. The most nearest neighbor of the time series data $(s_i^c(t-q), ..., s_i^c(t))$ is obtained by the following neighbor distance equation

$$d_i^c(t') = \|(s_i^c(t'-q), ..., s_i^c(t')) - (s_i^c(t-q), ..., s_i^c(t))\|_2 \tag{2}$$

where t' is the past time instant stored in the database that we want to compare with t, and q indicates how many steps back in time we consider. Then, we can define the nearest neighbor time instant as follows

$$\widetilde{t} = \arg\min_{t' \in T} d_i^c(t') \tag{3}$$

where $T = \{1, 2, ..., t\}$. So we repeats the nearest neighbor finding process y times to obtain the time instant set $N_y(t)$ that consists of y nearest neighbor time instants.

Step 2. The resource requirement of $t+1$ time instant is estimated by

$$\widetilde{s}_i^c(t+1) = s_i^c(t) + \frac{\sum_{t' \in N_y(t)} (s_i^c(t'+1) - s_i^c(t'))}{y}. \tag{4}$$

Step 3. We use the predicted value $\widetilde{s}_i^c(t+1)$ as a new initial data to find the similar past time series data with $(s_i^c(t-q+1), ..., s_i^c(t), \widetilde{s}_i^c(t+1))$, and repeat the step 1 and step 2 p times to obtain the predicted resource workload $\widetilde{S}_i^c(t,p) = \{\widetilde{s}_i^c(t+1), ..., \widetilde{s}_i^c(t+p)\}$

Step 4. In order to ensure the desired QoS, we choose the peak predicted workload plus a prediction error e as the final resource requirement which is given by

$$\widetilde{s}_i^c(t,p) = \max \widetilde{S}_i^c(t,p) + e \tag{5}$$

where e is the average of the historical prediction error defined as $e = \frac{1}{t}\sum_{t^* \in T} |s_i^c(t^*) - \widetilde{s}_i^c(t^*)|$.

Therefore, we get the predicted requirement value of CPU in the next period of time, and the prediction of other resources like memory and bandwidth can be performed using the same way.

IV. Resource Allocation

In the cloud resource pool, the physical servers have the heterogeneous resource configurations, and the resource usage states of the servers are time-various due to the multitasking resource sharing ability. In this section, based on the resource prediction results of Section III, we use a fine grained resource allocation approach proposed in our previous work [4] to efficiently utilize the cloud resource. Given the resource requirements of CIVUs and the current

resource usage states, our objective is to appropriately schedule each CIVU on the physical servers to maximize the cloud resource utilization.

To model the resource allocation problem in the video surveillance cloud platform, we assume that there exist M heterogeneous physical servers $P = \{1, 2, ..., M\}$ in the cluster. Each independent server $j \in P$ can be expressed as $p_j(t) = \{p_j^c(t), p_j^m(t), p_j^{tx}(t), p_j^{rx}(t)\}$ where $p_j^c(\cdot)$, $p_j^m(\cdot)$, $p_j^{tx}(\cdot)$, and $p_j^{rx}(\cdot)$ are respectively the available resource amount of CPU, memory, uplink bandwidth and downlink bandwidth until the current time instant t. Then, we can obtain the total available resource across the whole cluster until the current time instant t, which is defined as $T(t) = \{T^c(t), T^m(t), T^{tx}(t), T^{rx}(t)\}$. For example, the total available resource amount of CPU can be calculated by $T^c(t) = \sum_{j=1}^{M} p_j^c(t)$.

As mentioned in the above Section, the resource requirement can be predicted according to the historical running data of CIVU, and will be placed in the task queue of CRM not only in initial prediction phase but also in continuous prediction phase. Therefore, one resource requirement i in the task queue can be represented as $\widetilde{S}_i(t,p) = \{\widetilde{s}_i^c(t,p), \widetilde{s}_i^m(t,p), \widetilde{s}_i^{tx}(t,p), \widetilde{s}_i^{rx}(t,p)\}$.

Algorithm 1 Dominant Resource First Allocation

```
1: while taskQueue is not empty do
2:     task ← taskQueue.get();
3:     r̄ ← argmax (task.ProportionalVector);
4:     candidateSet ← ∅;
5:     for ∀pm ∈ pmList do
6:         if pm meets task then
7:             r* ← arg max(pm.ProportionalVector);
8:             if r̄ equals to r* then
9:                 candidateSet = candidateSet ∪ pm;
10:            end if
11:        end if
12:    end for
13:    candidate ← ∅;
14:    if candidateSet ≠ ∅ then
15:        minDistance ← ∞;
16:        for ∀pm ∈ candidateSet do
17:            distance ← DISTANCE(pm, task);
18:            if minDistance > distance then
19:                minDistance ← distance;
20:                candidate ← pm;
21:            end if
22:        end for
23:    else
24:        candidate ← RANDOM(task, pmList);
25:    end if
26:    candidate allocates resource to task;
27: end while
```

For adequately making use of each kind of resource, we define the concept of dominant resource [13]. Firstly, we calculate the physical resource proportional vector $P_j(t) = (\frac{p_j^r(t)}{T^r(t)})$ $(r \in \{c, m, tx, rx\})$ for each physical server j, and select $r_j^p(t) = \arg\max_r \frac{p_j^r(t)}{T^r(t)}$ as the dominant resource of physical server j. Secondly, we calculate the task requirement resource proportional vector $S_i(t) = (\frac{\widetilde{s}_i^r(t)}{T^r(t)})$ $(r \in \{c, m, tx, rx\})$ for each task i, and select $r_i^s(t) = \arg\max_r \frac{\widetilde{s}_i^r(t)}{T^r(t)}$ as the dominant resource of task i.

Our resource allocation algorithm schedules the task request on the physical server with the same dominant resource, and is given by Algorithm 1. In detail, following the FIFO principle, our algorithm always gets the head resource request from the task queue to perform the resource allocation (line 2), and calculates the task's dominant resource $r_\cdot^s(\cdot)$ (line 3). Then, a two-step process is carried out to select the best physical server. Firstly, by traversing the physical server list $pmList$, we search for all the candidate server to form $candidateSet$. Each candidate server in $candidateSet$ should meet the resource request (line 6) and then simultaneously own the same dominant resource with $r_\cdot^s(\cdot)$ (line 8). Secondly, we select the best physical server from $candidateSet$ (line 16-22). We adopt the multi-dimension resource matching metric which can be defined as the Euclidean distance of the task requirement resource proportional vector and the physical resource proportional vector (line 17)

$$distance = \|P_j(t) - S_i(t)\|_2. \tag{6}$$

We employ the best fit strategy, and schedule the task to the physical server that has the shortest distance with the task. Moreover, in the second step, if the $candidateSet$ is empty, we will select the one which can meet the request as $candidate$ at random from the $pmList$ (line 24).

V. Experimental Results

A. System Implementation and Experimental Setting

Before performing the performance validation, we introduce the system implementation details and experimental environment used in the following subsection. Our container-based video surveillance system is implemented according to the architecture presented in Section II. The cloud platform consists of 10 physical servers, and the configurations of the physical servers are shown in Table I. One of the physical servers is used as the controller node that includes the supporting functions such as identity service, microservice management, resource scheduling and image service. Another physical server performs the resource monitoring and prediction. The other physical servers are the compute nodes, which are equipped with a Docker container engine and can run CIVU.

The experimental video streaming data is obtained from a real video surveillance system deployed in Fuzhou, China. For verifying the performance of the cloud platform, we

Table I
PHYSICAL SERVERS.

Physical Servers				
Instance Type	CPU	memory	bandwidth	number
1	24 core	32 GB	100 M	2
2	32 core	32 GB	100 M	2
3	32 core	64 GB	50 M	2
4	32 core	32 GB	50 M	2
5	16 core	32 GB	50 M	2

choose three typical video surveillance services which are respectively Video transcoding, video forwarding, and video synopsis [14].

B. Results and Analysis

Firstly, we validate the service capacity of our container-based video surveillance platform compared with VM-based platform. We run three typical video processing services in two kinds of the virtualization cloud servers respectively, and record the maximum number of CIVU and VM-based IVU (VIVU) deployed in the saturation host. We define the host saturation state as follows: When the idle time of CPU is less than 10% or the physical memory usage is higher than 90%. We choose a Instance Type 1 server as the test host. In order to make the experimental results more accurate, we stop the other user processes in the host and empty the disk cache. After the video services start, we have a sampling of the CPU idle time and the physical memory usage every 5 seconds. Figure 2 shows the average maximum service number of CIVU and VIVU deployed in the saturation host under the different service types. As shown in the figure, due to the lightweight nature of container, the service deployment density of our cloud platform can be nearly 2 times of the deployment density of VMs. High deployment density in cloud computing means that fewer physical machines are needed to support the same amount of clients, thereby reducing the operation costs for cloud providers.

Secondly, we compare our resource prediction algorithm TSNNR with the widely used ARIMA model based approach [10]. In our experiments, the different video microservices are respectively used for test, and we only select the video forwarding microservice to illustrate the experimental results. We collect the resource usage data of consecutive 100 hours of the video forwarding microservice, and use the beginning 10 hours of historical data as the initial training dataset. The resource prediction is executed using TSNNR algorithm and ARIMA based algorithm respectively. For our algorithm, the time period is set to 10 minutes. We take 3 times of the time periods as a prediction period. Through the extensive tests, we finally adjust the parameters to $y = 5$, and q is set to one time period. Figure 3 respectively shows the prediction results of CPU utilization and uplink bandwidth usage with the two methods. As shown in the figure, the prediction results of these two methods all well fit the actual value curve. We further

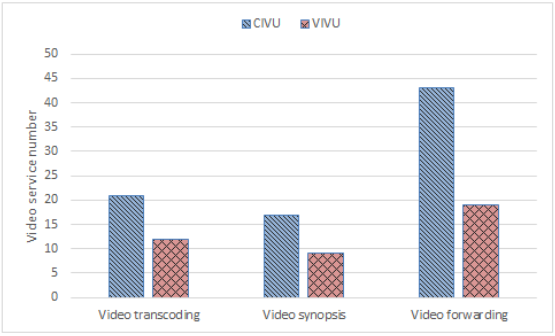


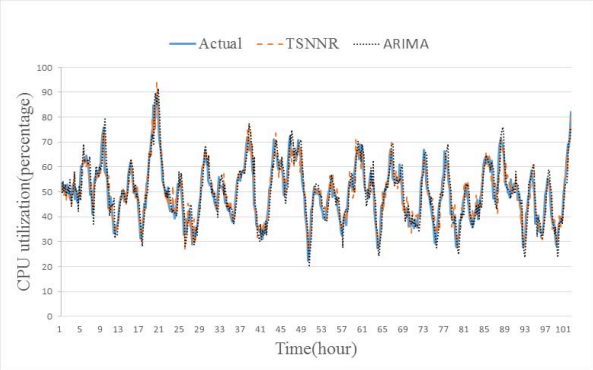
Figure 2. Service densities with different service types.

calculate the mean absolute percentage errors of the TSNNR and ARIMA by ($err = \frac{1}{n}\sum_{i=1}^{n}\frac{|real_i - predict_i|}{real_i} \times 100\%$), which are 7.92% and 8.71%. On the other hand, as reported in [10], ARIMA based methods have the complexity equal to $O(n^2)$, where n is the number of points in the workload used for prediction. In contrast, the complexity of our TSNNR method is only $O(n)$, where n is the size of sample set. Therefore, the results show that our prediction method is more accurate and efficient.

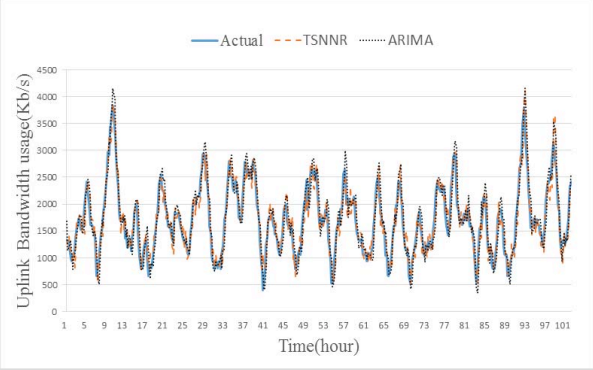
Thirdly, we compare our resource allocation algorithm DRFA with two other practical methods. One is $FilterScheduler$ which is the default scheduling policy used in OpenStack [15]. Another is $VectorDot$ proposed in recent literature [16], in which the objective is to select the physical server with the minimum value of the regular dot product of two vectors: $dotproduct(A, B) = \sum_{1 \le i \le |A|} a_i * b_i$ where $A = (a_1, ...a_i, ...a_n)$ means the physical server's load vector and $B = (b_1, ...b_i, ...b_n)$ means the CIVU's resource requirements as a fraction of the physical server's total capacity. In the experiment, we employ the heterogeneous resource allocation scenario with 6 types of CIVU and the different physical servers (given in Table I). Then, we calculate the average resource utilization ratio of the cluster when the cluster starts to reject the task requests. We repeat the experiment 10 times and calculate the mean value of the experimental results. Figure 4 gives the average resource utilization ratio of CPU, memory and uplink bandwidth of the three different methods. As shown in the figure, our algorithm DRFA outperforms the other two methods, and the utilization ratios of CPU and memory are all over 90%.

VI. RELATED WORK

There are some recent initiatives that focus on carrying out video processing tasks in the cloud, overcoming the processing power limitations of the traditional video surveillance applications [17]. In [18], the author analyzed the suitability of cloud solution by comparing video surveillance local infrastructure with the proposed cloud-based system in terms of performance, storage, scalability, reliability and



(a)



(b)

Figure 3. Results of the TSNNR and ARIMA-based prediction for 100 hour period. (a) Predicted and actual values of CPU utilization. (b) Predicted and actual values of uplink bandwidth usage.

collaborative sharing of media streams. Hossain et al. [19] proposed a dynamic resource allocation approach using a liner programming method for service composition in cloud-based video surveillance platform. Xiong et al. [3] proposed a cloud-based video surveillance prototype implemented in a private campus network. In [4], a multi-resource virtual machine allocation algorithm was proposed for video surveillance cloud. However, the above solutions are based on VM technology that is hardware level virtualization and has a lot of extra overhead and performance losses compared to non-virtualized physical machine. Container-based virtualization is an operating system level virtualization technology which presents interesting alternative to virtual machines in cloud [20]. It can offer better performance and easier deployment than VMs and has almost the same performance as the non-virtualized physical machine [21].

By adopting cloud computing, a major issue is how to efficiently utilize resources while meeting application service demand. To address this issue, many recent studies [11] [22] proposed the fine-grained cloud resource planning solutions. But application resource demand is varying due to time period. Fang et al. [8] proposed a cloud resource pre-

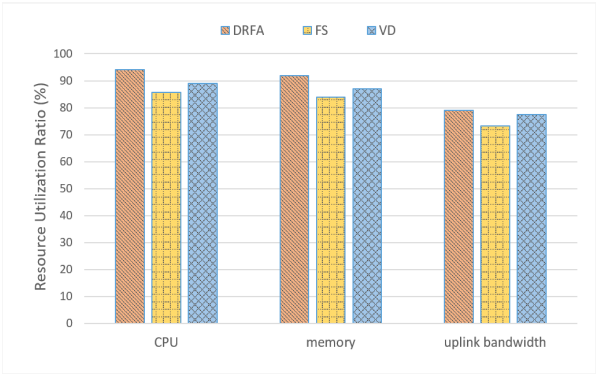


Figure 4. Resource utilization ratio.

diction and provisioning scheme that automatically predict future demand and perform proactive resource provisioning for cloud applications. Hu et al. [9] proposed a multi-step-ahead load prediction method based on a statistical learning technology for improving the efficiency of resources provisioning. To meet the QoS with a cost-effective amount of resources, Calheiros et al. [10] presented a cloud workload prediction method based on the AutoRegressive Integrated Moving Average (ARIMA) model. A second order autoregressive moving average was used to forecast load for cloud application in [23]. Jiang et al. [24] proposed a self-adaptive prediction solution to enable the instant cloud resources provisioning. The Network Weather Service [25] is a famous system designed to provide dynamic Grid resources performance forecasts. Wu et al. [26] proposed several performance prediction models for the Grid to profile workloads as well as infrastructure variations. But the above resource prediction literatures do not take the specific video service features into account, and do not give the method of how to integrate with the container-based cloud application systems.

VII. Conclusions

In this paper, we investigate the container-based video surveillance cloud solution. Our cloud platform leverages Docker container technology to provide the flexible, agile and efficient video microservices, and is seamlessly integrated with the video surveillance system conforming to ITU standards. Then, we present a predictive cloud resource provisioning approach for the video microservices that uses the workload predictions to adaptively manage resource mapping. Our approach takes the advantage of the service similarity matching and the time-series nearest neighbor regression to efficiently predict the resource requirement of each video microservice, and proactively plans the fine-grained resource allocation. We develop several video microservices on the cloud platform, and conduct the extensive experiments. The experimental results show our container-based cloud platform has a higher service deployment den-

sity, and demonstrate the accurate resource predictions and the efficient resource usage of our resource provisioning approach.

ACKNOWLEDGMENT

This work is supported by the National High Technology Research and Development Program of China (No. 2014AA015101); National Natural Science Foundation of China (No. 61300013); Doctoral Program Foundation of Institutions of Higher Education of China (No. 20130005120011); Asia Foresight Program under NSFC Grant No. 61411146001; the Beijing Training Project for the Leading Talents in S&T (ljrc201502); Cosponsored Project of Beijing Committee of Education.

REFERENCES

[1] X. M. Zhao, H. D. Ma, H. T. Zhang, Y. Tang, and G. P. Fu, "Metadata Extraction and Correction for Large-Scale Traffic Surveillance Videos," *IEEE Big Data 2014*.

[2] R. S. Feris, B. Siddiquie, J. Petterson, Y. Zhai, A. Datta, L. M. Brown, and S. Pankanti, "Large-Scale Vehicle Detection, Indexing, and Search in Urban Surveillance Videos," *IEEE Trans. on Multimedia*, vol. 14, no. 1, pp. 28-42, 2012.

[3] Y. H. Xiong, S. Y. Wan, Y. He, and D. Su, "Design and implementation of a prototype cloud video surveillance system," *J. Adv. Comput. Intell. Intell. Informatics*, vol. 18, no. 1, pp. 40-47, 2014.

[4] X. D. Yang, H. T. Zhang, H. D. Ma, W. S. Li, G. P. Fu, and Y. Tang, "Multi-Resource Allocation for Virtual Machine Placement in Video Surveillance Cloud," *HCC 2016*.

[5] X. M. Zhao, H. D. Ma, H. T. Zhang, Y. Tang, and Y. Kou, "HVPI: Extending Hadoop to Support Video Analytic Applications," *IEEE Cloud 2015*.

[6] C. Anderson, "Docker," *IEEE Software*, vol. 32, no. 3, pp. 102-c3, 2015.

[7] L. Li, T. Tang, W. Chou, "A REST Service Framework for Fine-Grained Resource Management in Container-Based Cloud," *IEEE Cloud 2015*.

[8] W. Fang, Z. H. Lu, J. Wu and Z. Y. Cao, "RPPS: A Novel Resource Prediction and Provisioning Scheme in Cloud Data Center," *IEEE 9th SCC*, pp. 609-616, 2012.

[9] R. Hu, J. Jiang, G. Liu and L. Wang, "KSwSVR: A new load forecasting method for efficient resources provisioning in cloud," *IEEE 10th SCC*, pp. 120-127, 2013.

[10] R. Calheiros, E. Masoumi, R. Ranjan, and R. Buyya, "Workload Prediction Using ARIMA Model and Its Impact on Cloud Applications' QoS," *IEEE Trans. on Cloud Computing*, vol.3, no.4, pp. 449-458, 2015.

[11] H. Zhao, M. Pan, X. Liu, X. Li, and Y. Fang, "Exploring Fine-Grained Resource Rental Planning in Cloud Computing," *IEEE Trans. on Cloud Computing*, Vol. 3, no. 3, pp. 304-317, 2015.

[12] ITU-T H.626 standard, "Architectural requirements for visual of surveillance," 2011.

[13] W. Wang, B. Li, and B. Liang, "Dominant Resource Fairness in Cloud Computing Systems with Heterogeneous Servers," *IEEE INFOCOM*, 2014.

[14] Y. Pritch, A. Rav-Acha, and S. Peleg, "Nonchronological video synopsis and indexing," *IEEE Trans. on PAMI*, vol. 30, no. 11, pp. 1971-1984, 2008.

[15] OpenStack, http://www.openstack.org/.

[16] A. Singh, M. Korupolu, and D. Mohapatra, "Server-storage virtualization: integration and load balancing in data centers," *ACM/IEEE SC*, pp. 1-12, 2008.

[17] M. Al-Rawahi, and E.A. Edirisinghe, "Video Forensics in Cloud Computing: The Challenges & Recommendations," *Journal of Information Sciences and Computing Technologies*, vol. 3, no. 2, pp. 201-216, 2015.

[18] M. Hossain, M. Hassan, M. Qurishi and A. Alghamdi, "Analyzing the Suitability of Cloud-based Multimedia Surveillance Systems," *15th IEEE HPCC & EUC*, pp. 644-650, 2013.

[19] M. Hossain, M. Hassan, M. Qurishi and A. Alghamdi, "Resource Allocation for Service Composition in Cloud-based Video Surveillance Platform," *2012 IEEE ICME Workshops*, pp. 408-412, 2012.

[20] D. Bernstein, "Containers and Cloud: From LXC to Docker to Kubernetes," *IEEE Cloud Computing*, Vol. 1, no. 3, pp. 81-84, 2014.

[21] W. Felter, A. Ferreira, R. Rajamony, and J. Rubio, "An updated performance comparison of virtual machines and Linux containers," *2015 IEEE ISPAS*, pp.171-172, 2015.

[22] C. Joe-Wong, S. Sen, T. Lany, and M. Chiang, "Multi-Resource Allocation: Fairness-Efficiency Tradeoffs in a Unifying Framework," *2012 IEEE INFOCOM*, pp. 1206-1214, 2012.

[23] N. Roy, A. Dubey, and A. Gokhale, "Efficient autoscaling in the cloud using predictive models for workload forecasting," *IEEE Cloud*, pp. 500-507, 2011.

[24] Y. Jiang, C. Perng, T. Li, and R. Chang, "Asap: A self-adaptive prediction system for instant cloud resource demand provisioning," *IEEE ICDM*, pp. 1104-1109, 2011.

[25] M. Swany and R. Wolski, "Multivariate resource performance forecasting in the network weather service," *ACM/IEEE Supercomputing*, pp. 1-10, 2002.

[26] Y. Wu, K. Hwang, Y. Yuan, and W. Zheng, "Adaptive workload prediction of grid performance in confidence windows," *IEEE Trans. on Parallel and Distributed Systems*, Vol. 21, no. 7, pp. 925-938, 2010.