A Flexible and Scalable Architecture for Video Surveillance as a Service Systems

Thanathip Limna

A Thesis Submitted in Fulfillment of the Requirements for the Degree of
Doctor of Philosophy in Computer Engineering
Prince of Songkla University

2017

A Flexible and Scalable Architecture for Video Surveillance as a Service Systems

Thanathip Limna

A Thesis Submitted in Fulfillment of the Requirements for the Degree of Doctor of Philosophy in Computer Engineering
Prince of Songkla University

2017

Thesis Title	A Flexible and Scalable Architecture for Video Surveillance as a Service Systems
Author	Mr. Thanathip Limna
Major Program	Computer Engineering

Major Advisor	**Examining Committee:**
....	.... Chairperson
(Asst. Prof. Dr. Pichaya Tandayya)	(Asst. Prof. Dr. Nikom Suvonvorn)
	.... Committee
Co-advisor	(Asst. Prof. Dr. Pichaya Tandayya)
....	.... Committee
(Assoc. Prof. Dr. Chokchai Leangsuksun)	(Assoc. Prof. Dr. Chokchai Leangsuksun)
	.... Committee
	(Asst. Prof. Dr. Worawan Diaz Carballo)

The Graduate School, Prince of Songkla University, has approved this thesis as fulfillment of the requirements for the Doctor of Philosophy Degree in Computer Engineering.

....
(Assoc. Prof. Dr. Teerapol Srichana)
Dean of Graduate School

This is to certify that the work here submitted is the result of the candidate’s own investigations. Due acknowledgement has been made of any assistance received.

....Signature
(Asst. Prof. Dr. Pichaya Tandayya)
Major Advisor

....Signature
(Assoc. Prof. Dr. Chokchai Leangsuksun)
Co-advisor

....Signature
(Mr. Thanathip Limna)
Candidate

I hereby cetify that this work has not been accepted in substance for any degree, and is not being currently submitted in candidature for any degree.

....Signature
(Mr. Thanathip Limna)
Candidate

Thesis Title	A Flexible and Scalable Architecture for Video Surveillance as a Service Systems
Author	Mr. Thanathip Limna
Major Program	Computer Engineering
Academic Year	2016

ABSTRACT

In recent years, moving traditional video surveillance systems into the Cloud-based Video Surveillance (CVS) system or Video Surveillance as a Service (VSaaS) is significant to support a large number of IP cameras via the Internet. Most concerned about applying the cloud computing technology and quality of service rather than the scalability and flexibility of the system. In this thesis, the two issues are considered in the design and implementation of a VSaaS system architecture. The architecture addresses a flexible and scalable component-based VSaaS that can be easily scaled from one server up to a complex cluster to support the varying requirements of users. The publish-subscribe message passing mechanism has been used for the cooperation between the controller and compute node worker, and it contributes to the system fault tolerance and scalability. In case of cloud computing resource management in this architecture, cloud services are accessed via Amazon AWS, especially EC2 and S3 Application Program Interfaces (APIs) for computing services and object storage respectively, as many cloud computing providers are supporting those APIs. Moreover, this thesis also presents possible component deployment plans suitable for any size or type of systems, which combine both physical and virtual machines. The API server applying the REST interface and a token based authentication has been designed to support multiple types of clients as well as to protect the controller from direct security attacks. Also, this thesis presents the concept of having the compute node worker separately designed and worked apart from the video processor. Therefore, not only the compute node worker can support video processors but also related stream processing of which interfaces implemented based on standard I/O.

In case of flexibility and scalability, the scheduling process plays an important role for the computing resource usage efficiency of a VSaaS system. Few previous works de-scribed video processing workload analysis and few video processing factors were revealed. Having unknown resource usage information of video processing it usually is difficult to search for an appropriate available computing node to assign for the requested video processing task. This thesis discusses in details about video processing workload characteristics applying various parameters, such as the type of video processing task, frame rate, frame size, and computing node specification. The analytical results have been applied to design the scheduler process to suitably place video processing tasks at different compute node specifications. This thesis proposes the video processing workload exploration for observing the capacity of available compute nodes by varying the parameters of related video processing tasks. The exploration data is then stored in a database to be used by the scheduler as the information for estimating the video processing resource usage of a new video processing task. The method and algorithm for estimating the resource usage of a new video processing task have been suggested, employing both data from the video processing task exploitation and CPU scaling factor. Furthermore, this thesis suggests the scheduler's criteria to assist the VSaaS administrator in optimizing the system resource usage suitable for the system type needed.

ชื่อวิทยานิพนธ์	สถาปัตยกรรมระบบบริการตรวจตราด้วยกล้องวิดีโอที่ยืดหยุ่นและขยายตัวได้
ผู้เขียน	นายธนาธิป ลิ่มนา
สาขาวิชา	วิศวกรรมคอมพิวเตอร์
ปีการศึกษา	2559

บทคัดย่อ

ในหลายปีที่ผ่านมา มีการเปลี่ยนผ่านจากระบบตรวจตราด้วยกล้องวิดีโอวงจรปิดแบบดั้งเดิม เป็นระบบตรวจตราด้วยกล้องวิดีโอที่ทำงานอยู่บนระบบการประมวลผลแบบกลุ่มเมฆ หรือระบบบริการตรวจตราด้วยกล้องวิดีโอ การเปลี่ยนแปลงดังกล่าวส่งผลให้ระบบตรวจตราด้วยกล้องวิดีโอสามารถรองรับการประมวลผลวิดีโอจากกล้องไอพีจำนวนมากผ่านทางระบบอินเตอร์เน็ตได้ ระบบที่สร้างขึ้นมาใหม่นั้นโดยส่วนใหญ่คำนึงถึงการประยุกต์ใช้งานเทคโนโลยีการประมวลผลแบบกลุ่มเมฆ และคุณภาพการให้บริการ มากกว่าคำนึงถึงความยืดหยุ่นและการขยายตัวได้ของระบบ วิทยานิพนธ์นี้จึงสนใจในสองประเด็นดังกล่าวเพื่อใช้สำหรับการออกแบบและพัฒนาสถาปัตยกรรมระบบบริการตรวจตราด้วยกล้องวิดีโอขึ้นมาใหม่ การออกแบบสถาปัตยกรรมนั้นเน้นแบ่งการทำงานของระบบออกเป็นโมดูลต่างๆ เพื่อให้ง่ายต่อการขยายระบบจากเครื่องเซิร์ฟเวอร์เครื่องเดียว ไปยังกลุ่มของเซิร์ฟเวอร์จำนวนมากที่มีความซับซ้อน เพื่อให้ระบบที่นำเสนอสามารถตอบสนองความต้องการของผู้ใช้งานระบบที่มีความหลากหลายและมีจำนวนมาก กระบวนการส่งผ่านข้อความในรูปแบบที่แบ่งออกเป็น ฝ่ายผลิตสื่อและฝ่ายรับสื่อในลักษณะของสมาชิก จึงถูกเลือกใช้งานสำหรับการสื่อสารระหว่าง โมดูลควบคุม และโมดูลหน่วยประมวลผล ซึ่งช่วยในการป้องกันความล้มเหลวและเพิ่มความสามารถในการขยายตัวของระบบด้วยอีกทางหนึ่ง ในส่วนของการจัดการทรัพยากรการประมวลผลแบบกลุ่มเมฆนั้น ในสถาปัตยกรรมระบบตรวจตราด้วยกล้องวิดีโอที่นำเสนอนี้ จะใช้งานรูปแบบการเชื่อมต่อผ่านทาง Amazon AWS โดยเฉพาะส่วนของ EC2 และ S3 เป็นหลัก สำหรับใช้จัดการเครื่องคอมพิวเตอร์ที่ใช้ในการประมวลผลและการจัดเก็บข้อมูล ซึ่งผู้ให้บริการการประมวลผลแบบกลุ่มเมฆส่วนใหญ่สนับสนุนรูปแบบการทำงานดังกล่าว วิทยานิพนธ์นี้ยังได้นำเสนอรูปแบบการจัดการเครื่องเซิร์ฟเวอร์ที่เหมาะสมต่อสถานการณ์ต่างๆ เพื่อการให้บริการระบบตรวจตราด้วยกล้องวิดีโอตามขนาดและประเภทของระบบที่ต้องการ รวมถึงการใช้งานระบบบนเซิร์ฟเวอร์จริงและเซิร์ฟเวอร์เสมือน ในส่วนต่อประสานกับระบบหลักนั้นเพื่อให้สามารถรองรับเครื่องลูกข่ายได้หลากหลายประเภท จึงใช้รูปแบบการสื่อสารแบบ REST ร่วมกับการพิสูจน์ตัวตนด้วยโทเคน การออกแบบดังกล่าวยังช่วยป้องกันระบบหลักออกจากการใช้งานโดยตรงจากผู้ใช้งานระบบเพื่อป้องกันระบบหลักจากการโจมตีระบบจากภายนอก วิทยานิพนธ์นี้ได้นำเสนอหลักการแยกโมดูลระหว่างโมดูลหน่วยประมวลผล ออกจากโมดูลการประมวลผลวิดีโอ เนื่องจากโมดูลหน่วยประมวลผลนั้นสามารถทำงานร่วมกับการโปรแกรมประมวลผลแบบอื่นที่ทำงานในลักษณะการประมวลผลสายข้อมูลแบบต่อเนื่อง ผ่านการพัฒนาส่วนติดต่อคอมพิวเตอร์มาตรฐาน

ในส่วนของความยืดหยุ่นและการขยายตัวของระบบนั้น การจัดตารางงานเป็นส่วนสำคัญในการทำให้การใช้งานทรัพยากรการประมวลผลมีความคุ้มค่า งานวิจัยก่อนหน้านี้ซึ่งมีจำนวนไม่มากนักได้นำเสนอการวิเคราะห์ภาระงานการประมวลผลวิดีโอ โดยใช้บางคุณลักษณะของการประมวลผลวิดีโอสำหรับการวิเคราะห์การใช้งานทรัพยากรการประมวลผล การที่จะหาหน่วยประมวลผลที่เหมาะสมกับความต้องการ

ของการประมวลผลวิดีโอนั้นทำได้ยากหากไม่ทราบว่าการประมวลผลนั้นๆ ใช้ทรัพยากรการประมวลผลไปเท่าใด ในวิทยานิพนธ์นี้จึงอภิปรายรายละเอียดของลักษณะการประมวลผลวิดีโอโดยใช้คุณลักษณะของการประมวลผลวิดีโอต่างๆ เช่น ชนิดของการประมวลผลวิดีโอ อัตราเฟรมวิดีโอ ขนาดเฟรมวิดีโอ และคุณสมบัติของหน่วยประมวลผล ผลจากการศึกษาลักษณะของการประมวลผลวิดีโอได้นำไปใช้ในการออกแบบกระบวนการจัดตารางงานที่สามารถหาหน่วยประมวลผลที่เหมาะสำหรับการประมวลผลภาพวิดีโอนั้นๆ ได้ โดยที่สามารถทำงานได้ในทุกหน่วยประมวลผลที่มีคุณลักษณะแตกต่างกัน วิทยานิพนธ์นี้ได้เสนอให้มีหน่วยสำรวจการประมวลผลวิดีโอเพื่อรวบรวมข้อมูลของหน่วยประมวลผล โดยปรับเปลี่ยนปัจจัยต่างๆ ที่มีผลต่อการประมวลวิดีโอ ข้อมูลดังกล่าวจะถูกบรรจุในฐานข้อมูลเพื่อใช้สำหรับประมาณการทรัพยากรการประมวลผลของการประมวลผลวิดีโอที่เข้ามาใหม่ วิทยานิพนธ์นี้ได้นำเสนอวิธีการประมาณความต้องการทรัพยากรการประมวลผลของงานประมวลผลที่ต้องการทำงานใหม่โดยใช้ข้อมูลจากหน่วยสำรวจการประมวลผลวิดีโอและค่าปัจจัย CPU นอกจากนี้วิทยานิพนธ์นี้ได้นำเสนอเกณฑ์การจัดตารางงานเพื่อช่วยให้ผู้ดูแลระบบตรวจตราด้วยกล้องวิดีโอสามารถปรับแต่งการใช้งานทรัพยากรได้เหมาะสมกับระบบที่ดูแลอยู่

ACKNOWLEDGEMENTS

This thesis could not have fully come about without helps, supports and encouragements from the following people and organizations.

Firstly, I would like to express my deepest gratitude toward Assistant Professor Dr. Pichaya Tandayya, my thesis supervisor. Without her endless enthusiasm, motivation, generosity, understanding, and support, this study would hardly have been completed. Above all, my invaluable insight of life tremendously gained throughout the time of research and writing of this thesis.

Secondly, I would like to express my warmest gratitude toward Associate Professor Dr. Chokchai Leangsuksun, my co-advisor, for giving advices and identifying several issues in this thesis. I am appreciative of the reviews, advices and valuable comments of Assistant Professor Dr. Nikom Suvonvorn and Assistant Professor Dr. Worawan Diaz Carballo, the thesis committee, which have significantly improved this thesis.

I would like to thank all lecturers in Prince of Songkla University who endowed valuable knowledge and guidance during my study. I would like to acknowledge the valuable insights in several discussions about features of traditional video surveillance systems, future of video surveillance systems and their transformation applying cloud computing technology with Dr. Somchai Limsiroratana. I am thankful for precious advices and comments in many issues to improve my thesis from Assistant Professor Dr. Sangsuree Vasupongayya and Associate Professor Dr. Taweesak Reungpeerakul. Also, I am appreciative to Dr. Andrew Davison for English proofreading of several papers of mine.

This thesis would not successfully been carried out without the help from Mr. Touchai Angchuan and Mr. Pipat Pipitpattanapong for providing an account to IP cameras of Department of Computer Engineering, Prince of Songkla University. The explanation of RabbitMQ by Mr. Twin Panichsombat clarified the messaging pattern paradigms and I have applied it for the inter-component communication in the Nokkhum. Moreover, this thesis can extend the experimental results using servers from the Office of Academic Resources and Information Technology, Songkhla Rajabhat University, under the assistance of Mr. Kritwara Rattanaopas and Mr. Taofik Lamsub. Without they precious support, it would not be possible to conduct this thesis. Furthermore, I would like to thank many open source contributors who work hard and are confident in and devoted to their projects which make the world of computer programming so much easy to code. Their products reduce the complexity of my thesis and enable quick implementation in many parts.

Many thanks to my colleagues in Department of Computer Engineering for enjoyable and appreciated exchanges on various topics. Also, I would like to thank all staff in Department of Computer Engineering, especially Miss Bongkot Prucksapong for dealing with many documents and the conference traveling plan during my study.

I really appreciate the encouragement from my parents, sister, relatives, and friends, and their cheerfulness throughout the time of my study.

I would like to acknowledge my scholarship from the Thailand Research Fund and Prince of Songkla University through the Royal Golden Jubilee Ph.D. Program (Grant No. PHD/0047/2552). Also, I am grateful for the thesis research funding from PSU Graduate School. Both financial supports have helped me to compose a cloud computing testbed for testing the Nokkhum VSaaS system. I would like to acknowledge the funding from Faculty of Engineering and Department of Computer Engineering for my trip to participate in the international conferences.

Last, but no means least, although I could not possibly list them all, I would like to acknowledge the helps, supports, encouragements, and inspirations received from other nice people who have directly or indirectly helped me in many ways during these several years.

Thanathip Limna

CONTENTS

CONTENTS (Continued)

CONTENTS (Continued)

LIST OF TABLES

LIST OF FIGURES

LIST OF FIGURES (Continued)

LIST OF FIGURES (Continued)

LIST OF ABBREVIATIONS AND SYMBOLS

API	Application Programming Interface
AMQP	Advanced Message Queuing Protocol
AWS	Amazon Web Services
CCTV	Closed-circuit Television
CPU	Central Processing Unit
CVS	Cloud-based Video Surveillance
FPS	Frames per Second
HTML	HyperText Markup Language
HTTP	Hypertext Transfer Protocol
IaaS	Infrastructure as a Service
IP camera	Internet protocol camera
MOM	Message oriented middleware
NVR	Network Video Record
QoS	Quality of Service
RAM	Random-access memory
REST	Representational State Transfer
SaaS	Software as a Service
SRC	System Request Command
URC	User Request Command
VCA	Video Content Analysis
VM	Virtual Machine
VSaaS	Video Surveillance as a Service
VSS	Video Surveillance System

CHAPTER 1

INTRODUCTION

This chapter presents the overview information of the study. It consists of four main parts: the Introduction and motivation, the objectives of this thesis, the scopes of the study, and the contributions of the study.

1.1 Introduction and Motivation

Video Surveillance Systems (VSS) are widely used in many areas, including residential areas, offices, factories, colleges, and traffic control systems, especially for security reasons [1]. They are used for monitoring unusual events that endanger lives and properties. Sometimes, video records become evidences or proofs in the justice procedure. Currently, cameras and related equipment are inexpensive and can be easily installed. This simply causes a significant increase in home use VSSes. Although the home use VSSes can efficiently handle a few video cameras, it is difficult to manage when applying a large number of video cameras in a large organization, such as a university, a hospital, a stadium, and a transportation hub. Large organizations require a lot of video cameras to cover all their operating areas. Therefore, good preparation and administration are essential for implementing a system for such requirements. There are not only problems about deploying larger VSSes, but also the VSSes have to sufficiently and efficiently provide storage space for the required time period to store the involved video records. Any organization that plans to use a VSS has to prepare some budgets to support its requirements.

Currently, the VSSes support Internet Protocol (IP) cameras [2] which connect to a Network Video Recorder (NVR) or a generic computer for video processing. The video cameras management and video processing software are usually bundled with the video cameras or are sold separately. They usually are high-performance software, and can manage a lot of cameras and special video processing but this comes with a price. Normally, free video management software is specifically limited to certain types of cameras and it cannot combine cameras from different manufacturers. Furthermore, VSSes must have appropriate storage space corresponding to the time period to store the video records involved. Each video record from a camera usually requires a lot of storage space. Consequently, if anyone wants to set up a VSS, one has to pay for video cameras, processing units, additional storage space, first-time installation, computing software, administration, and maintenance costs (to validate and verify the system availability). In order to reduce some costs, whoever want to set up their own VSS can rent a video surveillance service in forms of Software as a Service (SaaS) over the Internet. Currently, the characteristic of video cameras has changed from analog to digital as IP cameras and SaaS on the Internet are widely spread in the market. Therefore, there are attempts to transform VSS models into Internet services and provide video surveillance services to many users, namely Video Surveillance as a Service (VSaaS). Such services can solve the storage space problem, supporting a lot of video processing tasks and being able to elastically increase the number of video cameras. The users only have to pay fees according to their requirements. Although there are several video surveillance services in the market such as SecurityStation [3], Secure-i HVR™ [4], and OVS™ [5], the technical issues behind the scenes to provide such services, for example, the software architecture, resource management and cost-effective optimization, are still not disclosed.

VSaaS generally supports online video recorders, surveillance video processors, and online video storage. The main advantage of VSaaS is that the users can be less concerned about system software/hardware maintenance, and instead focus on providing and managing IP cameras. Also, the users can choose new video processing solutions as soon as the provider deploys them on the system, without system reinstallation. In addition, provider competition reduces operational costs and influences on best-quality video processing provision for the users. Most current VSaaSes provide simple video processing, such as motion detection for specific areas of interest, storage space, and an alert system. Unfortunately, there is a lack of information about the employed technologies and VSaaS architectures due to trade secrets. Also, some open video surveillance systems are not suitable for SaaS models because they are designed to support specific applications. These include car tracking in parking areas and traffic monitoring, which may not be applicable to home or residential areas. Moreover, it is difficult to modify some types of software to run in a distributed manner to support SaaS due to the tightly-coupled cooperation among the software's components.

Recent works suggested the architectures of which their VSaaS systems were implemented based on cloud infrastructure and big data technology, i.e., Amazon AWS [6], and Hadoop [7], as Cloud-based Video Surveillance (CVS) systems. For instance, a Hadoop MapReduce distributes image processing tasks and employs the Hadoop Distributed File System (HDFS) for storing video records. Examples included P2PCloud [8], VAQACI [9], and the cloud video recorder (CVR) system [10]. Another approach used Amazon AWS employing Virtual Machine (VM) applying Amazon EC2 and cloud object storage using Amazon S3 [6] as described by Hossain et al. [11], [12] and Rodriguez-Silva et al. [13]. However, these researches did not reveal deployment information, such as how the components were placed, how to scale the computing units, the minimum number of servers that could be deployed on the system, the front-end component arrangement for user access, and the provision of the video/camera management services. In addition, some researches presented resource allocation techniques and algorithms for minimizing the number of VMs on cloud infrastructures, including works done by Nan [14], Miao *et al.* [15], and Hossain *et al.* [11], [12]. Their proposed methods supposed that video analysis workloads consumed static computational resources. Therefore, the VM providers provided the same VM capacity and capability for all types of workloads. In reality, the VSaaS system has to support different video analysis workloads which are combined from any customers. Consequently, the resource allocation for VSaaS must consider factors involving the type of video analysis, video frame size, and frame rate.

According to resource allocation for VSaaS, video processing tasks consume different computing resources according to their kind of video processing, video frame size, and frame rate. Video task processing on different computing machine specifications involves different computing resource consumptions according to different CPU vendors, CPU architectures, and CPU frequencies. It is hard for the video processing task scheduler to accurately predict computing resource consumption as different tasks involve different data input resulting in various types of video processing analysis. The suitable methodology for estimating different computing resource consumptions is to run a video processing testing suite to collect the related consumption results. The information can be concluded as a heuristic for video processing task scheduling. The heuristic can also be automatically adjusted to a particular computing resource pool by rerunning the testing suite on the appearance of new machines.

The objective of this thesis is to purpose a VSaaS system which works on Infrastructure as a Service (IaaS), focusing on software system architecture, system behaviors, and user requirements. The system architecture is based on traditional cloud services: the compute service employs Amazon EC2, and the object storage service utilizes the Amazon S3 Application Program Interface (API). The VSaaS can easily add new computation node workers to support the ever increasing number of IP cameras, and also saves time and energy in planning and controlling the system performance. In addition, this thesis presents video processing workloads and task scheduling running on both physical machines and VMs on a cloud infrastructure. The proposed scheduling method considers different video analysis configurations from various VSaaS users on heterogeneous computational units. This workload analysis and task scheduling aim to provide that the system can place a video processing task on a suitable compute node.

1.2 Objectives

1.2.1 To propose a system architecture for VSaaS running on both physical machines and VMs that can serve a large number of users on the Internet, and handle many IP cameras. The users can add or remove IP cameras to and from the VSaaS.

1.2.2 To propose a video processing task scheduling based on actual computing resource consumption from video processing workloads. The resource predictor module considers several factors including the kind of video processing, compute node specification, and video frame size and rate.

1.3 Scopes

1.3.1 This thesis involves generic video processing found in residential areas, such as motion detection and video recording. It does not consider the video processing for specific functional areas.

1.3.2 The proposed VSaaS is tested on a private IaaS system which is built from five generic PCs using the OpenStack software suite.

1.3.3 This thesis assumes that the IaaS provider or private cloud system can provide unlimited storage space to store video records.

1.3.4 The proposed VSaaS architecture works with a cloud infrastructure employing Amazon AWS, especially Amazon EC2 and Amazon S3.

1.4 Contributions

1.4.1 Presenting the new component-based design to promote the scalability and flexibility of VSaaS which easily scales the VSaaS services for supporting different sizes of IP cameras and user requirements as follows:

(a) Encapsulating the functionalities of each component.

(b) Running on both physical machines and VMs provided by an IaaS cloud provider.

(c) Separating the VM layer to the VSS layer by using components.

(d) Deploying scenarios based on seven topology configurations.

(e) Using open-standard, human-readable JSON file format to configure the components for various usage scenarios.

1.4.2 Proposing the task scheduling techniques to ensure the scalability of the system that can support large numbers of users and IP cameras.

1.4.3 Presenting a video processing tasks resource consumption predicting module to ensure that the scheduler can place the required tasks on suitable compute nodes.

1.4.4 Conducting experiments and summarizing the results to assess the flexibility obtained from the component-based design.

1.4.5 Evaluating the proposed task scheduling techniques to ensure the scalability of VSaaS can support the required tasks when increase the numbers of users and IP cameras.

CHAPTER 2

LITERATURE REVIEWS

This chapter is to provide background information applied in the thesis including video surveillance systems, cloud computing, and cloud-based video surveillance. The video surveillance section describes current video surveillance systems and modern video surveillance technology. The next section introduces cloud computing and moving traditional video surveillance to the cloud. Finally, the last section concludes all literature reviews related to video surveillance as a service system design.

2.1 Video Surveillance Systems

Video Surveillance Systems (VSS) currently employ two types of cameras: closed-circuit television (CCTV) and IP cameras. Many CCTV systems do not only record video but can also perform functions such as motion detection and recording motion video footage. However, CCTV systems have limitations concerning camera usage distance, storage capacity, installation and reconfiguration problems when adding new cameras or new Video Content Analysis (VCA) algorithms. IP camera systems avoid these restrictions with longer working distances and wider areas. They can easily be plugged into any computer network in order to distribute data across the Internet. IP camera systems commonly provide computers capable of VCA and recording. The systems can be scaled up to support many IP cameras, and offer new or sophisticated VCA features. Currently, VSSes are moving away from standalone applications toward software as a service for providing a large computing unit that supports a larger number of users. In this section, this thesis compares traditional and modern video surveillance systems which motivate the improvements proposed in the design.

2.1.1 Traditional Video Surveillance Systems

Current VSSes are employed in many applications [1], [16], [17], [18], [19]. They have been used in wide areas and have processed videos from many cameras. Traditional VSSes supporting IP cameras [2] were often connected to a Network Video Recorder (NVR) or a generic computer to provide VCA. Home use VSS software including the video camera management and VCA were usually bundled with IP cameras. The user had to install system software on a computer which allowed it to connect to the IP cameras. The vendor software was limited to their types of cameras and could not interface with cameras from different suppliers. Also, high-performance software that could manage a larger number of cameras and provide special VCA was usually sold separately as additional software. Usually, this kind of software was very expensive and required complex configuration. Anyone wants to set up a VSS would incur many expenses apart from the cost of the IP cameras, including the expenses of first-time installation, network equipment, maintenance and additional computing software.

In typical VSSes, the data was collected from video cameras and delivered to video processing units. A typical flow of the video processing units in VSS started with object detection and ended with storing the information into a database as shown in Figure 2.1 [1]. Many systems required specific configurations. They had certain limits according to the manufactured cameras. Some video processing tasks required complex computation. Therefore, many systems tried to reduce the computation time on complex system architectures.

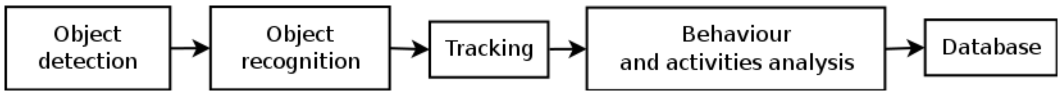


Figure 2.1: A typical flow of video analysis in a surveillance system.

Figure 2.1 shows a typical flow of video analysis in a VSS. There were many video analysis systems, applied to surveillance applications, proposing a flexible architecture, reconfigurable, and failure prevention. Many researches proposed component based architectures due to flexibility, easy configuration and development. Most development frameworks were, however, standalone applications and could not automatically handle a large number of cameras.

2.1.1.1 Video Surveillance Software

Generally, home use VSSes have been developed for a single computing unit including IP camera setting, video record management, and simple video processing. They can handle about two to ten cameras according to their computer capability. However, this kind of architectures can scale the system up for supporting video streams sent from a lot of cameras. In this method, it typically assigns that the video streams go into the computing node as shown in Figure 2.2. If there is any failure in a computing node, the video processing in the node will be stopped until the crash has been fixed. In this situation, the information would be lost. In case of video processing that requires high accuracy, it may affect other parts of the system process. This implementing pattern costs much lower than that of a large-scale software architecture, but it needs an administrator with a high level of skills for system deployment, management and maintenance.

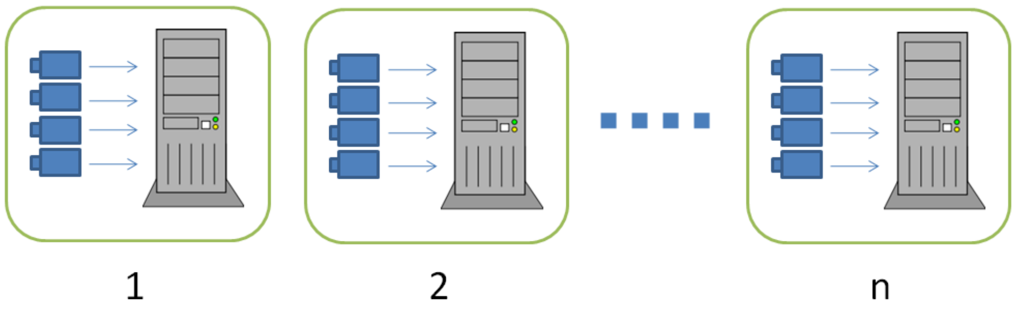


Figure 2.2: Current video surveillance architecture.

There are many free and open source video surveillance software running on standalone computers, such as ODViS [17], OpenVSS [19], and Zoneminder [20], which provide video camera management, simple VCA, video playback and video record storage. Most software are easy to install, configure, and use. However, their software components are rather atomic and hard to scale. One solution is to add a new computing node and re-configure the VSS. However, the number of cameras continuously increases and VCA configurations vary as for different vendors. It, therefore, is difficult to manage and maintain the expanding system with heterogeneous devices.

The other type of VSSes is a distributed system or a client/server model with a processing control unit and a video processing subsystem. The server distributes works to other computing nodes in order to enhance the computing performance and reduce the failure of network communication. Some VSSes have not been designed for scaling or preventing failures. Most system designs are mainly aware of the camera cost. Generally, a VSS is a standalone application running on a single physical desktop. Therefore, it cannot distribute works to others. The system development mainly focuses on enhancing the performance of the video processing algorithms while trying to reduce the system resource usage as much as possible and also provides cooperative video stream processing. Some systems focus on processing multiple cameras on a single physical machine and lack of system flexibility. There are several softwares proposing this kind of VSS, utilizing a software architecture consisting of many components and processing layers, such as IVSS [21], DiVA [18], CANDELA [22], S-VDS [23], and SSF [24]. These distributed architectures are designed to support a large number of cameras and various configurations. The components can be distributed across many computers to provide higher performance VCA and system availability. However, the architecture must be carefully configured as it includes different modules running on separate machines, and requires tightly-coupled work cooperation. Moreover, these kinds of systems require a specialist to determine which components need to be scaled up.

Exploiting the Internet, some video surveillance software provides online system access for video camera monitoring, video storage, and software configuration. Examples include ViSOR [25], S-VDS [23], and MoSES [26]. Although, this makes it easier to access the system, users must prepare a large amount of budget for hardware, software, and service charges.

2.1.2 Video Surveillance as a Service

Online video storage enhances traditional VSSes by increasing user convenience, and has influenced research on remote video surveillance recorders and VCA on the Internet [27]. One advantage of this architecture is that it can serve any IP camera from any location to users anywhere across the Internet. Users do not have to own recording and storage hardware. Instead, they can just rent an NVR or a VCA from a video surveillance provider. The provider offers an Internet based recorder and storage space via a service package which matches the user's requirements. However, it is hard for the video surveillance provider to predict the required supply of computing and storage hardware for on-demand processing.

Basically, Video Surveillance as a Service (VSaaS) is a VSS in a cloud environment for supporting users located at different places as shown in Figure 2.3. In Figure 2.3, the users register and record their IP cameras to the VSaaS system, and also pay the service charge over a billing period. VSaaS providers provide several facilities for managing the cameras, video processing, storage, and billing. The VSaaS system can scale the video processing part for supporting various, dynamic, and a lot of video content analysis for users. Therefore, several VSaaSes have been deployed on cloud infrastructure using any cloud services, especially elastic computing units and object storage. VSaaS systems can request for dynamic computing resources and can easily scale the system up according to the system requirement.

The main advantage of VSaaS is that the users can be less concerned about system software/hardware maintenance, and instead focus on providing IP cameras. Also, users can choose new video processing solutions as soon as the provider deploys them on the system, without system reinstallation. In addition, provider competition reduces operational cost and provides best-quality video processing for the users. Most current VSaaSes provide simple video processing, such as motion detection for specific areas of interest, storage space, and an alert system. Unfortunately, there is a lack of information about the technologies and VSaaS architectures due to trade secrets. Also, some open VSSes are not suitable for SaaS models because they are designed to support specific applications. These include car tracking in parking areas and traffic monitoring, which may not be applicable to home or residential

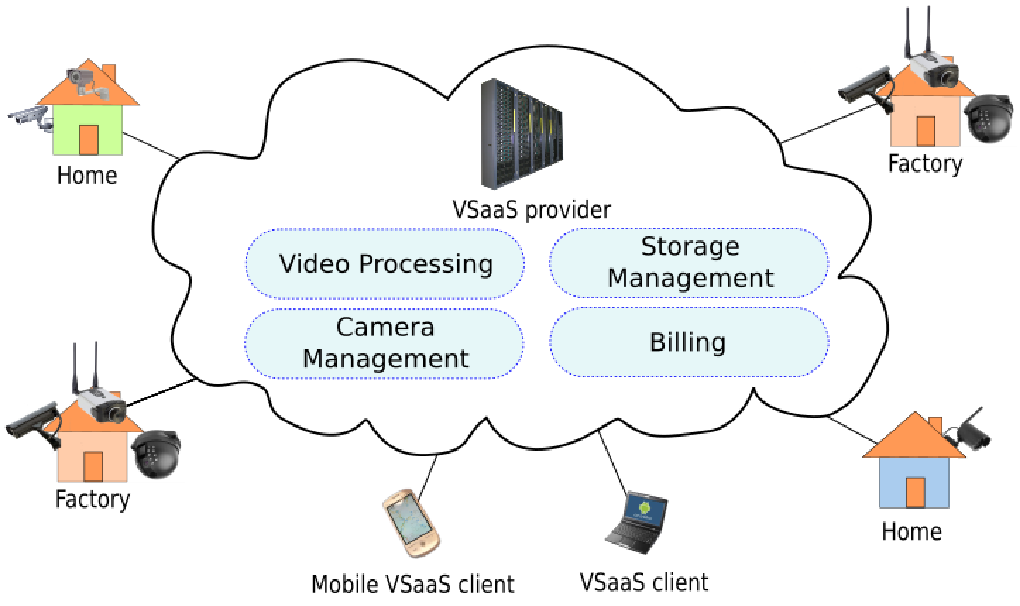


Figure 2.3: Overview concept of VSaaS

areas. Moreover, it is difficult to modify some types of software to run in a distributed manner to support SaaS due to the close cooperative work among the software's components.

A new video surveillance platform has emerged from cloud computing technology, especially IaaS such as Amazon Web Service (AWS) [6], Hadoop, and virtualization technology. It provides more system availability, security, reliability, and maintainability for CVS than traditional software [28]. Cloud technology changes remote video surveillance into VSaaS which is a SaaS model. This kind of architectures solves problems concerning limitations of computing resources, such as storage space, computing unit, and network management. Cloud computing enables a remote video surveillance provider to start with a small VSS which can later grow according to the users' demands. Consequently, the provider does not need a high budget for starting the business and does not need a complex hardware supply plan for supporting user requirements. Current VSaaSes in the market provide an NVR, a simple VCA, and storage space for collecting video records. The VSaaS providers offer various service plans of which their customers choose based on their requirements. Examples of VSaaS currently provided in the market include SecurityStation [3], Secure-i Hosted Video Recorder™ [4], and OVS™ [5].

Many CVS systems can be classified into two groups by the cloud technology they use. The first group implements their video surveillance architecture based on Hadoop technology which includes MapReduce and the Hadoop Distributed File System (HDFS). Example systems of this group are P2PCloud [8], VAQACI [9], and CVR systems [10]. They utilize the HDFS to store video records and to provide high performance video retrieval. In addition, MapReduce can automatically distribute processing tasks to Hadoop computing nodes. This ability means that a large number of video records can be distributed as VCA tasks to Hadoop nodes.

The second group implement IaaS by employing Amazon AWS as a video surveillance tool. Examples of this group include the systems developed by Hossain *et al.* [11], Rodriguez-Silva *et al.* [13], and Chen *et al.* [29]. Normally, this kind of architectures uses the Amazon Simple Storage Service (Amazon S3) to collect video records and the Amazon Elastic Compute Cloud (Amazon EC2) to provide VMs as computing units.

Both groups of CVS systems transform the tasks of video surveillance applying cloud-based technology as the software does not depend on any development framework. However, the architecture designs in [11], [13], and [29] are rather general. For example, it is unclear how issues are handled such as the compute node number for the first-time installation or component distribution during deployment and system scalability.

Previous works [11], [12], and [14] as focused on CVS system design, mainly utilized Amazon EC2 and S3 API. Most works were interested in optimizing resource allocation especially VM such as works proposed by Hossain *et al.* [11], Nan *et al.* [14], Hossain and Song [30], Yang *et al.* [31]. Hossain *et al.* [11] presented a VM allocation scheme for supporting video streaming for emergency officials. Nan *et al.* [14] proposed a cost effective resource allocation optimization approach for multimedia cloud that was based on a queuing model. Hossain and Song [30] presented a VM resource allocation model to satisfy Quality of Service (QoS) requirements for CVS. Yang *et al.* [31] proposed a mapping approach for placing heterogeneous VM instances into heterogeneous physical machines based on video surveillance workloads. Hossain *et al.* [12] described a VSS framework for dynamic workloads such as face detection and storage tasks. Alamri *et al.* [32] and Hossain [33] were interested in QoS for distributed video surveillance services, especially video transcoding. They proposed a VSS and a service configuration algorithm based on video transcoding workloads. Hossain [33] described video workloads involving 320x240 pixels at 30 frames per second (FPS). Alamri *et al.* [32] presented computing resource consumption for surveillance video streaming and video repurposing/transcoding, but without any information about video workloads. Unfortunately, there have been few video processing factors and workload characteristics which identify the computing resource consumption.

Other interesting works involved CVS as follows: Miao *et al.* [15] presented an optimizing resource allocation for cloud-based rendering between cloud and mobile phones. Song *et al.* [34] proposed a parallel encoding and queuing analysis for remote display of video surveillance desktop in cloud environment. Sharma and Kumar [35] proposed an architectural framework for human activity recognition in CVS environment. Chen *et al.* [36] presented a video analysis service for integrating into the existing CVS, the implementation is available in the CityEyes system [29].

Deploying VSaaS in a cloud computing environment involves VSaaS system design and computing resource management, two issues which affect the quality of service. Video task scheduling involves symmetric video analysis workloads, including the frame size and rate, and video analysis type, running on a homogeneous computing units. Such VSaaSes display high resource efficiency when everything is static but these settings restrict the flexibility of the VSaaS's capacity, and do not alway support user requirements. Furthermore, when some computing hardware is replaced, the infrastructure will be changed from a homogeneous to a heterogeneous computing system. Therefore, flexible resource management is needed in VSaaS systems, especially for task scheduling handling dynamic user requirements and heterogeneous computing units.

2.2 Cloud Computing

Cloud computing [37] [38] is a popular computing model to support large volumetric data [39], and focuses on the concept of user request services and the pay-as-use schema. There are many definitions for cloud computing. However, it is well known as everything as a service. There are different categories of cloud services. The most currently interested and generally served has three layers which are SaaS, Platform-as-a-Service (PaaS), and IaaS as shown in Figure 2.4 [39]. The IaaS is the kind of services in cloud computing, which provides VMs for customers. The customers do not own physical hardware but can access the VMs which work as real servers via the Internet. According to IaaS, a VSS runs on IaaS can easily scale the processing resources according to the customers' requests. The video surveillance providers who choose to provide their system on cloud computing can pay for the VM resources they use only. Therefore, this approach can reduce the startup cost (computing hardware). They can also focus on only the software development.

In Figure 2.4, the IaaS provides a highly flexible computer infrastructure to other layers such as VMs, computer storage and network. Customers can pay for a flexible price as their companies grow relating to their applications' time to market. This kind of services has significantly increased as observed from Amazon EC2, GoGrid, Rackspace, etc. The SaaS delivers software over the Internet. It is a modern distribution software model. The user can access most applications through websites, for example, Gmail, Facebook, etc. It also has centralized features or maintains security updating.

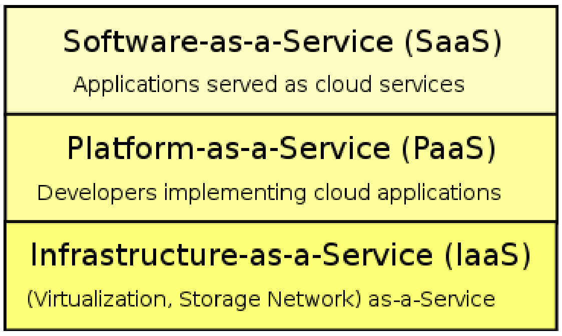

Software-as-a-Service (SaaS)
Applications served as cloud services
Platform-as-a-Service (PaaS)
Developers implementing cloud applications
Infrastructure-as-a-Service (IaaS)
(Virtualization, Storage Network) as-a-Service

Figure 2.4: Cloud computing layered architecture.

In case of changing the VSS served on the Internet, customers do not need to care about software maintenance and can choose new video processing solutions as soon as the provider deploys them without new system installation on the user side. The customer can care less about software and hardware maintenance, and focus on providing IP cameras only. On the other hand, the providers may compete with each other by reducing the operational cost and provide best quality video processing to customers. Therefore, the providers have to optimize their video analysis system to consume little electricity power and provide cost effective computer resource utilization, and meanwhile, keep the QoS up to the customers' satisfactory.

2.2.1 Video Surveillance in the Cloud Environment

Currently, cloud computing has been well known as an information technology platform. Some companies have moved the video surveillance architecture to run on cloud computing in forms of a SaaS model shown in Figure 2.3 and charge the users by calculating from the storage amount and the number of IP cameras used. As shown in Figure 2.3, the users can plug-in video cameras to the Internet at any place. VSaaS concerns capturing video streams from IP cameras, encoding and storing them in the cloud storage, and charging the user for the related storage space package. Providing video surveillance services on cloud computing needs to be aware of the QoS, performance management and consolidation problems.

2.2.2 Video Surveillance as a Service Over Infrastructure as a Service

VSaaS facilities are for providing services with elasticity properties while techniques for exploiting system resources are not publicly apparent or clearly explained. For current video surveillance architectures, it is the system administrator's job to provide sufficient computing resources for the system request. However, this approach may not always be appropriate. The system often uses computing resources inefficiently, and it is rather expensive (incurring hardware, storage and electricity costs).

The current trend for cloud computing services is that users popularly use IaaS. The evidence can be observed from the requirements for VMs in Amazon EC2 or Virtual Private Server (VPS). The customer does not need to own any physical machine, instead they can own VMs on the Internet. Considering the elasticity of the cloud computing characteristics, transforming the current video surveillance software onto SaaS and running on IaaS will allow the system to easily start a VM with required resources.

Although Neal and Rahman [40] [41] showed higher pricing in deploying VSaaS on public IaaS, current results show that the total cost of cloud services is less expensive than purchasing and deploying the hardware locally. Also, they proposed a cost effective scheme by combining two separate cloud computing vendor solutions together to take advantage of available pricing. Fully, deploying VSaaS on public cloud is quite expensive. VSaaS deploying cloud technology is necessary for supporting technology transformation. In addition, many organizations continuously change their infrastructure to a cloud computing platform or a private cloud service. The VSaaS system can be deployed on private cloud to reduce the hardware complexity. Some organizations gradually change their infrastructure to a cloud platform, so that at time there can be hybrid infrastructures including cloud and legacy physical servers. Therefore, in real usage the VSaaS architecture should be able to adapt the design for supporting both cloud technology and physical hardware for seamless deployment.

2.3 Summary

VSaaS is a new platform for modern VSS that provides video surveillance service using cloud computing technology. It reduces hardware and maintenance cost, and is friendly for general users who seek for home VSS. Although, there are many VSaaS systems designed for supporting a large number of users. They still lack of information about technical system design, scalability, and deployment topologies. The other issue for deploying VSaaS is video task scheduling. The scheduler essentially responds to different configurations of video processing tasks and places them on suitable compute nodes. This issue received less concern on many pieces of research, but it is essential and difficult to handle various tasks on the VSaaS. The VSaaS system design and video task scheduling are important issues being focused in this thesis.

CHAPTER 3

SYSTEM DESIGN AND IMPLEMENTATION

This chapter presents details of the proposed VSaaS called the "Nokkhum" (a Thai word for "Asian quail") system. First, it introduces the Nokkhum design concept addressing a scalability, flexibility and specification of the proposed VSaaS. Second, it presents the Nokkhum system design describing the architecture definition, components, and system scalability. Third, the analysis of video workload characteristics which identify the video processing workloads considering several factors. Fourth, it proposes Nokkhum scheduling using the analysis results in the scheduling design to place the required video processing task in a compute node. Finally, the summary section summarizes all presented issues.

3.1 Nokkhum Design Concept

According to the VSaaS paradigm, a provider offers video processing and storage service to users. The users must register their information for opening an account with the provider and can then use VSaaS services provided by the VSaaS system via the Internet. The provider must prepare the VSaaS system and provide services which are available to users at all times. The system must handle dynamic user requests and start the required video processing processes when and where it is appropriate. Many video surveillance systems are designed to support a single organization with groups of users, but VSaaS tends to be more complex by including various organizations and groups.

The benefits of cloud computing such as reliability, scalability, dynamic resources provision, and quick deployment, enable VSaaS providers to provide their services exploiting minimal computing units, and to dynamically scale the computing units according to user requirements. This motivation is the major influence for designing the proposed VSaaS architecture utilizing the elastic computing and storage service from the cloud platform.

Public cloud services currently are quite expensive for providing such a VSaaS system to wide area users. Also, it is difficult to deploy the VSaaS to perform better than the breakeven point. However, the advantages of cloud computing technology help to deploy applications easily, and also many organizations are on progress in transferring from the legacy physical computing units to apply a private cloud platform. Nowadays, system software cannot avoid the cloud technology involvement in designing the VSaaS. Therefore, the proposed VSaaS system has to support both cloud computing and physical machines for a smooth transition in the current stage of technology.

The proposed Nokkhum architecture is a VSaaS system that can runs on top of a cloud infrastructure. Both the system size and the business size can be easily scaled to match the user requirements. The VSaaS design is divided into five components for supporting scalability and flexibility, and can simply distribute all components to different sizes of machines. Moreover, the architecture is designed to run on physical machines if necessary, in order to offer the best video processing performance, a small VSaaS system, and hybrid deployment.

3.1.1 Video Surveillance as a Service Design Specification

This section presents a specification for designing the Nokkhum VSaaS. The expected VSaaS system is to be designed to fulfill the following requirements:

1. The proposed system can support multiple users and different organizations.
2. The system can be deployed on physical and virtual machines.
3. The system can run on IaaS by implementing AWS EC2 for elastic computing and AWS S3 for object storage.
4. The proposed system can be deployed on a single machine and extended to a large scale system of which components are distributed to multiple machines.
5. Focus on to deploying the system on a heterogeneous machine specification and the compatibility of legacy computer systems.
6. All VMs utilize Kernel-based Virtual Machine (KVM) [42] as a virtualization infrastructure.
7. Study on three video processing tasks including video recording, motion detection, and motion recording, employing OpenCV [43].
8. The proposed system can support a variety of video formats according to user requirements.
   - Eight frame rates: 1, 5, 7, 10, 15, 20, 25, and 30 FPS.
   - Six frame sizes: 160x120, 320x240, 640x480, 800x600, 960x720, and 1120x840 pixels.

- Support recommended video input/output by SWGIT [44] at the resolution of 640x480 pixels, and the frame rate of 30 FPS.
- Ogg codec [45] is used as a default codec for video output.

9. The video processing task scheduling design is based on first-come, first-serve (FCFS) scheduling.
10. The scheduling applies real computing resource consumption in heterogeneous machines.
11. The VIRAT Video Dataset [46] is employed to explore the resource usage of video processing and to verify the scheduling results.

3.1.2 Nokkhum Scalability Design

The scalability of the Nokkhum VSaaS architecture is designed by using the following assumptions:

1. The video surveillance system can run on both physical servers and a cloud IaaS.
2. The architecture automatically acquires VMs to support the customer's changing requirements.
3. The VSaaS provides a private cloud for an organization and/or a public cloud for user groups.
4. The VSaaS fully supports IaaS with the Amazon EC2 and Amazon S3 APIs, and also is suitable with a system that uses only a physical server or more.
5. The VSaaS starts as a small system, and can easily scale up to match dynamic user requirements.
6. The design aims for a system that utilizes a broadband network throughout.

Following the above assumptions, Nokkhum VSaaS is a highly scalable architecture composed of five components. The controller manages system resources and scheduler for passing video processing tasks to a suitable compute nodes. Consequently, each compute node includes a system resource reporter and a video processing task starter to help the controller monitor the computing resource pool and to start the task according to the controller requirement consecutively. Message oriented middleware (MOM) [47] [48] is employed to exchange information among the main distributed components using publish–subscribe pattern, and allows the components to be distributed across different computing platforms (both generic personal computers and VMs).

3.1.3 Nokkhum Flexibility Design

The MOM increases system availability, because if some components crash, the other components in the system can still continue running. Nokkhum VSaaS is designed to flexibly tolerate difficult scenarios such as:

1. When the electricity is cut off, or the network is out of service at a video camera site, the system can automatically recover after the camera becomes available again.
2. If a compute node worker has a problem, e.g. the network is out of service, a program crashes or a message broker disconnects, then the system can provide an alternative compute node worker.
3. If the Nokkhum controller does not respond, but the compute node workers are still running, then the video processors can continue working. When the controller resumes, it can recover its previous states by processing the information collected in the message broker server and database server.
4. If the message broker server does not respond, then the compute node worker, and the controller can continue running with the last available information. After the message broker server resumes, the compute node worker and the controller will start their communication again.
5. The system can start on a single machine and scale up by adding more worker machines.

3.1.4 Summary of Nokkhum Design

The Nokkhum VSaaS system has been designed for a flexible, scalable component-based architecture which can deploy the components on both physical and VMs running IaaS using the Amazon AWS API (EC2, S3). It has been designed and developed to support video applications for various organizations and business sizes, where the deployment and configuration involve a flexible range of computing units. It has an API interface server providing a REpresentational State Transfer (REST) over Hypertext Transfer Protocol (HTTP) [49] which supports any client platform, and is able to control, view, or manage video analysis, video recording, and camera configuration.

The Nokkhum components begins when the user composes a video processing configuration using the web interface, and submits it to the API. After the API has validated the configuration, it stores a video processing command in the database, which the controller later reads. It finds a suitable computing node which connect to the others through the MOM. The suitable computing node starts a video processor with the configuration from the user which processes the video stream following the user's configuration. All output from the processor is immediately pushed to cloud storage. In addition, the user can manage the status of the video processing and playback video records via the web interface.

Nokkhum's scalability benefits from the MOM which enables it to distribute components to many servers. This thesis supposes that VSaaS may be provided in the cloud environment employing different computer specifications for video processing. In order to do so, the video processing task scheduling has to be designed for supporting heterogeneous computing nodes which are connected to the Nokkhum system. The scheduling method has to relate on video processing workload analysis and resource estimation on the Nokkhum VSaaS. The proposed method is based on video processing task exploration involving different frame rate and size, type of video processing and computer specification. The task exploration collects and records CPU and memory usage in a database for the scheduler to utilize it in computing resource estimation for allocating a new video processor task to a suitable compute node. When there is a new compute node in the VSaaS system, a video task scheduler many inefficiently place a new video processing task on a new compute node. This could cause video frame drops, or processing task crashes or termination either with or without an error due to insufficient resources. Therefore, the scheduling design has to estimate a new video processing task based on the existing experimental results.

3.2 Nokkhum Component-based System

This section presents a component-based video surveillance architecture including five components for providing a VSaaS on both physical servers and VMs. Nokkhum system can deploy the components on a hybrid system of physical servers and VMs via the Amazon EC2 API and also Amazon S3 API for image and video record storage. This method engenders system portability so that the VSaaS system is suitable for different business sizes. In addition, an API server supports interfaces from any client by utilizing a REST style architecture via HTTP. In summary, this thesis proposes a flexible and scalable system architecture, and components design and implementation, for providing VSaaS on physical server and IaaS.

Nokkhum VSaaS consists of five components (see Figure 3.1): controller, compute node worker, video processor, API, and client (web interface). VSaaS users can access and control their video processing tasks via a web interface connected to the API server. The controller is a daemon process that handles user requests from an API server and directs commands to available compute nodes using the MOM. The compute node worker receives commands from the controller via a message broker, manages its video processors, monitors its resources, and reports status details back to the controller. In this thesis, the video processors are implemented using OpenCV.

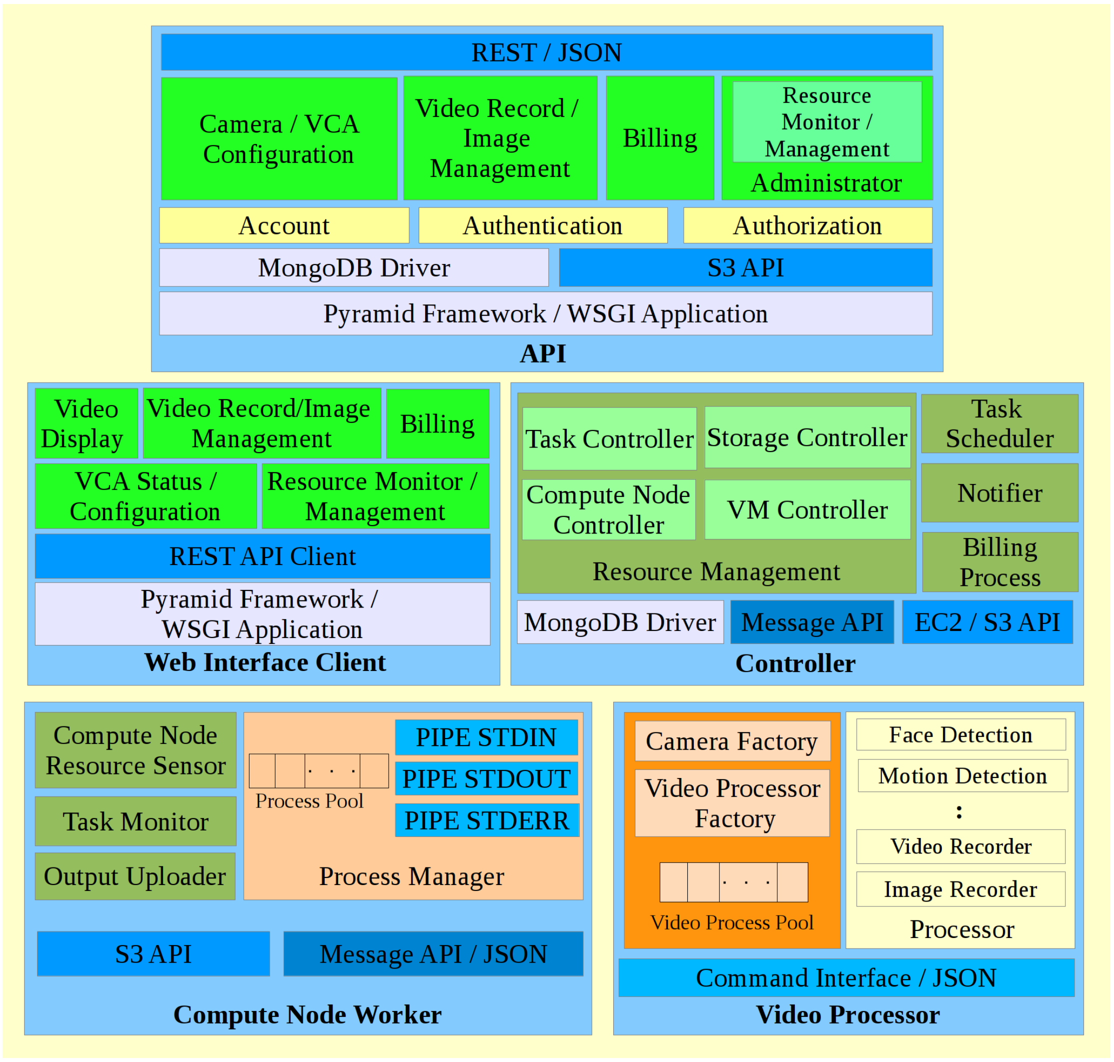


Figure 3.1: All components and modules of the Nokkhum VSaaS.

3.2.1 Nokkhum Components

Nokkhum consists of five components developed with C++ and Python as shown in Figure 3.1. Each offers specific functions and work cooperation for enabling system availability. The following sections describe the five components, namely the Nokkhum controller, Nokkhum compute node worker, Nokkhum video processor, Nokkhum API, and Nokkhum client shown in Figure 3.1.

3.2.1.1 Nokkhum Controller

The Nokkhum controller is a daemon process made of multiple modules and sub-controllers which perform many tasks as shown in Figure 3.2. The most important module is resource management, which is handled by four sub-controllers:

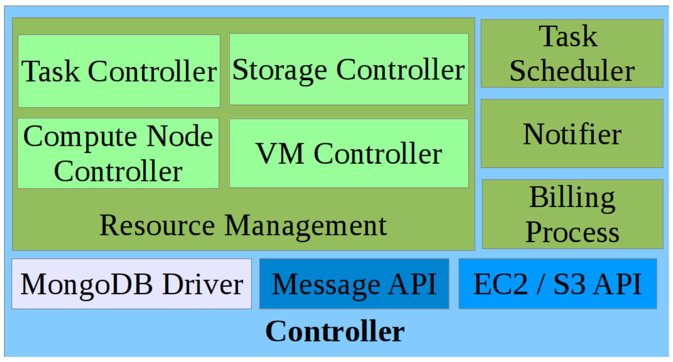


Figure 3.2: Modules and sub-controllers of the Nokkhum controller.

- **The task controller** is a central video processing controller which handles VCA tasks via the task monitor and task manager. It deals with a new task as to start the task in a suitable compute node. The task monitor checks all running video processor tasks. If one does not respond, then the task monitor will send a command request to restart the processor task. The task manager provides an interface for controlling video processor activities and is mainly used by the task scheduler.

- **The compute node controller** is a compute node worker manager which processes all the resource status reports. The data is delivered from Nokkhum compute node worker sensors via the message broker server. A resource status report includes total CPU usage, total memory usage, and total hard disk usage of a particular compute node worker. It also includes video processing resource usage data, such as CPU usage, memory usage, important video processor results, and the availability status of the video processor. If the resource status report shows that a video processor is unavailable, then the compute node controller will call the task controller to restart the video processor. Besides, when the task scheduler or the VM controller requests for a compute node worker's resource information, the compute node controller will provide that information, and predict the resource usage for all the compute node workers. Currently, this prediction utilizes a Kalman filter [50] applying to the last 20 compute node resource status reports. The compute node controller supports both physical servers and VMs.

- **The VM controller** enables Nokkhum to control VMs using the Amazon EC2 API. It acquires a resource prediction from the compute node controller to decide whether to terminate a VM or to start a new one. The VM controller will acquire a new VM when there is a command waiting in the processor command queue, and there is no computing resource to handle it.
- **The storage controller** manages video records and images. Part of its duties are to remove expired video records of any video processor.

Apart from resource management, the Nokkhum controller also includes a task scheduler, a notifier, and a billing process module:

- **The task scheduler** allocates video processors or VCA tasks to the most suitable Nokkhum compute node workers. It acquires resource information from the compute node controller and starts video processing tasks via the task controller. This module has a command waiting for the queue that collects processing commands from the user and recovers commands from the task controller.
- **The notifier module** provides alert messages to users those have activated this module via a video processing task. This module is activated when the compute node controller receives a resource status report that includes a notification message.
- **The billing controller** handles billing processing for the Nokkhum VSaaS.

The Nokkhum controller deals with the users' video surveillance requirements and calls compute node workers to start surveillance applications via a message passing method. It starts the work by retrieving video processing action commands from the database and directs them to available compute node workers. In this way, it can start and stop video processing tasks, and check video processing status details. The controller directs message commands to computer node workers using the message broker server. The compute node workers then build the related video analysis processes.

3.2.1.2 Nokkhum Compute Node Worker

A compute node worker is a daemon process that runs on each computation node. It provides computer resource monitoring, video processing task monitoring, and video processing deployment interface as shown in Figure 3.3. The compute node worker uses PIPEs [51] for communicating with the video processors and reports information resource updates to the controller using message passing. The compute node worker consolidates the video processing tasks assigned by the Nokkhum controller. The tasks include video analytical solutions such as motion and face detection. The compute node worker includes resource sensors for monitoring the availability and capacity of computer resources and collects the information about the controller's monitoring. The compute node worker also includes resource sensors for monitoring usage requirements, such as the CPU and memory utilization, and storage space. This resource information is delivered to the Nokkhum controller.

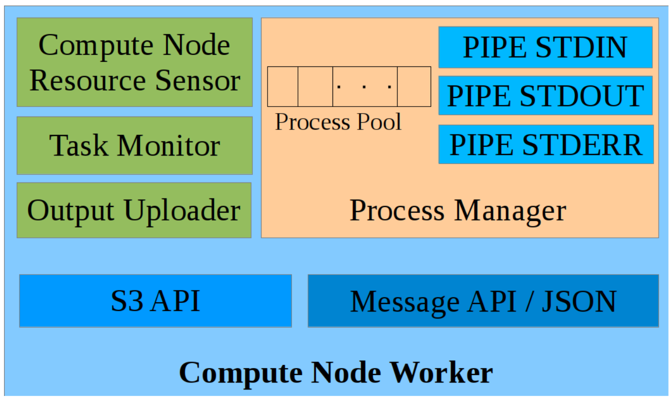


Figure 3.3: Modules of Nokkhum compute node worker.

One important module in the compute node worker is the video processing task manager which controls and monitors tasks and the task pool. This module manages the task life cycle involving the creation and termination of tasks. Also, the manager can watch task behaviors and handle shortcomings in order to promote system availability. The video processing task runs on a compute node worker containing video processing solutions for the video surveillance system. When the video processing task obtains logging messages, this module sends the messages to the controller via a resource status report. The output uploader module uploads images and video records to cloud storage via the Amazon S3 API. After the video processor output has been completely uploaded, the module can release it from the hard disk. In addition, if a compute node worker crashes, the tasks running on it will be terminated and restarted on another compute node worker without affecting other workers.

3.2.1.3 Nokkhum Video Processor

A Nokkhum video processor provides a VCA as a set of video analysis modules using the OpenCV library as shown in Figure 3.4. The camera configuration and video processor attributes are described using JavaScript Object Notation (JSON) [52] for building a video processing process. The camera configuration is a JSON object containing attributes such as the name, Uniform Resource Identifier (URI) [53], width, and height. The video processor attribute is a JSON object which contain a JSON array and identified by "processors" keywords. The video processor attribute allows the use of nested descriptions for complex configurations. The Nokkhum video processor parses JSON information from the compute node pipe that works as a standard input and constructs video processing threads. The video processor is a computing process controlled by the compute node worker.

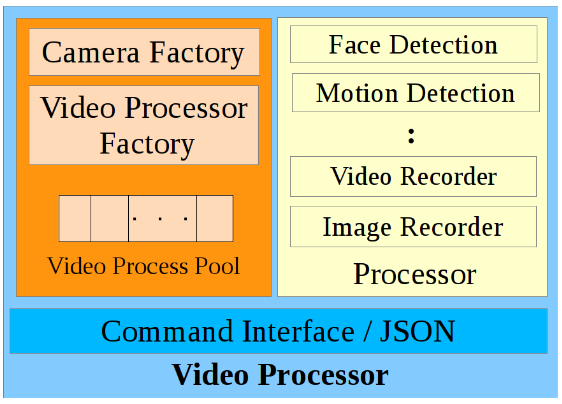


Figure 3.4: Modules of Nokkhum processor.

The Nokkhum video processor is flexible because users can design their VCA via JSON configurations. This thesis implements this component by employing threads and queues of image objects. Each video analysis is a running thread which connects to others via an image queue. Each video analysis thread generally includes two queues for image input and output. The video analysis thread receives an image object from the input queue and processes it. Afterwards, it puts the object into the output queue for another thread. A configuration example for the Nokkhum video processor is shown in Figure 3.5.

Figure 3.5 shows a configuration example involving four video processing modules: a motion detector, a face detector, a video recorder, and an image recorder. When a configuration is sent to a Nokkhum video processor, it translates that configuration and create related image queues.

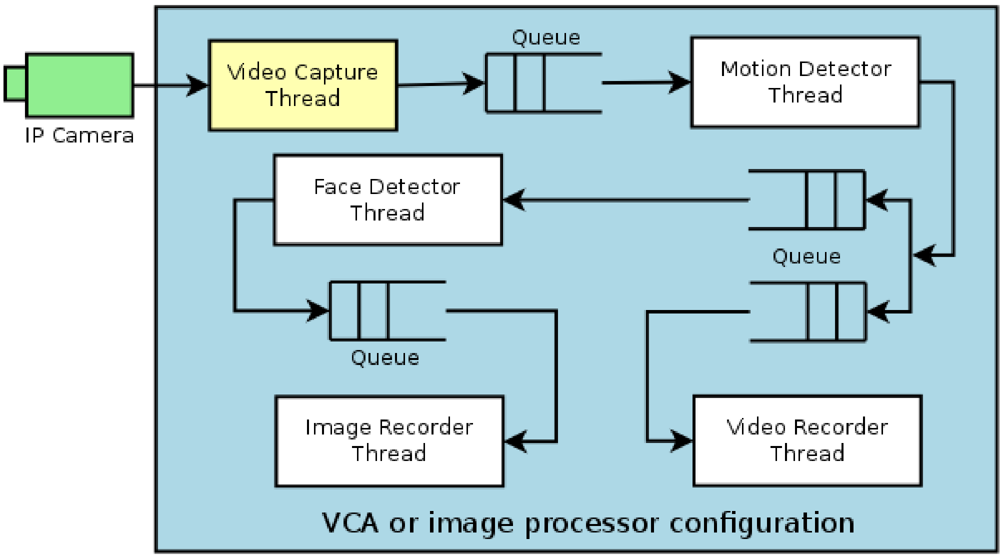


Figure 3.5: An example of Nokkhum video processor configuration.

3.2.1.4 Nokkhum API

A popular way to manage a distributed system is to use a single API server and multiple clients. One advantage is that the developer can carefully implement a central API server, so that the client implementation is lightweight, and it can support various operating system platforms. The Nokkhum API provides for camera management, video processing task control, video play-back, and media management for basic users as shown in Figure 3.6. For administrators, the API server provides system monitoring and high-level permission management of the functions provided to the users.

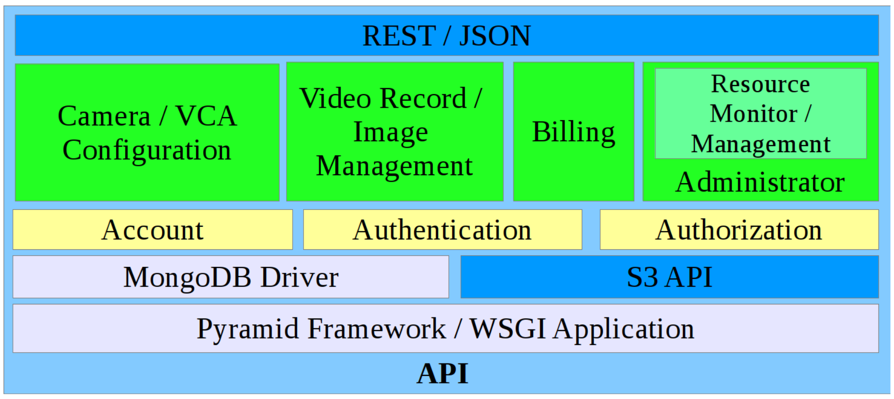

REST / JSON
Camera / VCA Configuration
Video Record / Image Management
Billing
Resource Monitor / Management
Administrator
Account
Authentication
Authorization
MongoDB Driver
S3 API
Pyramid Framework / WSGI Application
API

Figure 3.6: Modules of Nokkhum API.

The API has been developed using Python, Pyramid [54], and MongoDB [55], and using a REST style architecture via HTTP [49]. Clients are identified with token-based authentication when they connect to the server. A token is generated by the server during the authentication, and used in the reply sent to the client. If the API requires a secure connection for protecting the privacy of user requests, a simple solution is to switch to Hypertext Transfer Protocol Secure (HTTPS) [56].

3.2.1.5 Nokkhum Client and Web Interface

A Nokkhum client could be implemented in several ways, e.g. as an interface web, a mobile application, or a desktop application, because it connects to the Nokkhum API server using HTTP and a REST architecture. This thesis prefers a web interface client since it can be used via a web browser on both desktop and mobile devices. Moreover, a web-based interface can be displayed on many platforms, and is more easily developed and maintained than native programs. The web interface is implemented with Python and HTML, and utilized by both users and the administrator, providing camera and system information according to user role permissions. Users can create new camera configurations, compose image analyses, and watch videos or images obtained from the cloud storage as shown in Figure 3.7.

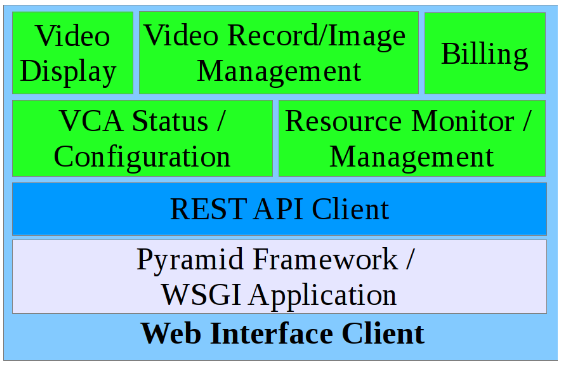

Video Display
Video Record/Image Management
Billing
VCA Status / Configuration
Resource Monitor / Management
REST API Client
Pyramid Framework / WSGI Application
Web Interface Client

Figure 3.7: Modules of Nokkhum client, web interface implementation.

Although the web interface can be used on many platforms, the users may have to adjust the device's screen resolution which is somewhat inconvenient. Consequently, it can be a burden for the developer who develops the native client to adjust it for best user experience and satisfaction. When applied in a real business model, the developer may apply a responsive style to solve the problem.

3.2.2 Nokkhum Architecture

Nokkhum offers a scalable VSaaS architecture by using message passing, implemented with the Advanced Message Queuing Protocol (AMQP) [48], for connecting the Nokkhum controller and the compute node workers. This design provides many options for system deployment and the exploitation of mixed computing resources (both virtual and physical machines) for improving the performance or flexible deployment according to the limitation of the targeted machine. The simplified version of Nokkhum's cooperating components is shown in the architecture overview in Figure 3.8. The inter-component communication and data format are described in the following sections.

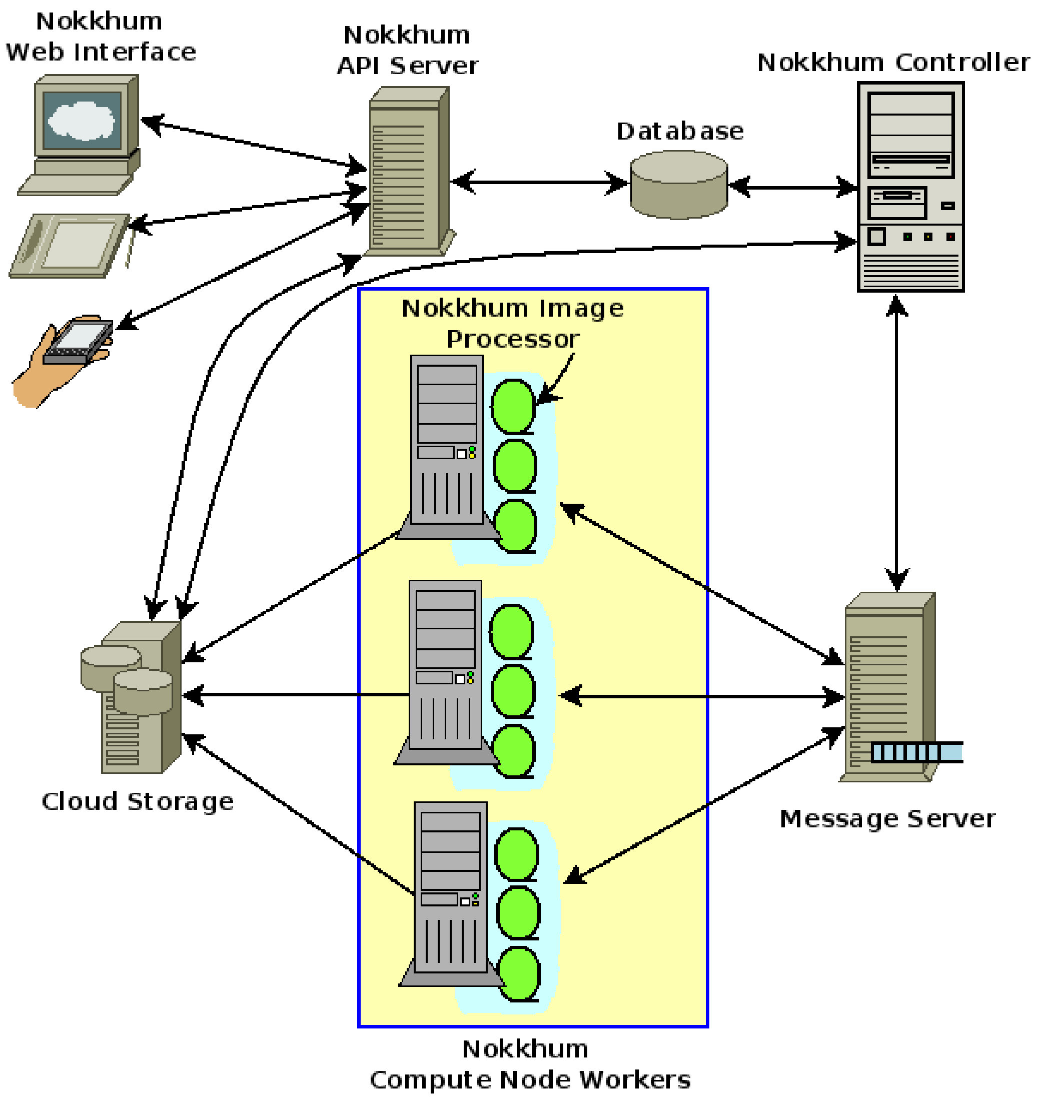


Figure 3.8: The overview of the Nokkhum architecture.

3.2.2.1 Nokkhum Components Cooperative Working

A sequence process of Nokkhum components is divided into two parts: first is interaction between the user and the front-end components as shown in Figures 3.9, and second is interaction between all back-end components as shown in Figure 3.10. In Figures 3.9, after a user has already registered his information in the VSaaS system, the user then initially creates a camera configuration and a video processing solution on a Nokkhum client, which is then passed to a Nokkhum API that stores the command in a database. Before storing that information, the Nokkhum API must validate, and verify the camera and video processing and compost the user request command for next Nokkhum controller processing. This process terminates after the Nokkhum API acknowledges the Nokkhum client. The user can watch, download, and delete the video records and images via the Nokkhum client.

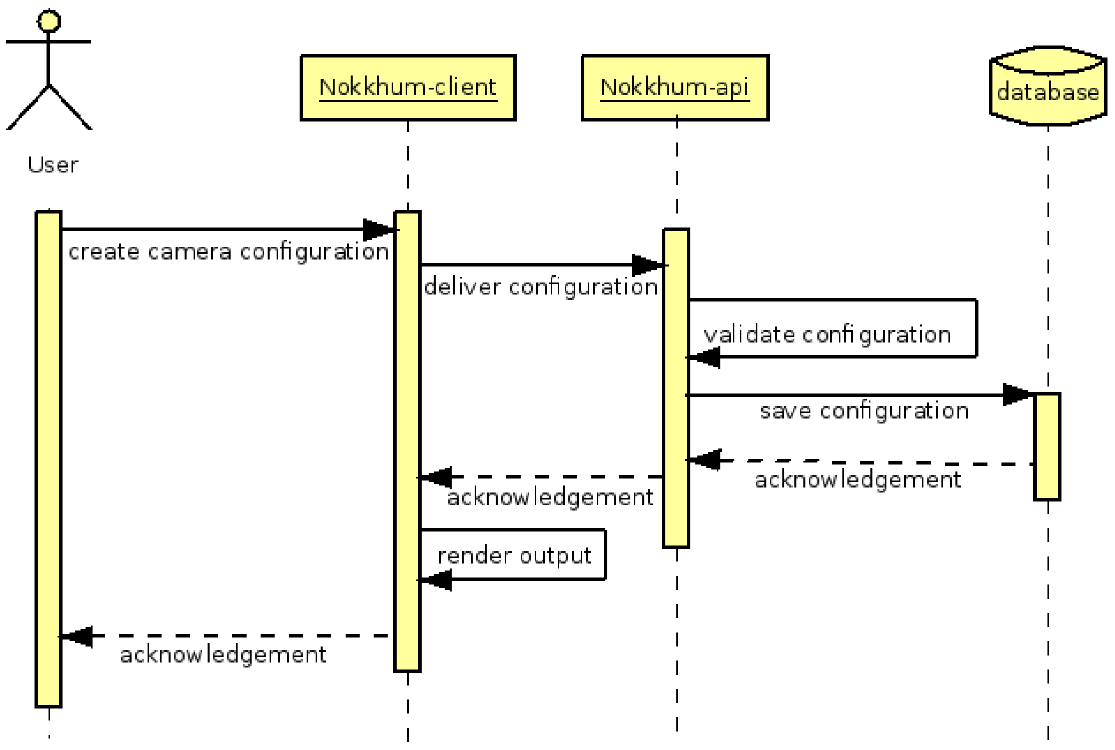


Figure 3.9: The VSaaS user configuration sequence.

Figures 3.10 presents a sequence diagram describing a back-end components work. When the Nokkhum compute node worker first appears in the Nokkhum system, it publishes a greeting message to the Nokkhum controller through the message broker. After the Nokhum controller gets the greeting message, the Nokkhum controller publishes an initialized central configuration for the new Nokkhum compute node worker. The Nokkhum compute node worker usually publishes the computing resource usage every certain time period. The

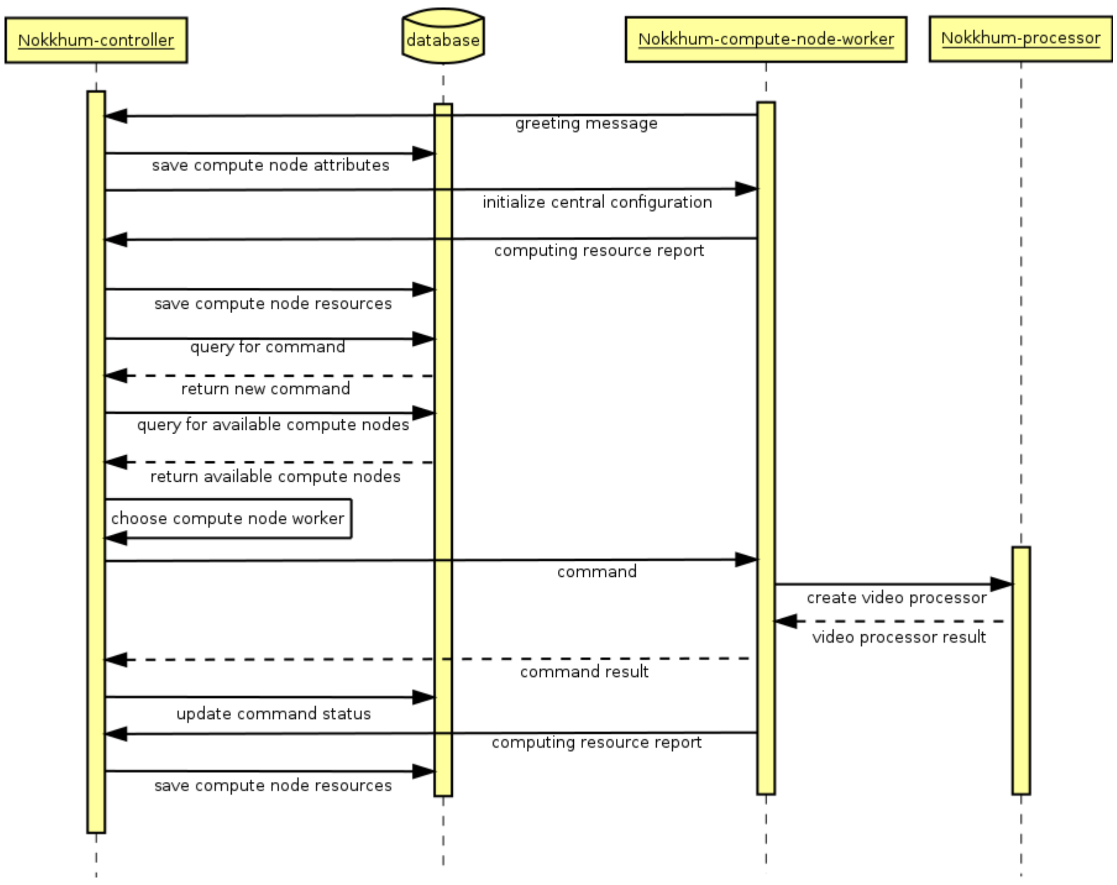

Nokkhum-controller
database
Nokkhum-compute-node-worker
Nokkhum-processor
greeting message
save compute node attributes
initialize central configuration
computing resource report
save compute node resources
query for command
return new command
query for available compute nodes
return available compute nodes
choose compute node worker
command
create video processor
video processor result
command result
update command status
computing resource report
save compute node resources

Figure 3.10: Interactions among system components of the VSaaS during the initialization.

controller keeps the information to be later used for determining a suitable Nokkhum compute node worker. The task scheduler on the Nokkhum controller becomes active when there is at least one video processing task on the database. After it gets the user's command, the compute node resource predictor module on the Nokkhum controller finds a suitable Nokkhum compute node worker using the persistent resource usage information of Nokkhum compute node workers in the database. The Nokkhum controller directs the user request command to the suitable Nokkhum compute node worker via the message broker. When the compute node worker receives the command message, it constructs a video surveillance solution and starts the related video processors. As the video processors produce output data (video records and/or images), the compute node worker stores the data in the cloud storage. All the video processor status such as CPU and memory usage can be reported to the Nokkhum controller for overviewing the remaining resource usage for next scheduling. The controller also monitors and manages the storage usage history of the video records and images.

3.2.2.2 Inter-component Communication

Nokkhum employs two main kinds of inter-component communication for exchanging information and controlling the system's behavior. The communication between the Nokkhum controller and Nokkhum compute node workers uses MOM for distributing commands and reporting resource status details to its physical and virtual machines. Program-to-program communication on the same computing machine is utilized between compute node workers and video processors. These two approaches are explained in more details below.

1) **Controller and Compute Node Workers Communication**

   As shown in Figure 3.8, the controller communicates with compute node workers using message passing. Two types of communication patterns are used: direct exchange and topic exchange. The controller and the workers employ direct exchange for greeting messages, and updating worker resources and video processing status or behavior. That allows the workers to send these kinds of the message without waiting for a response. The controller and the workers use topic exchange for synchronizing command messages, such as to start or stop video processor processes, or to request greeting information. Topic exchange communication is designed to wait for a response message and command confirmation.

   As shown in Figure 3.10, when a new compute node worker starts and finishes the booting state, it will send a greeting message to the message broker server. Then, the controller will receive a greeting message and add the worker to the resource pool. After receiving the greeting message, the controller sends configuration details to the worker, such as cloud storage configuration that includes the communication protocol for accessing the storage. Therefore, Nokkhum VSaaS requires central configuration from the controller, which distributes it to every compute node involved. After the compute node worker receives configuration information, it will update its system resource usage details, run the required video processors, and report the CPU utilization and memory usage to the controller.

2) **Compute Node Workers and video processors Communication**

   A compute node worker communicates with a video processor using the pipeline. A video processor is created by a worker when it receives a starting control message. The worker outputs all the commands for controlling the video processor's behavior via its standard input. When the video processor generates a message output, the compute node worker will read it from its standard output. The commands and results are written in a JSON format to facilitate processing. The worker checks the availability of all the video processors by monitoring their pipelines. Then, the compute node worker will report to the controller via a resource update system message.

3.2.2.3 Data Format

JSON is utilized as the default data format for cameras, video processors, and inter-component messages because it is both lightweight and available in several computer languages. Cameras and video processor attributes are described via JSON objects. A camera object includes the frame size, frame rate (frames per second) for video, camera manufactory information, and username and password to access the camera. A video processor object includes the name and attributes of the video processor. Inter-component message passing uses a JSON object for describing the command property. Listing 3.1 shows a JSON description for starting video surveillance commands composed by the Nokkhum controller.

3.2.2.4 Running Nokkhum on Infrastructure as a Service

The cloud infrastructure that provides physical or virtual machines, and other computing resources, is known as Infrastructure as a Service (IaaS). IaaS supplies computing resources on demand according to the user's requirements. IaaS plays an important role in starting up businesses by replacing high computer hardware purchasing costs with lower pay-as-use resource rental fees. Moreover, this scheme allows large businesses to reduce their hardware maintenance and administrative costs. As a result, many organizations are currently transforming their IT platforms into cloud computing services. However, there are few available details about the implementation of the SaaS.

IaaS providers typically implement their own APIs to access and manage computing resources by exploiting a compatible version of the AWS API. AWS provides many services, but the most popular are Amazon EC2 for computing and Amazon S3 for storage. There are many open source IaaS products which support the AWS API, including Eucalyptus [57], OpenNebula [58], CloudStack [59], and OpenStack [60]. Since IaaS can provide a private cloud infrastructure for an organization, Nokkhum is designed using the Amazon EC2 and Amazon S3 APIs which can run on most IaaS software providers.

```
{
  "action" : "start",
  "attributes" : {
    "cameras" : [
      {
        "id" : "52779ae724b5b108e243649e",
        "password" : "",
        "model" : "DCS-930L",
        "width" : 640,          // video width
        "height" : 480,         // video height
        "fps" : 10,             // frames per second
        "name" : "camera-01",
        "audio_uri" : "http://example.com/audio.cgi",
        "video_uri" : "http://example.com/video/mjpg.cgi?.mjpg",
        "image_uri" : "http://example.com/image/jpeg.cgi",
        "username" : ""
      }
    ],
    "processors" : [
      {
        "name" : "Motion Detector",
        "wait_motion_time" : 5, // wait for motion in 5 second
        "interval" : 3,         // process every 3 image
        "sensitive" : 95,       // motion sensitivity 0 - 100
        "processors" : [
          {
            "height" : 480,     // image height
            "fps" : 10,         // frames per second
            "width" : 640       // image width
            "name" : "Video Recorder",
            "record_motion" : true, // enable motion recording
          }
        ]
      }
    ]
  }
}
```

Listing 3.1: An example of JSON description for starting a video processor command.

3.2.3 Nokkhum System Scalability

The Nokkhum VSaaS system is divided into five components, with each providing specific functions. These components can be distributed across many computers which are connected to the message broker server. The MOM enables Nokkhum to scale easily to support dynamic user requirements. Moreover, Nokkhum can start running on just one machine, providing a small VSaaS system, and simply make a transition to support more cameras and video processing tasks, involving more computing machines. Nokkhum components could be fully deployed in many concurrent servers as shown in Figure 3.11

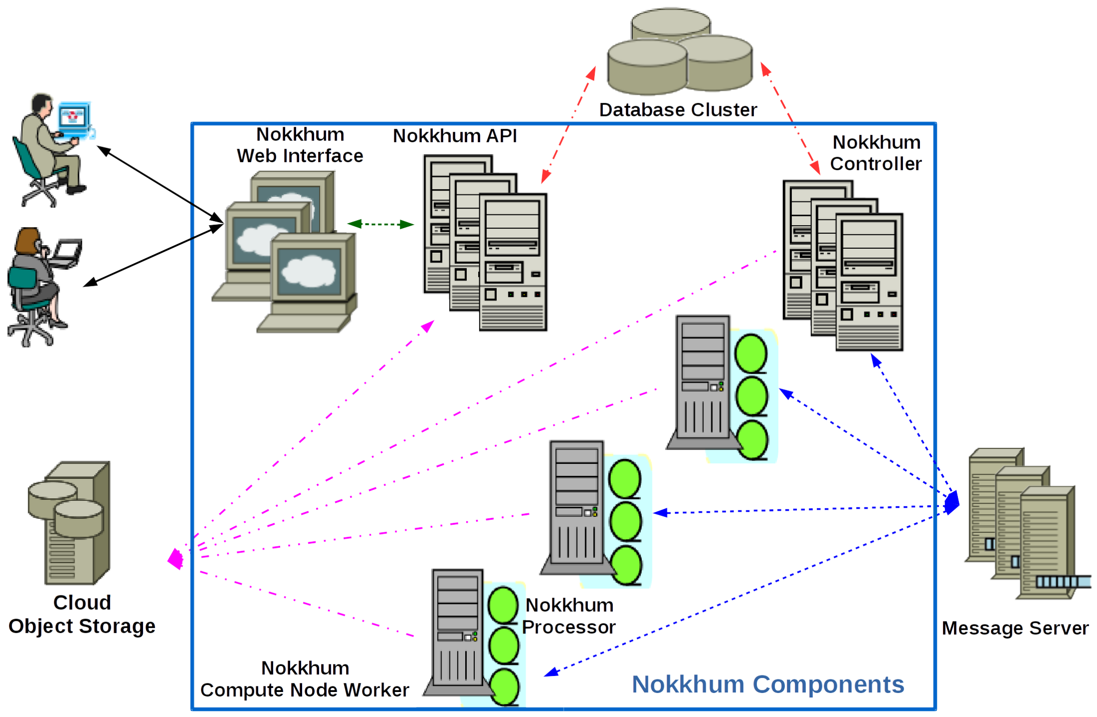


Figure 3.11: The overview of the Nokkhum architecture.

Figure 3.11 relates to Figure 3.8, and presents a full deployment of the Nokkhum system and necessary services that can prevent a situation of a single point of failure, and illustrates the scalability for supporting a large number of users. However, the full deployment is not possible in many situations, especially small and medium systems. This section describes possible topologies for deploying Nokkhum VSaaS in seven scenarios.

As seen in Figures 3.12 - 3.18, Nokkhum's components and other infrastructure software can be distributed across many types of computing machines in seven scenarios. Figure 3.12 shows the smallest system type, where all the components run on a single machine supporting a small VSaaS. In order to extend the system for processing more cameras, the computing unit (Nokkhum compute node worker and video processor) can restart congested

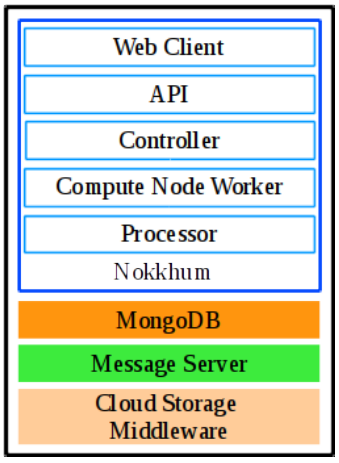


Figure 3.12: Nokkhum topology configuration pattern A - all services run on a single machine.

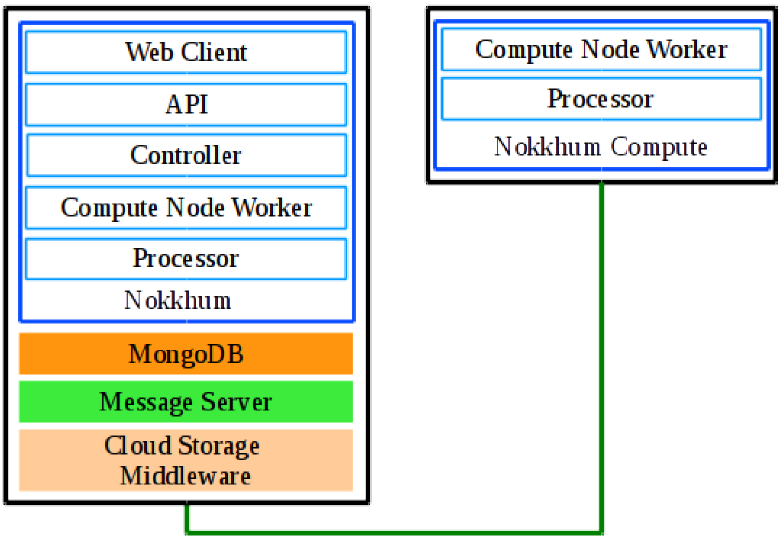


Figure 3.13: Nokkhum topology configuration pattern B - separating the compute node worker.

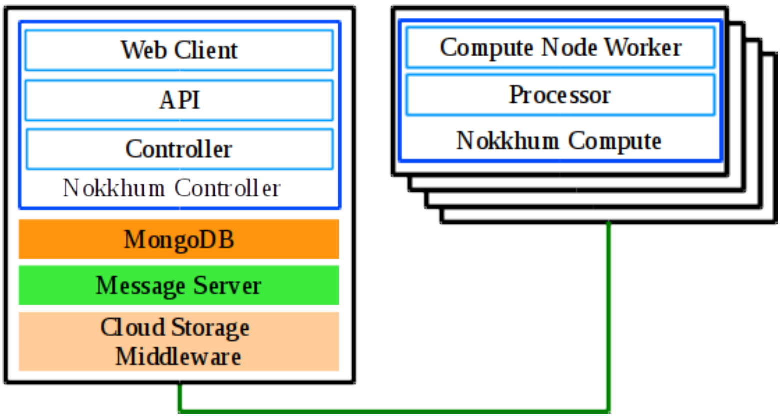


Figure 3.14: Nokkhum topology configuration pattern C - separating multiple compute node workers.

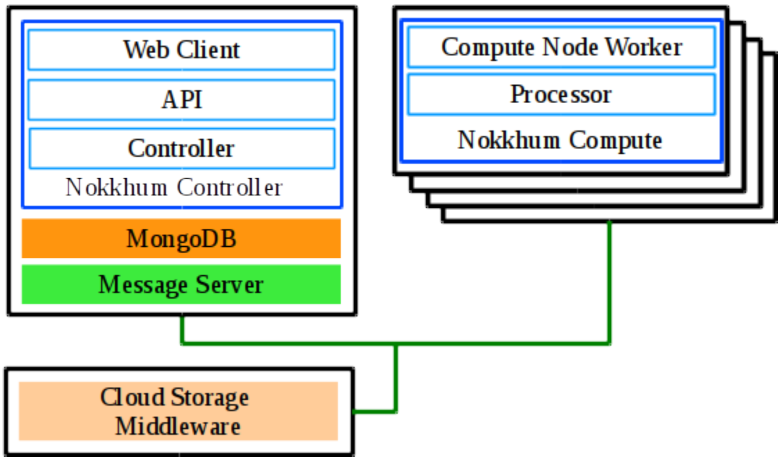


Figure 3.15: Nokkhum topology configuration pattern D - separating cloud objects storage for higher volumetric.

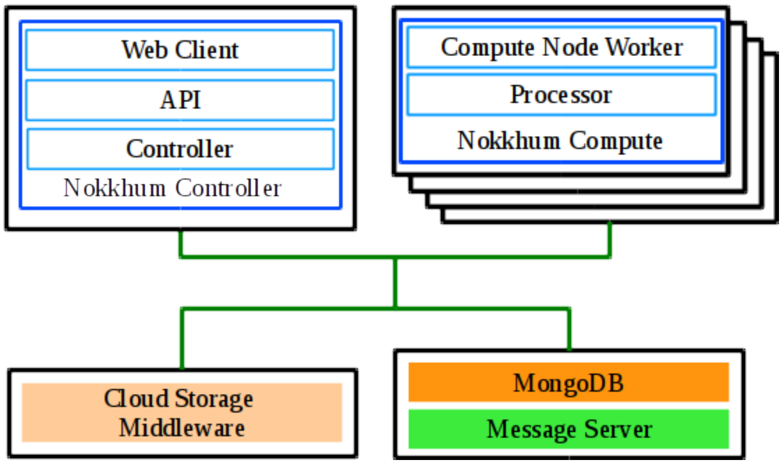


Figure 3.16: Nokkhum topology configuration pattern E - separating the database and message server.

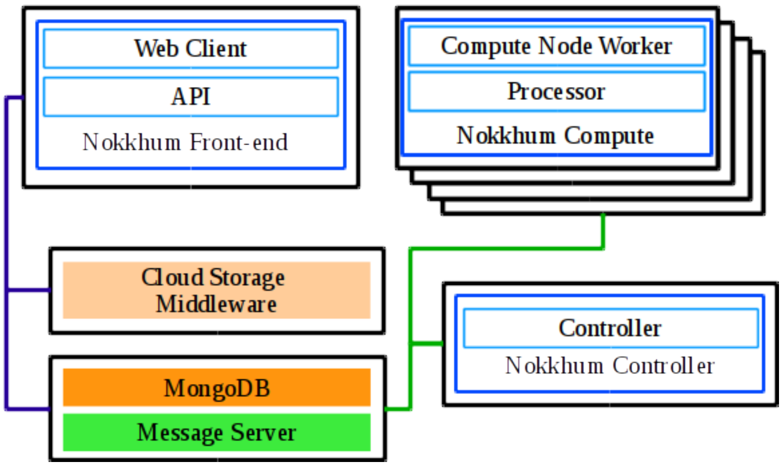


Figure 3.17: Nokkhum topology configuration pattern F - separating the Nokkhum controller and Nokkhum Front-end.

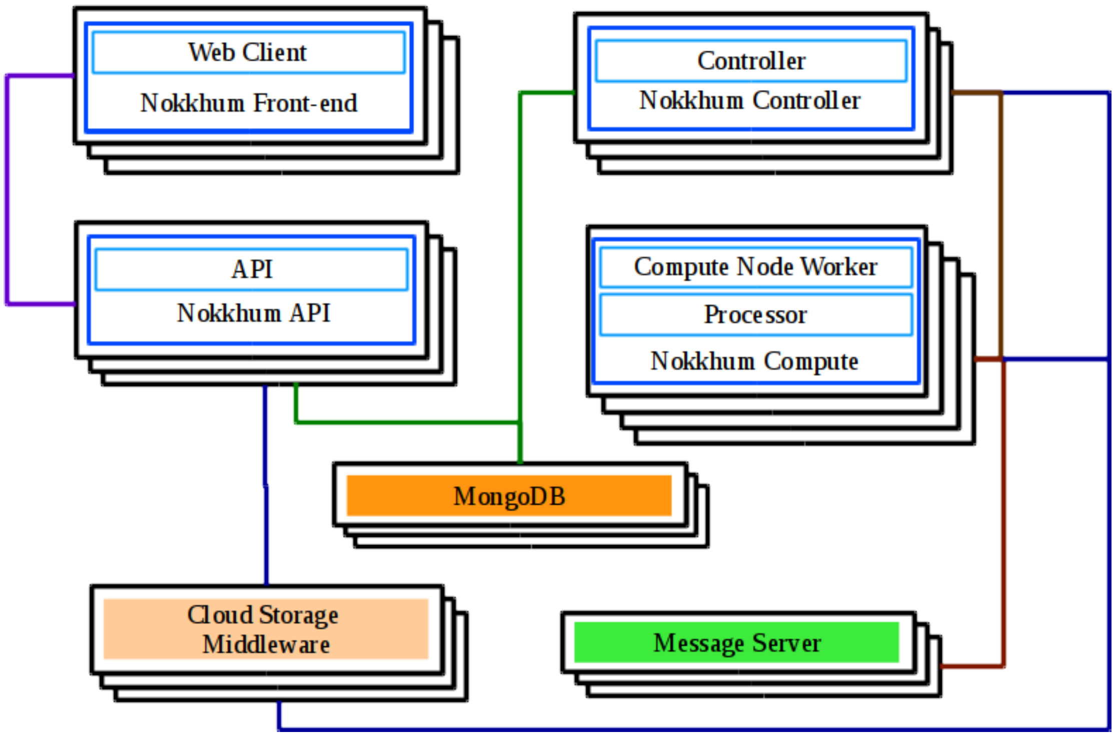


Figure 3.18: Nokkhum topology configuration pattern G - fully distributed components.

tasks on another computing machine added to the controller node as shown in Figure 3.13. In Figure 3.14, more computing units are added to deal with the increasing number of cameras, and to handle the expanded video processing requirements. In this configuration, the computing units and the controller unit are separated parts. Figure 3.15 is similar to Figure 3.14, but the cloud storage is moved to another machine for easier storage management and reducing the computation load of the controller node. This configuration is suitable for a medium-size VSaaS.

The system topologies in Figures 3.16, 3.17 and 3.18 are preferable when a highly scalable system is needed because the Nokkhum components and infrastructure software modules are distributed across many computing machines. In Figure 3.16, MongoDB and the message server are moved to run on different machines in order to increase the system availability. The message server and MongoDB are shared services for supporting computing units and controller nodes. If the Nokkhum components and the message server are not connected, it is likely that the system will fail. Therefore, the extracted parts, MongoDB, and the message server, are designed to work in close cooperation with the controller node in all proposed topologies in order to increase the system availability. In Figure 3.17, the Nokkhum controller is separated from the front-end machine. The rationale is that the controller component can run on its own without affecting other components, and if the controller component runs on a private network, it also increases the security. The architecture is also designed to support a complete distributed system of which components are located on many computing machines as shown in Figure 3.18, thereby providing a large scale VSaaS system. This scenario is appropriate for a public VSaaS provider and offers a high level of system availability. On the other hand, it involves many computing machines and requires many system resources.

The Nokkhum architecture addresses the system scalability across various scenarios. The VSaaS providers can freely choose configuration patterns for supporting their requirement. Moreover, the Nokkhum component configuration can combine both physical and virtual machine servers according to the organization's economy and information technology proficiency. The organization can choose their security policy whether running many components in a private network or providing context components in a public network.

3.3 Analysis of Video Processing Workload Characteristics

Few VSSes perform video workloads analysis or emphasized on scalable distributed video processing over a pool of computing resources. Many VSSes only utilized a single video size and varied frame rate in their case studies. Recently, VSSes had been transformed into VSaaS for supporting various user requirements, and the workload analysis was likely to provide wrong results for these systems. They did not take into account significant parameters involving changing video frame size and rate, or video processing based on several machine specifications. This section describes video workload characteristics affecting computing resources based on the Nokkhum VSaaS architecture.

3.3.1 Video Processing Task Exploration

Most VSSes employ similar compute nodes specification, because it is easy to manage many video processing tasks with the same parameters. This simple scenario is insufficient for a VSaaS system with dynamic requirements involving many variables, such as frame rate, frame size, and video processor type. In addition, the cloud environment provides VM specifications according to its capacity and limitations. This means that the video processing task will consume different computational resources on different VM specifications, in a difficult-to-predict manner. Therefore, this thesis utilize video processing task exploration to determine the computational resources consumed by the desired video processing task, depending on its parameters. The results are used in workload analysis described in Section 3.3, which become a heuristic for video task scheduling. There are two different ways to run video processing exploration: the first is manually run by the administrator, and the second is automatically done by the Nokkhum controller, which is described as follows:

1) Video processing task exploration during system installation is manually run by the administrator, because he usually knows the number of compute nodes and their specifications. With manual execution, the administrator can collect all the resource usage information about the video processing tasks soon after the initialization. This means that the Nokkhum system does not need to collect any further resource consumption information for the task scheduling. Nokkhum can use the existing information for scheduling without repeating the task exploration.

2) The Nokkhum controller automatically runs video processing task exploration under two circumstances: the first is when there are no experimental results in the database related to the compute node. The second occurs when the node is idle for at least 100 % when the total percentage (more than 100 %) is calculated by 100 % multiplied by the number of CPU cores. In this case, the administrator does not have to decide to execute video processing task exploration. Also, the scheduler has to focus on workload estimation in order to respond to the tasks in the queue. If it has to wait for experimental results from video processing task exploration, tasks will probably have a long wait in the queue.

The exploration employs two types of video processing tasks: motion detection and video recording. A surveillance video from the VIRAT video dataset was used in our experiments, with eight frame rates (1, 5, 7, 10, 15, 20, 25 and 30 FPS) and six frame sizes (160x120, 320x240, 640x480, 800x600, 960x720, and 1120x840 pixels). A total of 96 ($2 \times 8 \times 6$) test cases were executed, with each test running for two minutes, for a total of 192 minutes, which is too long. This means that scheduling required another approach to deal with missing experimental results from the task exploration caused by the skipping of some tests to make the scheduling faster. The resource decision approach will be presented in Subsection 3.4.2.

3.3.2 VCA and Video Recorder Workloads Analysis

VCA and video recorder workloads play an important role in driving VSS. Normally, the consumption of computing resources by the VCA and video recording tasks is a function of the frame rate and frame size. The CPU and memory utilizations of the VCA and video recording tasks vary when running on different machines. There are many multimedia systems, both VSSes and VSaaSes, which focus only on static workloads and homogeneous computing machines. However, the situation is rather different in real deployment environments, especially in Cloud infrastructures.

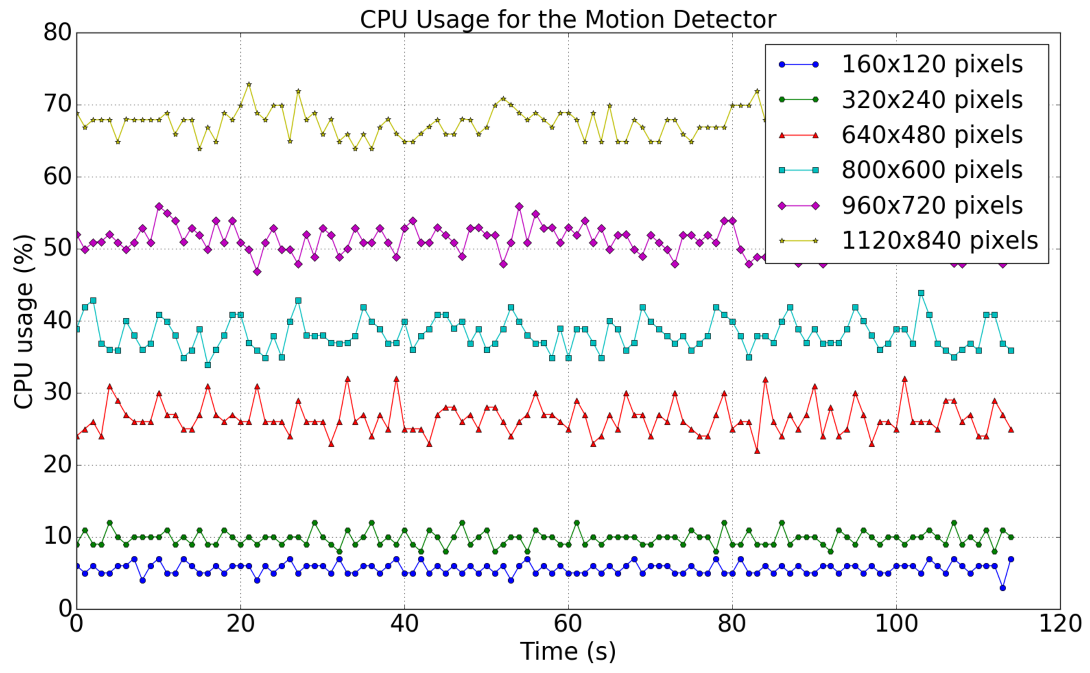
In general, the IaaS provider promises to provide all customers with the same VM template for the same charge. However, when the IaaS provider adds new hardware of a different specification, it is possible that it will provide a different certified VM template. Then, when the VSaaS provider deploys a VSaaS system on a Cloud IaaS, the efficiency of the VCA and video recording tasks scheduler will be affected. Therefore, this thesis have studied the behaviors of the VCA and video recording tasks in terms of computing resource consumption to improve the scheduling performance.

3.3.2.1 VCA and Video Recorder

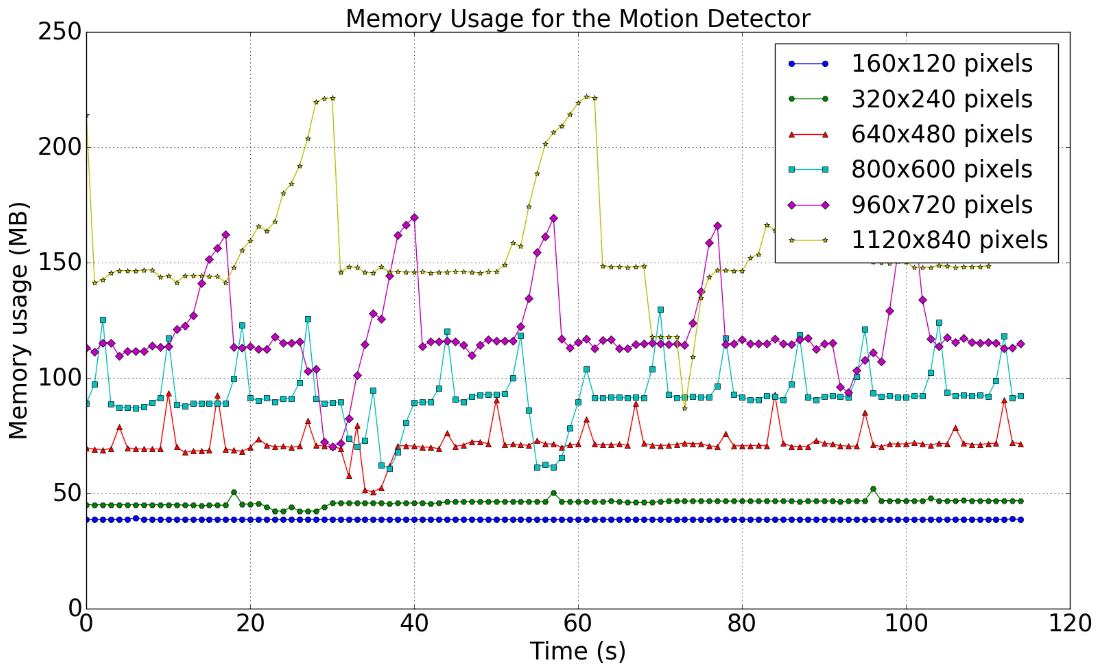
The two main factors affecting resource consumption are the video frame rate and frame size. It is difficult to study VCAs used in VSaaS, due to the many available types. The most popular is motion detection for filtering motion sequences of, which the footages are subsequently passed to the video recorder or used to notify the user. Focusing on motion detection and video recording tasks, and results of resource consumption handled by initially fixing the frame rate and varying the frame size are shown in Figures 3.19 and 3.20. The CPU and memory consumption for different frame rates are shown in Figures 3.21 and 3.22. All the experiments were performed on a physical machine, an AMD FX(tm)-8320 eight-core processor with a 3.5 GHz CPU and 16 GB RAM.

The video's frame rate is fixed at 10 frames per second (FPS) in Figures 3.19 and 3.20, and the results show how increases in the frame size increase the consumed computational resources, namely CPU and memory usage, for both motion detection and video recording. Also, when the frame size was fixed at 640x480 pixels and the increasing frame rate of the motion detector and video recorder increased more consumed computational resources as shown in Figures 3.21 and 3.22. In further experiments, when increasing the frame rate, it caused the motion detector task to consume memory up to the maximum buffer allowed, which is expected for the Nokkhum processor.

Figures 3.19, 3.20, 3.21 and 3.22 show individual resource consumption. However, the VSaaS video processing task can combine multiple VCAs for complex analysis. An example of VCA combination is the cooperation between the motion detector and video recorder for recording when motion sequences occur. Figure 3.23 shows the resource consumption of the motion recorder which employs a continuously running subtask to filter motion events before passing them to the video recorder. Therefore, the bottom line of each CPU usage

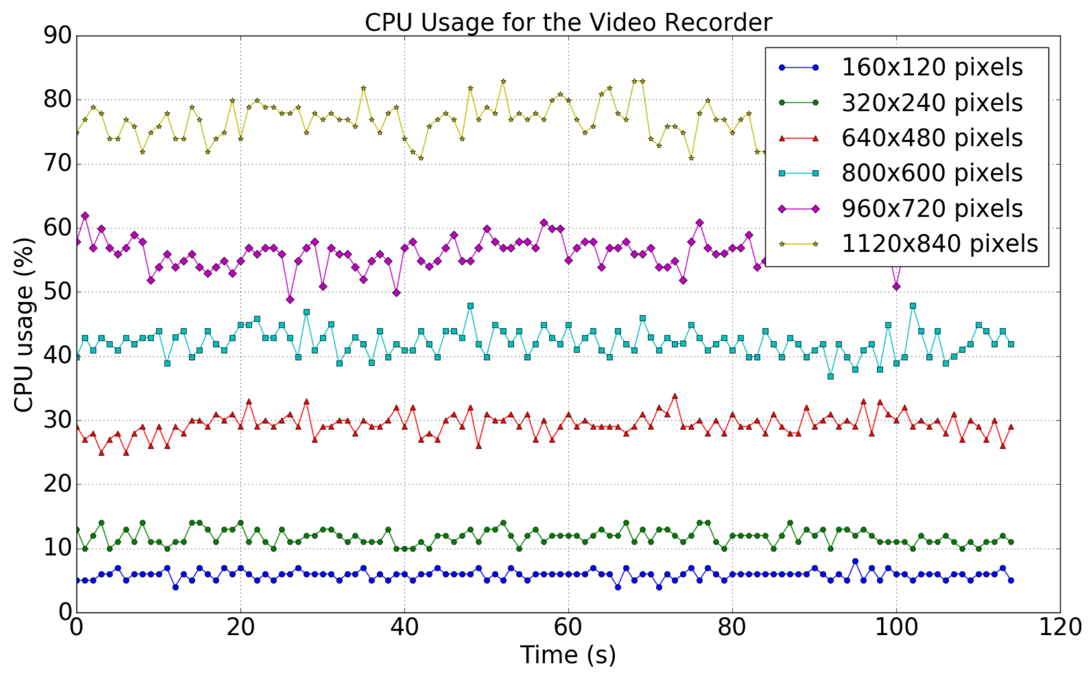


(a) CPU usage for the motion detection

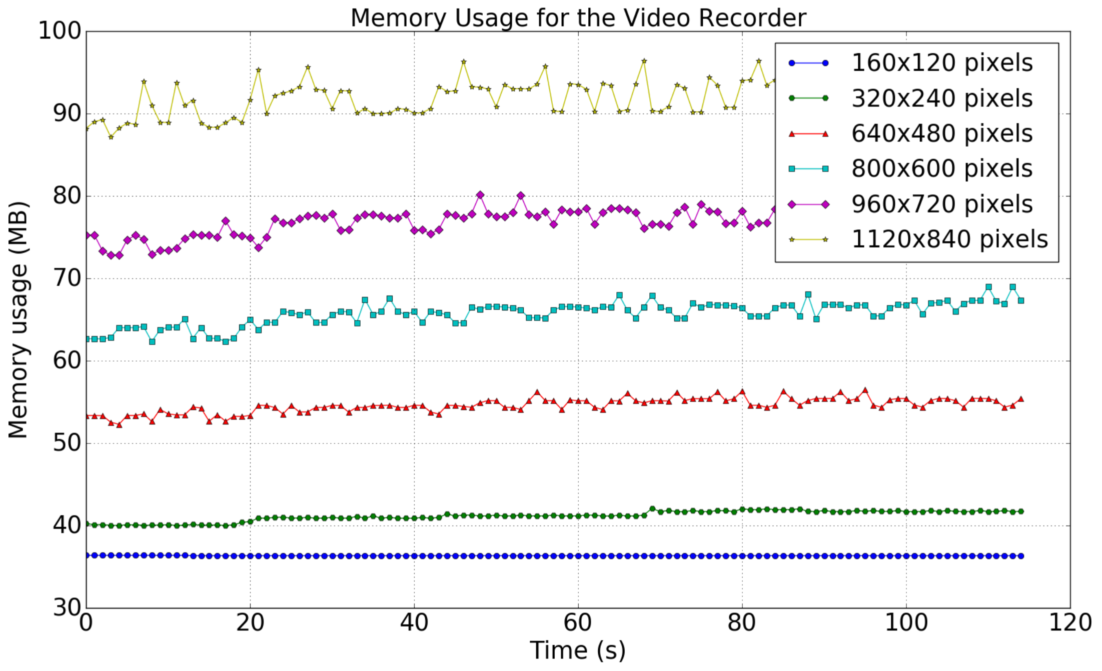


(b) Memory usage for the motion detection

Figure 3.19: Motion detector resource consumption at various frame sizes with the fixed frame rate of 10 FPS.

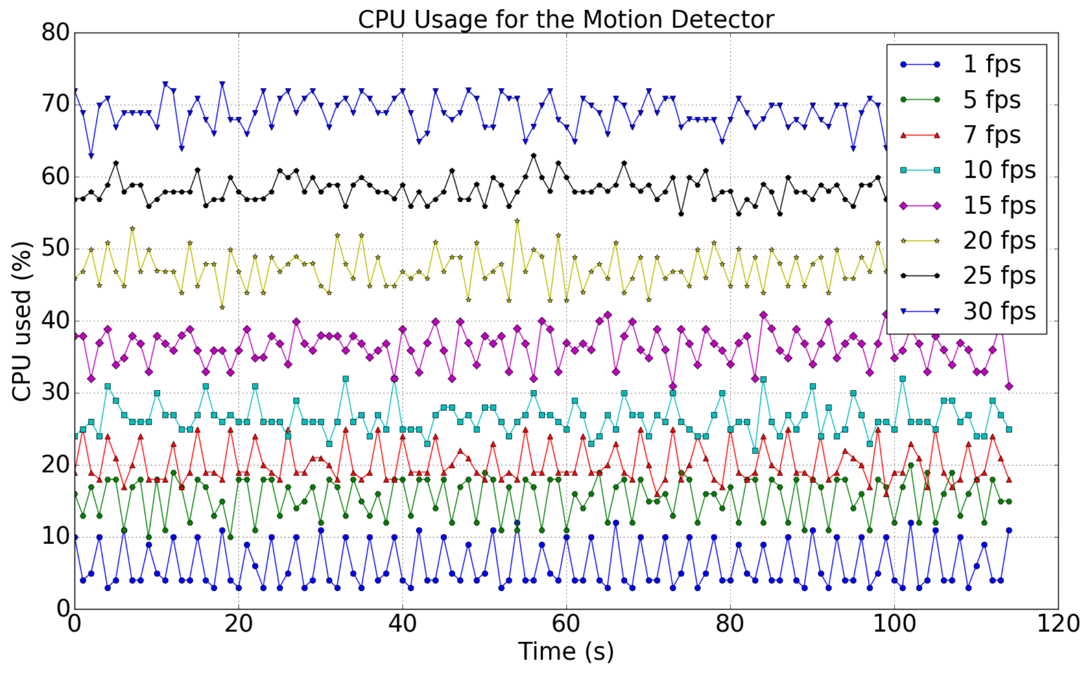


(a) CPU usage for the video recorder

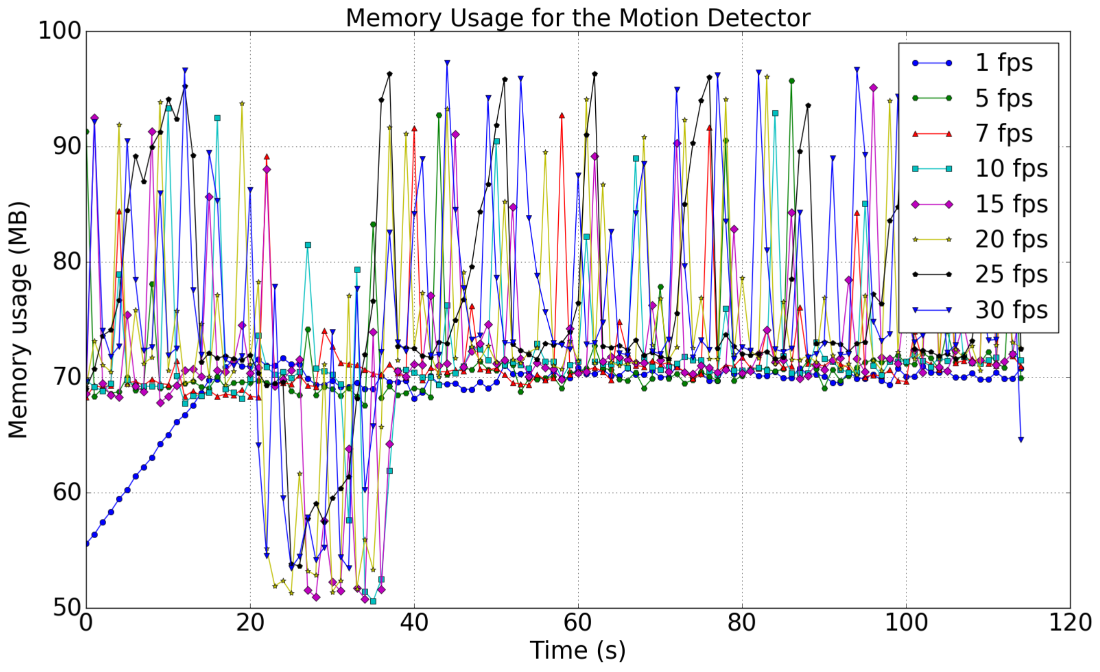


(b) Memory usage for the video recorder

Figure 3.20: Video recorder resource consumption at various frame sizes with the fixed frame rate of 10 FPS.

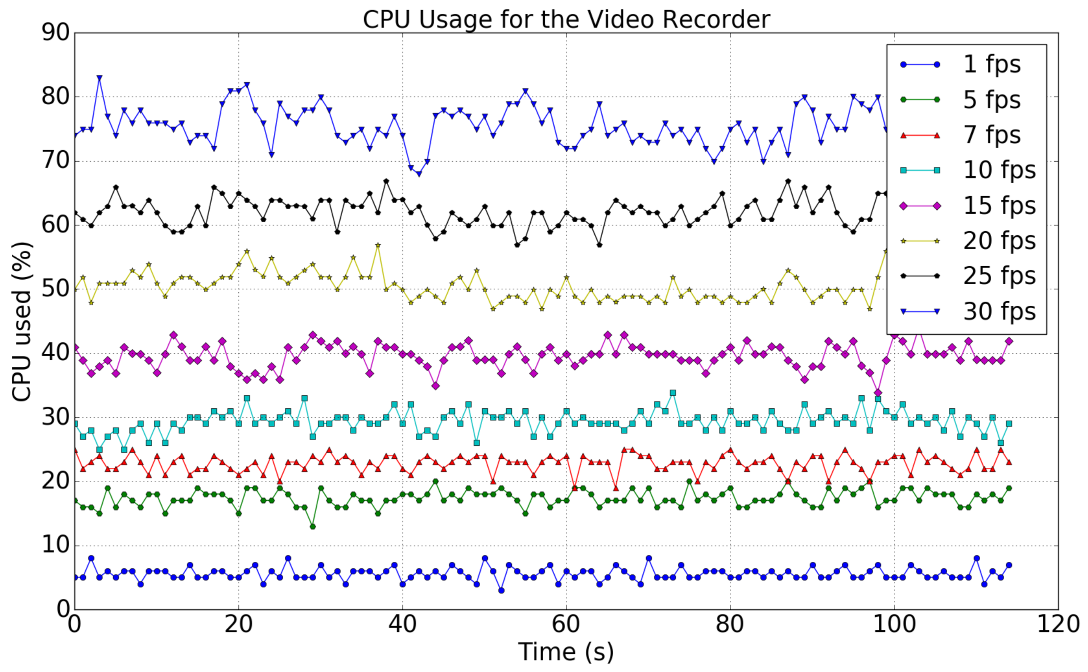


(a) CPU usage for the motion detection task

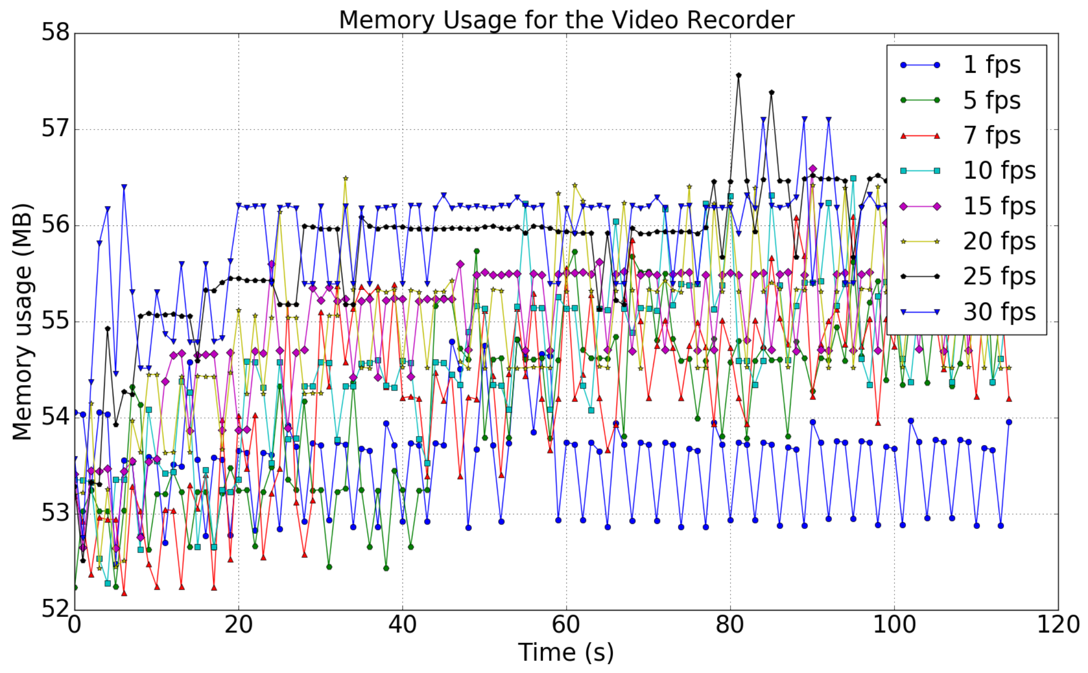


(b) Memory usage for the motion detection task

Figure 3.21: Motion detector resource consumption at various frame rates with the fixed frame size of 640x480 pixels.

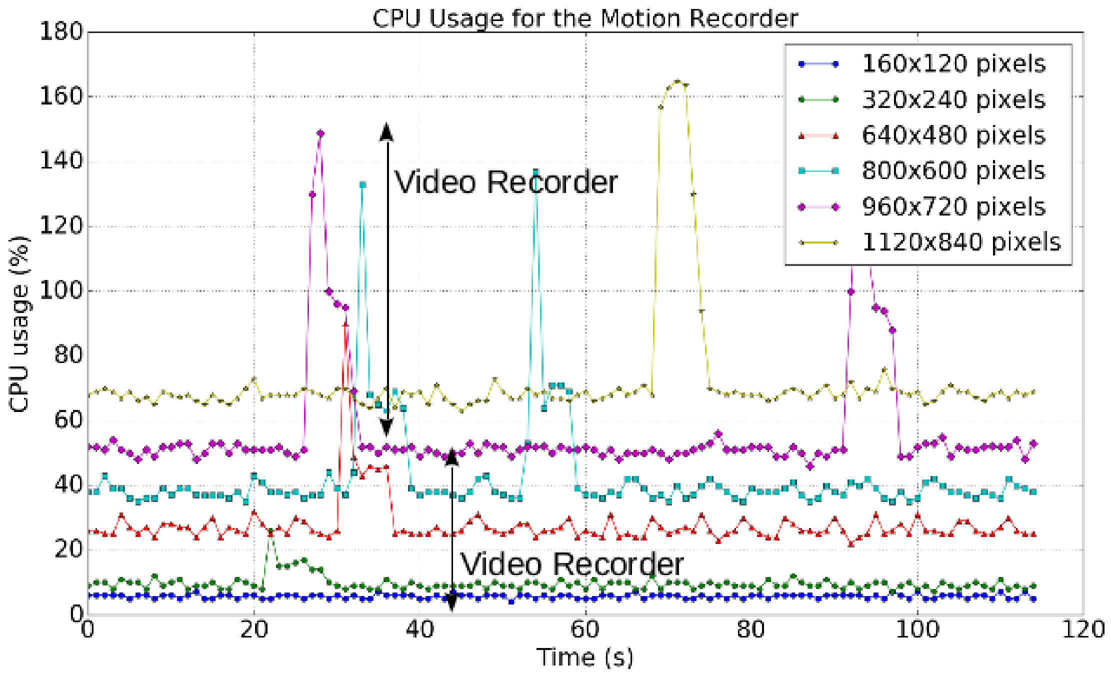


(a) CPU usage for the video recorder task

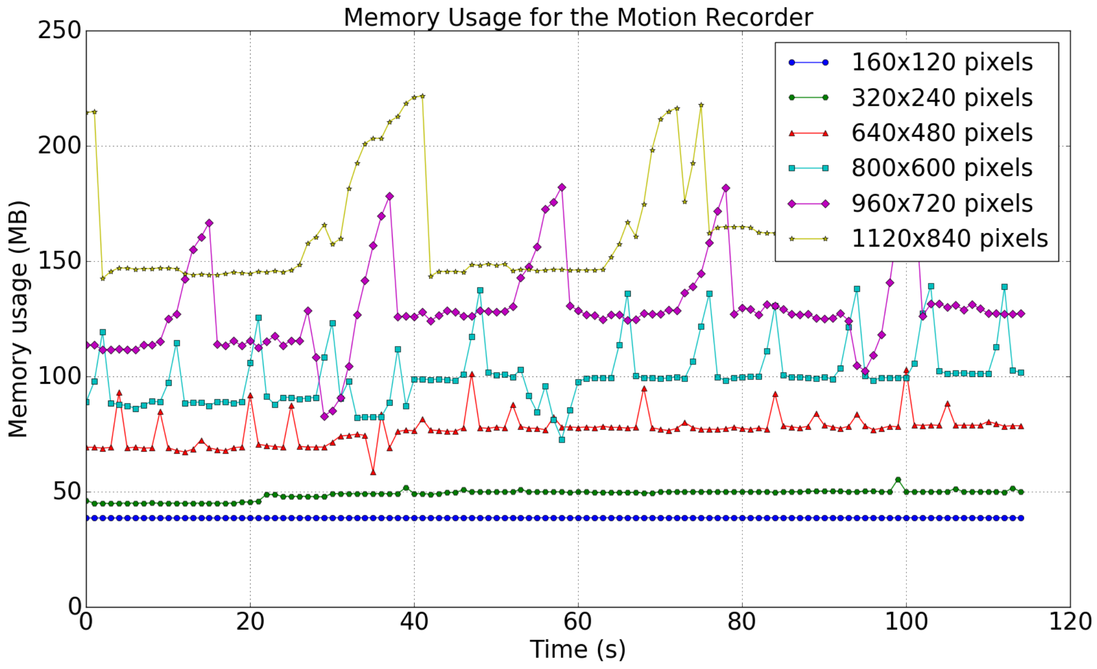


(b) Memory usage for the video recorder task

Figure 3.22: Video recorder resource consumption at various frame rates with the fix frame size of 640x480 pixels.



(a) CPU usage for the motion recorder task



(b) Memory usage for the motion recorder

Figure 3.23: Motion recorder resource consumption at various frame sizes with the fixed frame rate of 10 FPS.

graph in Figure 3.24(a) is the resource consumption of the motion detector. The overshooting line represents the CPU resource consumption of the video recorder. Also, memory usage has similar characteristics to the motion detector on the bottom line as shown in Figure 3.24(b).

3.3.2.2 VCA and Video Recorder in Various Computing Specifications

This section presents resource consumption for the motion detector, video recorder, and motion recorder running on different physical machines shown in Figures 3.24, 3.25, and 3.26. All experiments utilized a fixed frame rate at 10 FPS and a frame size of 640x480 pixels. The testing machines used a x64 CPU with more than 4 GB of RAM.

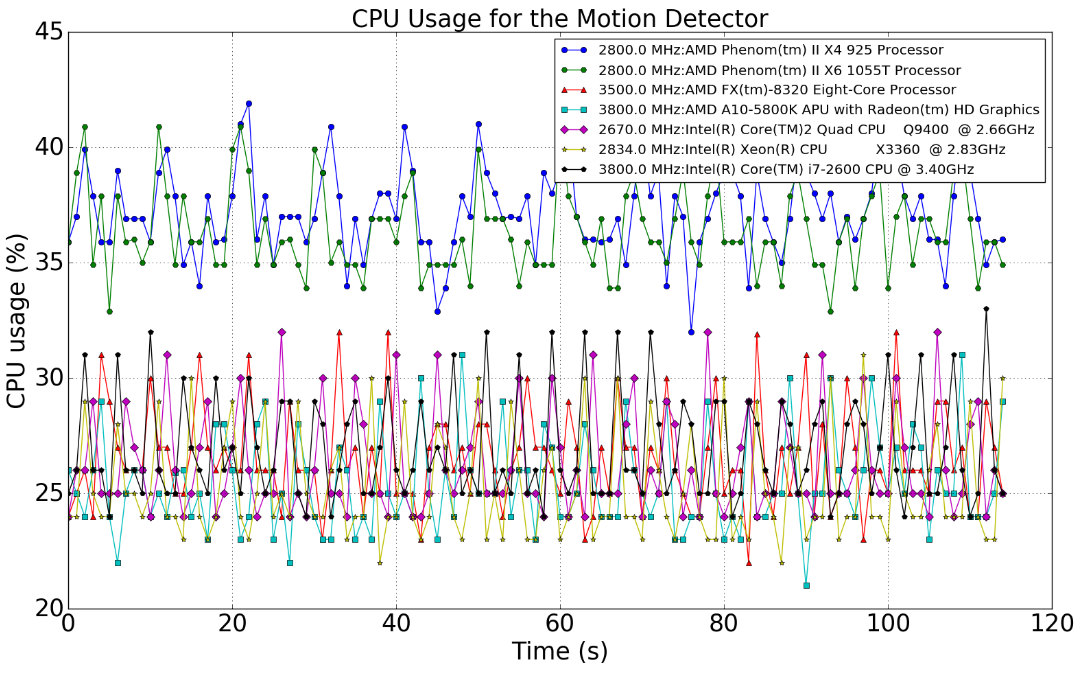
In Figure 3.24(a), it is difficult to find any outstanding related factors to differentiate the CPU usage characteristics in the motion detector when using different and complicated CPU architectural designs. When dividing the experimental results according to vendor (AMD and Intel), it seems that the CPU frequency is the only discriminator. The memory consumption of the machines in Figure 3.24(b) imply similar utilizations.

Video recorder resource consumption is shown in Figures 3.25(a) and 3.25(b). The video recorder resource consumption is different from the motion detector due to different CPU/memory architectural designs. For example, some desktop CPUs employ a special video codec chip set, to lower CPU utilization, but return the same frame size and rate.

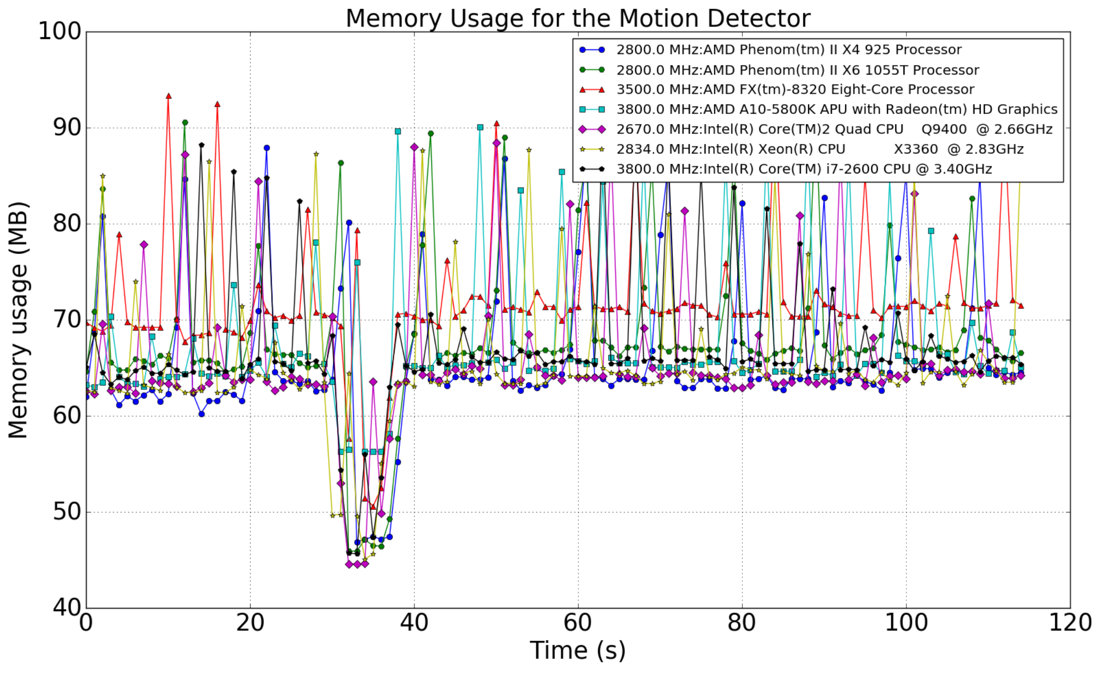
The motion recorder is a combination of the motion detector and video recorder, and so inherits CPU and memory utilization characteristics from both. Its results are shown in Figures 3.26(a) and 3.26(b). The bottom line in Figure 3.26(a) comes from the motion detector, and the overshooting line comes from the video recorder. However, it is difficult to identify which motion recorder characteristics are affected by the video recorder's CPU consumption, which depends on the CPU model and vendor.

3.3.2.3 Virtual and Physical Machines

The Nokkhum VSaaS supports hybrid virtual and physical machines for the scalability of the cloud environment. Comparing results between VMs and physical machines helps the Nokkhum system to improve the task scheduling performance for both machine types. The results involving motion detection, video recording, and motion recording are shown in Figures 3.27, 3.28, and 3.29. All the experiments used the frame rate of 10 FPS and frame size of 640x480 pixels. The physical machines support virtualization technology with the KVM [42].

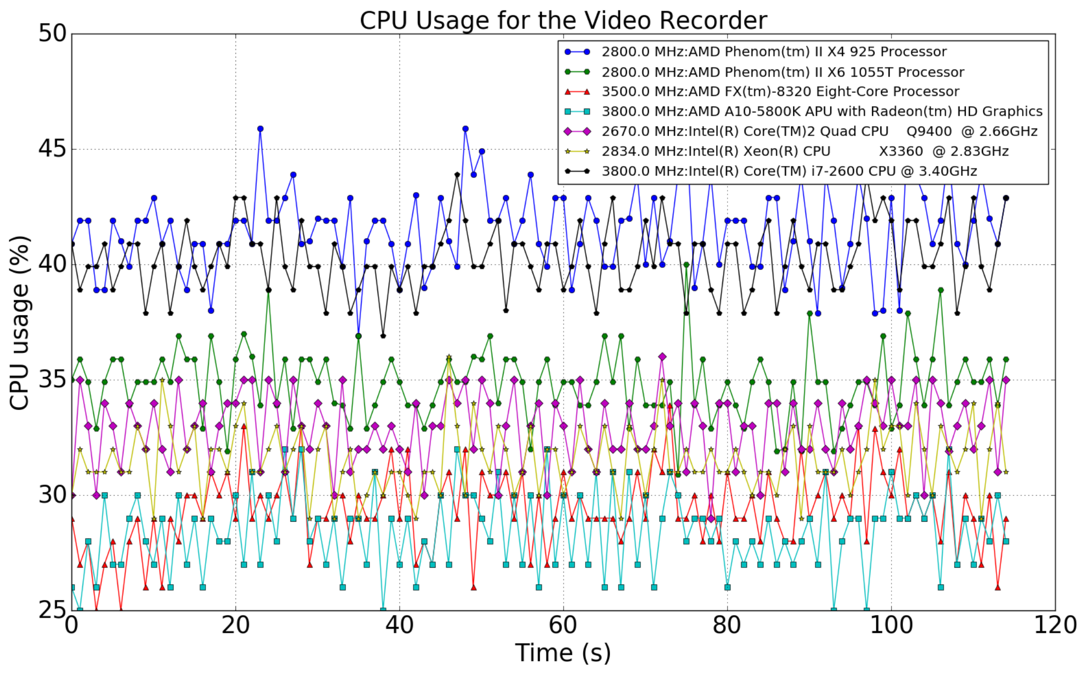


(a) CPU usage for the motion detector

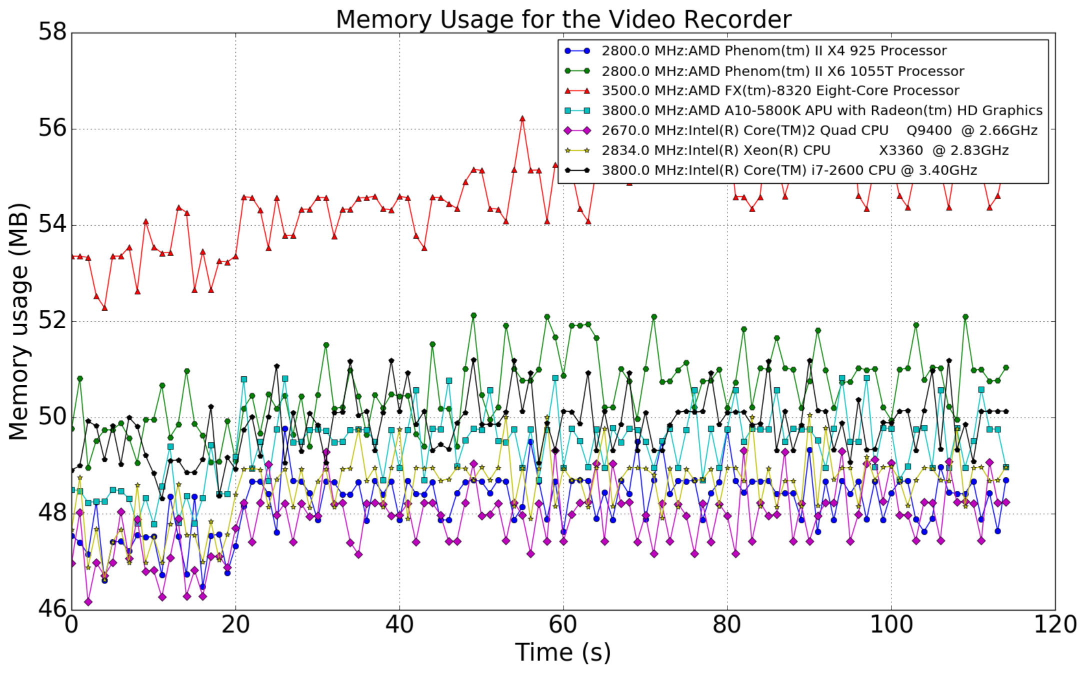


(b) Memory usage for the motion detector

Figure 3.24: Resources consumption for the motion detector running on different physical machines with the fixed frame rate at 10 FPS and frame size at 640x480 pixels.

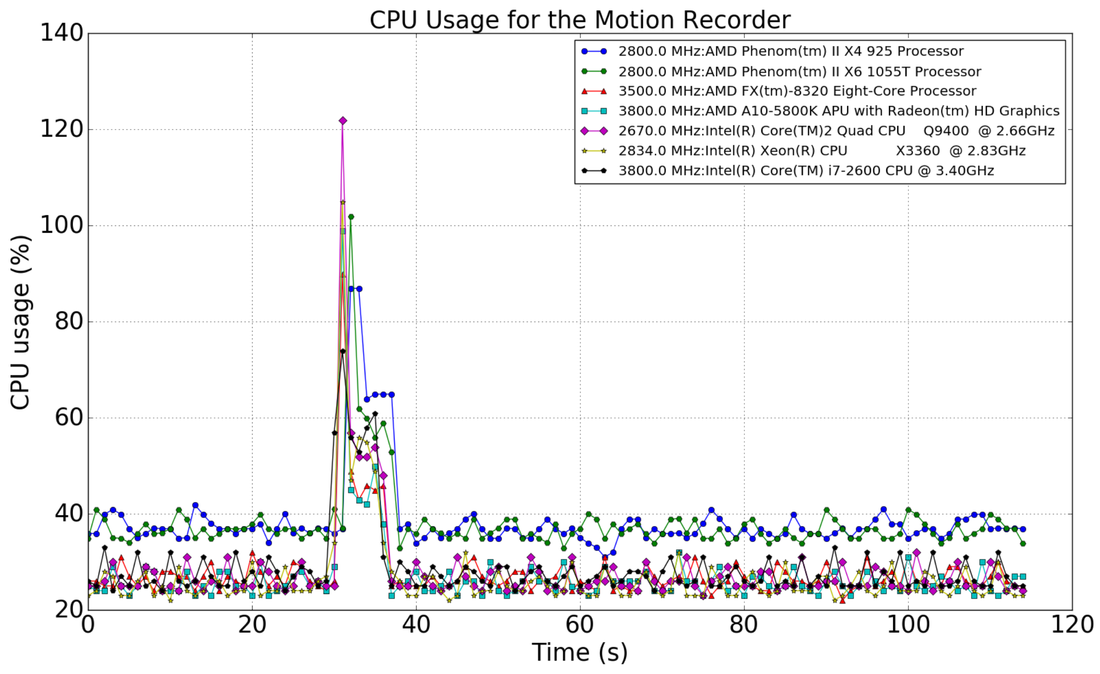


(a) CPU usage for the video recorder

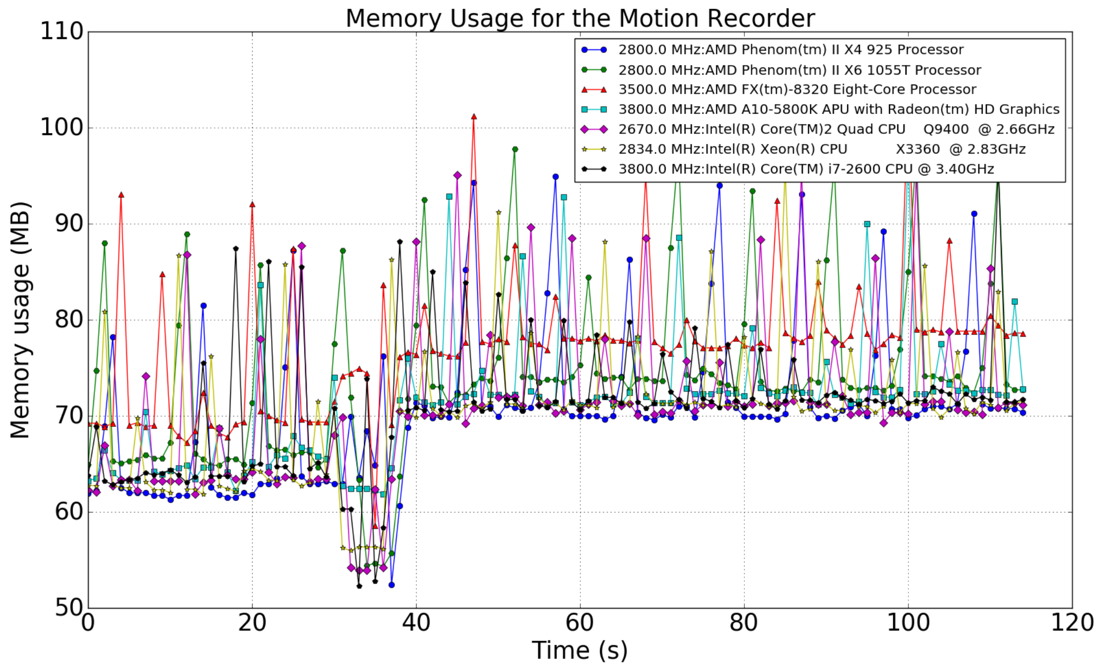


(b) Memory usage for the video recorder

Figure 3.25: Resources consumption for the video recorder running on different physical machines with the fixed frame rate at 10 FPS and frame size at 640x480 pixels.



(a) CPU usage for the motion recorder



(b) Memory usage for the motion recorder

Figure 3.26: Resources consumption for the motion recorder running on different physical machines with the fixed frame rate at 10 FPS and frame size at 640x480 pixels.

Figures 3.27(a), 3.28(a), and 3.29(a) present the CPU utilization comparison between the VMs and physical machines. The patterns of the results of the VMs and physical machines look similar. However, the VMs' CPU utilization is a little bit higher. The memory usage in Figures 3.27(b), 3.28(b), and 3.29(b) show a different characteristic comparing to that of the CPU utilization. All memory consumption of the VM task is a little bit lower than that of the physical machine.

Figures 3.27(a), 3.28(a), and 3.29(a) present the CPU utilization for VMs and physical machines, showing that the pattern of results are similar for both, but that the VMs' CPU utilization is a little bit higher. Figures 3.27(b), 3.28(b), and 3.29(b) show that memory consumption for VM tasks is a little bit lower than that for physical machines.

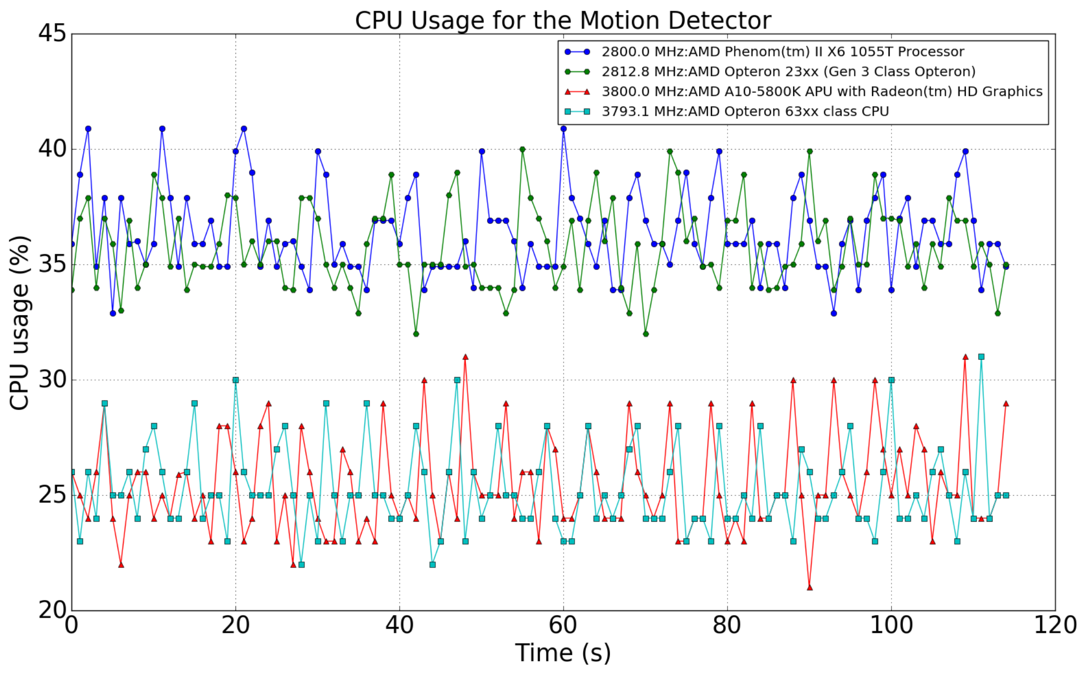
3.3.2.4 Summary

Figures 3.19 - 3.29 show results from experiments using the frame rates 1, 5, 7, 10, 15, 20, 25, and 30 FPS, and frame sizes 160x120, 320x240, 640x480, 800x600, 960x720, and 1120x840 pixels. The resource consumption results become unclear when they combine frame rate and frame size, and it is difficult to discriminate which are from the VCA or the recording tasks. Moreover, the computing resource utilizations of the VCA and video recorder tasks are different when different machine specifications and hypervisors are employed.

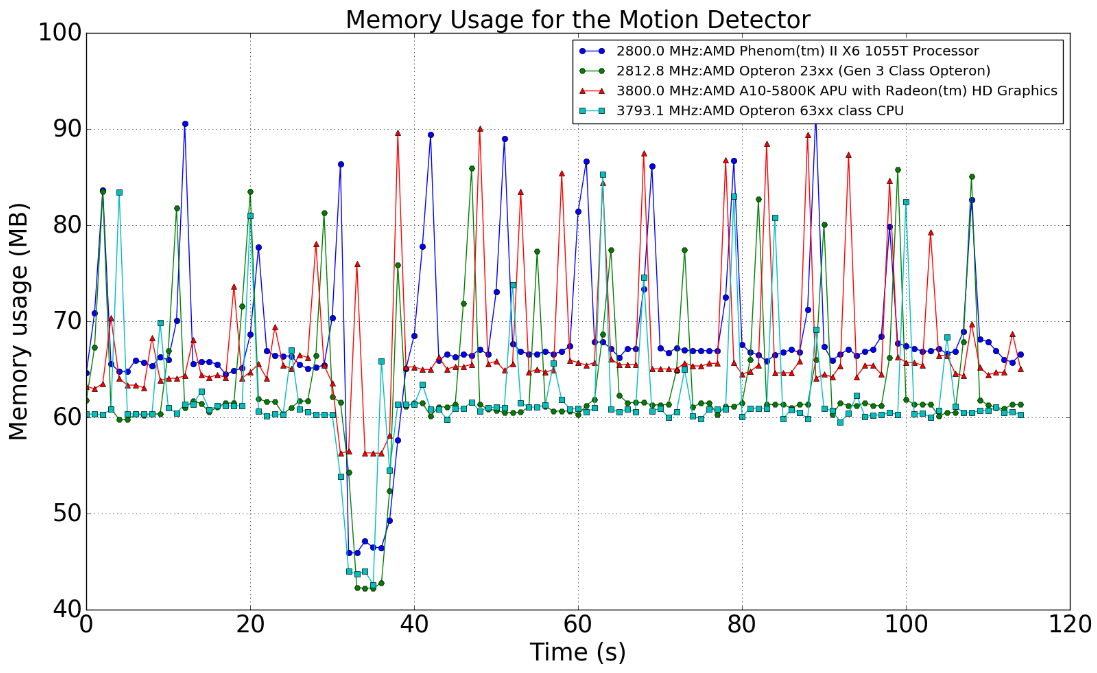
The video processing workload characteristics show that different machine specifications, especially those affecting the CPU model, influence resource usage consumption. In order to support multiple machine specifications, the workload scheduler has to consider CPU and memory usage from the task exploration when assigning a task to a suitable compute node. The scheduler utilizes the exploration results by applying three resource usage criteria, as described in Subsection 3.3.3.

3.3.3 Resource Usage Criteria

Results from the same compute node specification and similar video processing tasks point in the same direction, but it is difficult to apply them to task scheduling. The scheduler needs to use resource usage criteria for computing resources estimation, based on criteria that involve the resource utilization of all the video processors which generally consume resources linearly. The factors for approximating resource usage include the average data set, average minimum data set, and average maximum data set, and the proposed criteria are shown in Figure 3.30.

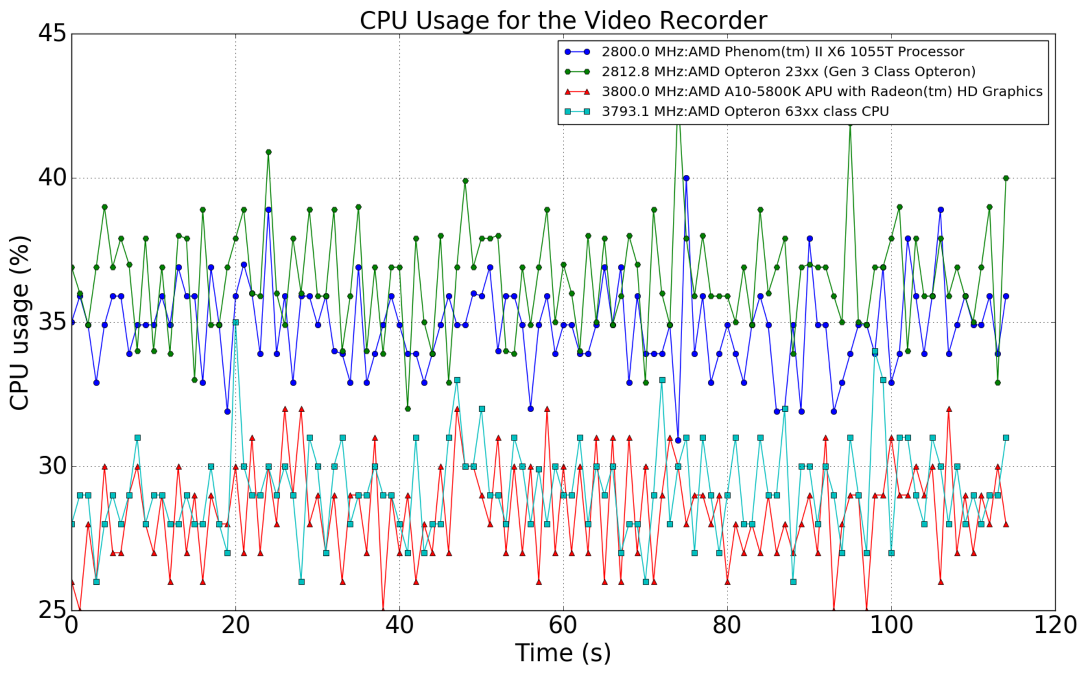


(a) CPU usage for the motion detector

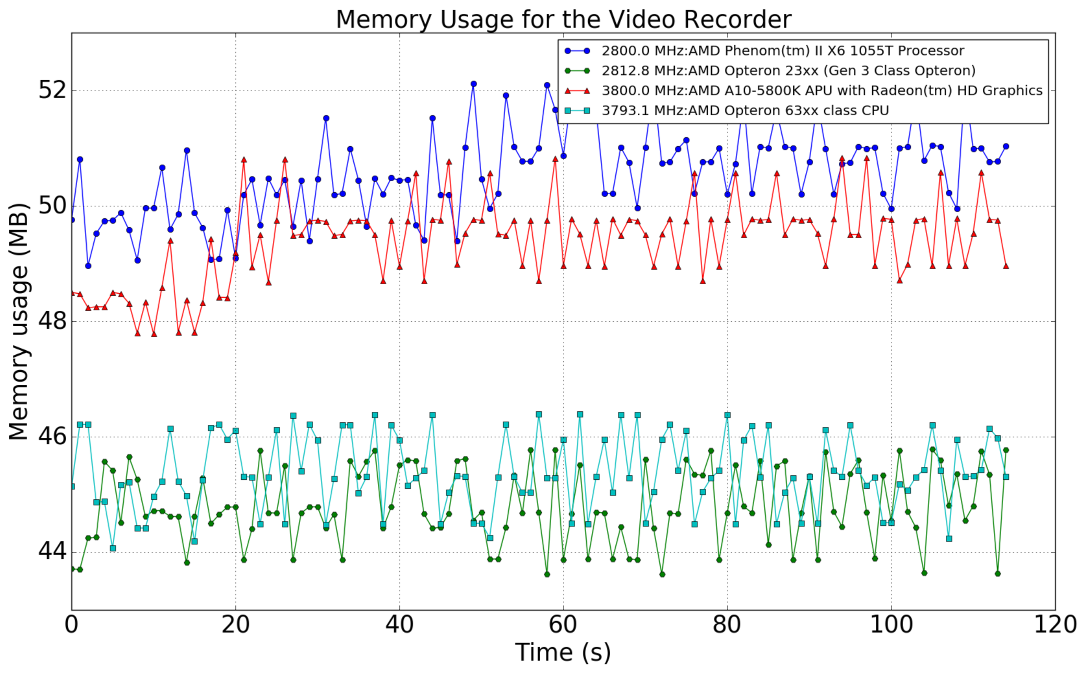


(b) Memory usage for the motion detector

Figure 3.27: Resource consumption for the motion detector running on different physical and virtual machines applying the fixed frame rate of 10 FPS and frame size of 640x480 pixels.

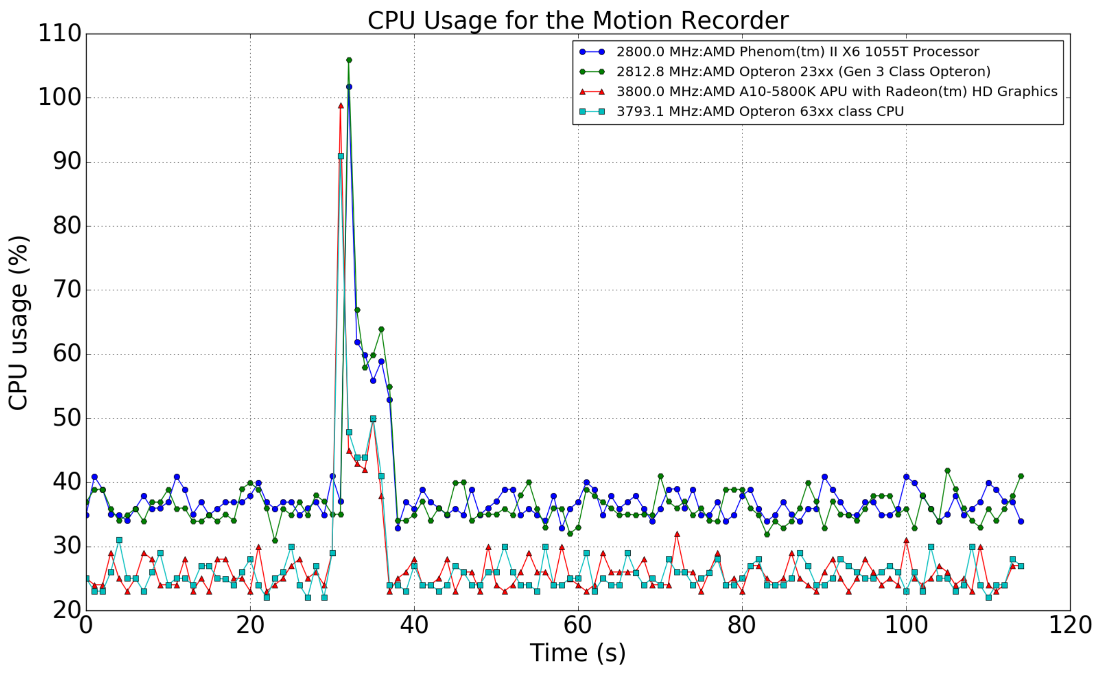


(a) CPU usage for the video recorder

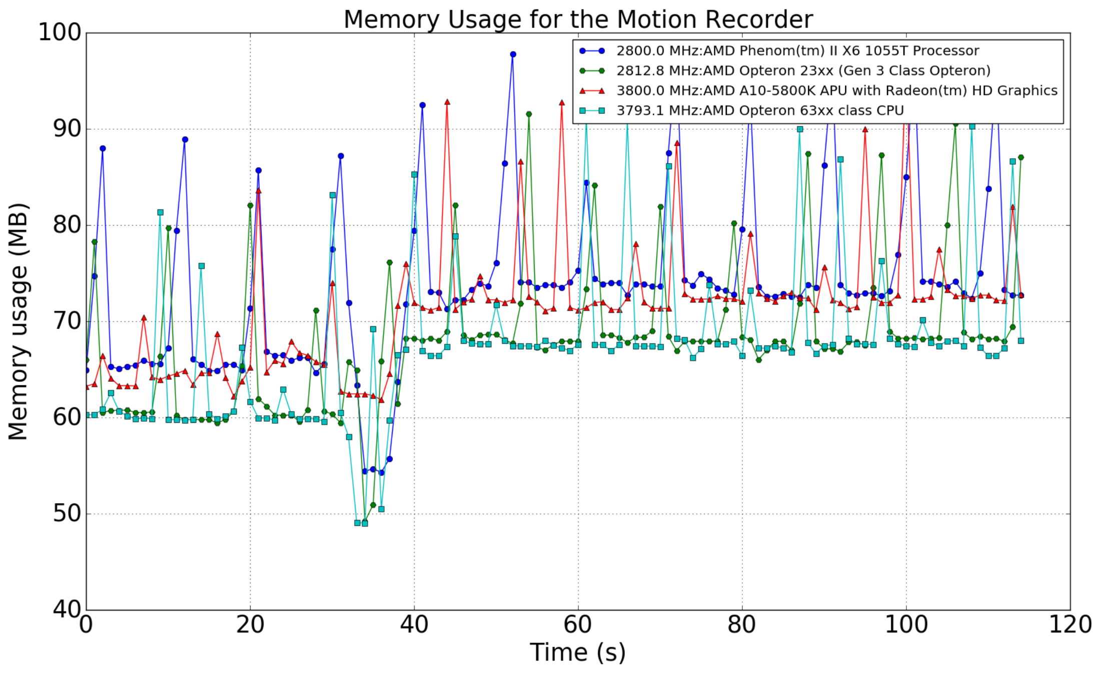


(b) Memory usage for the video recorder

Figure 3.28: Resource consumption for the video recorder running on different physical and virtual machines applying the fixed frame rate of 10 FPS and frame size of 640x480 pixels.



(a) CPU usage for the motion recorder



(b) Memory usage for the motion recorder

Figure 3.29: Resource consumption for the motion recorder running on different physical and virtual machines applying the fixed frame rate of 10 FPS and frame size of 640x480 pixels.

The first criterion is the average data set, representing the data mean suitable for general CPU utilization events. The second criterion is the average minimum data set, which is the data average lower than the mean, and best for heavy CPU utilization although it may affect memory utilization. For example, when the CPU is busy or can not process the image on time, the memory utilization will increase. The last criterion is the average maximum data set, which is the data average higher than the mean. This ensures that the computing resources will be sufficient for the required video processing.

In short, VSaaS administrators must identify their system specification and choose a suitable criterion for highly efficient computing resource management. Section 4.3 presents experimental results applying the proposed three different criteria and discuss suitable exploitations in real situations.

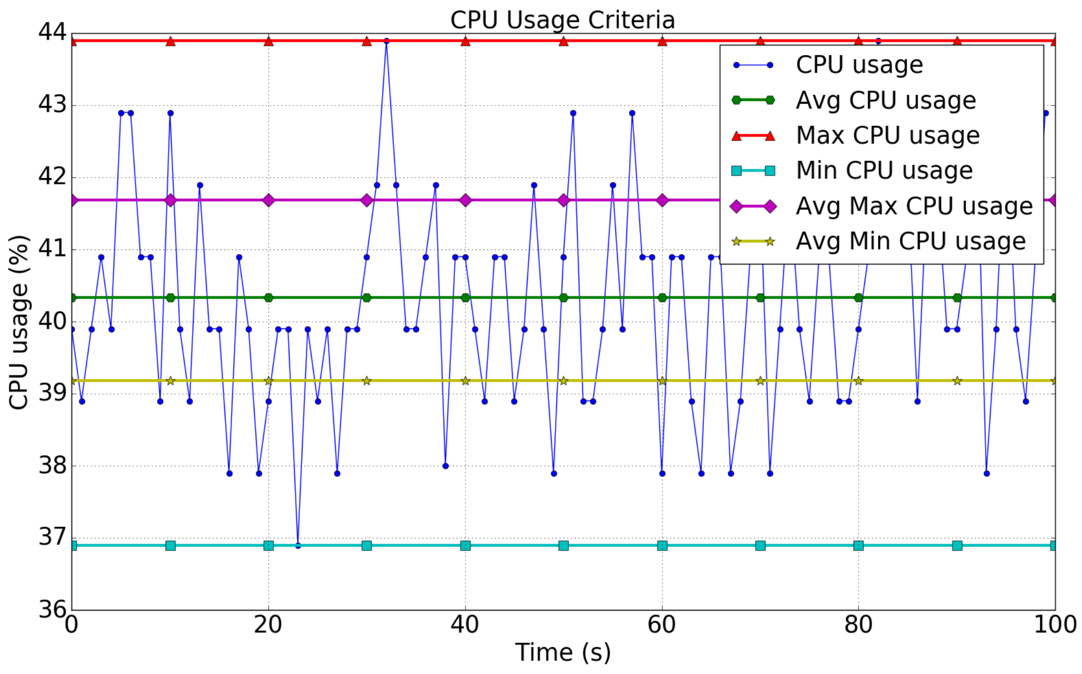
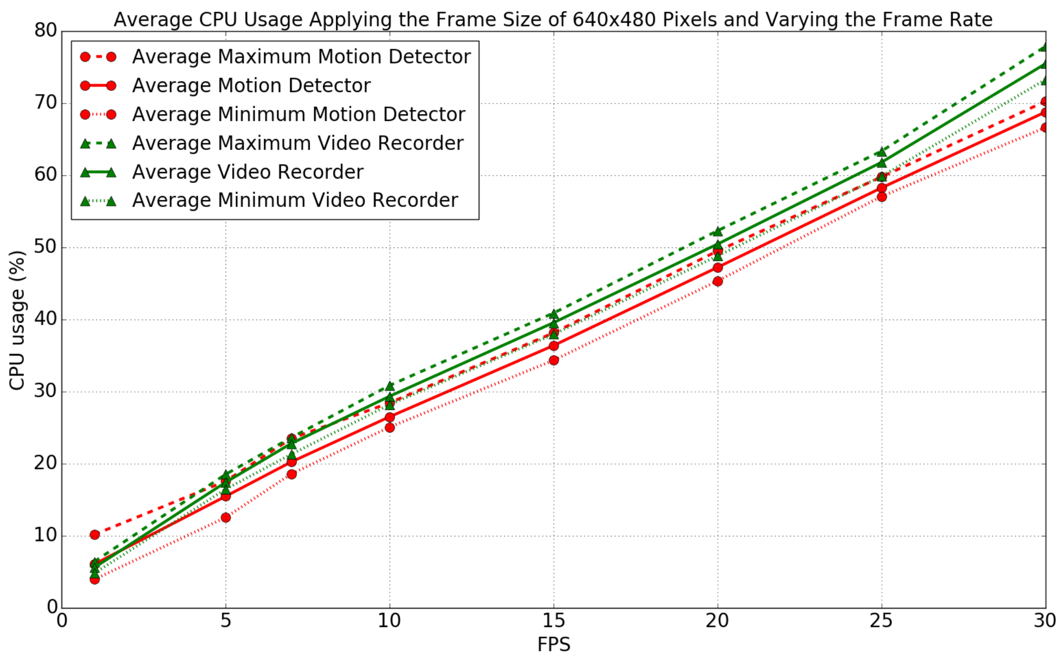
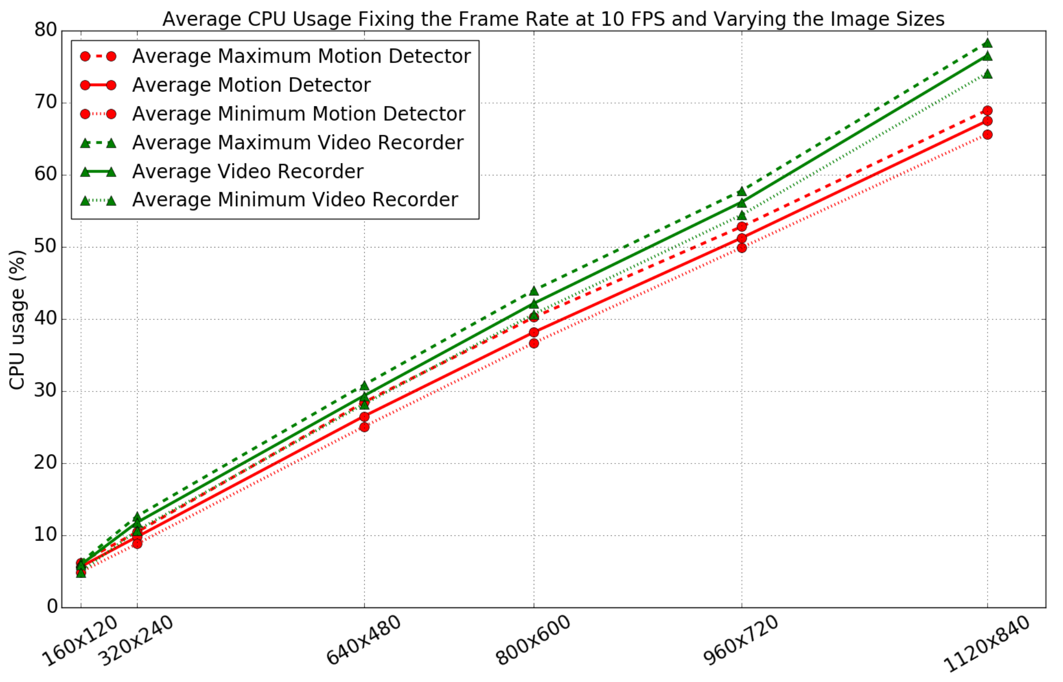


Figure 3.30: Nokkhum resource criteria: the average data set, average minimum data set, and average maximum data set.

The three computing resource usage criteria can also be used to estimate the CPU and memory usage in video processing task scheduling. In this section, only the CPU usage is present as shown in Figure 3.31, because it has the same characteristics as memory usage. Figure 3.31 shows CPU usage at several video frame rates and frame sizes, including motion detection and video recording in Figures 3.31(a) and (b). In Figure 3.31, the average resources usage increases when increasing the frame rate or frame size, and the results can be plotted as straight lines with different slopes.



(a) CPU usage for the three resource usage criteria, varying the frame rate, and fixing the frame size at 640x480 pixels



(b) CPU usage for the three resource usage criteria, varying the frame size, and fixing the frame rate at 10 FPS

Figure 3.31: CPU usage for motion detection and video recorder using the three resource usage criteria.

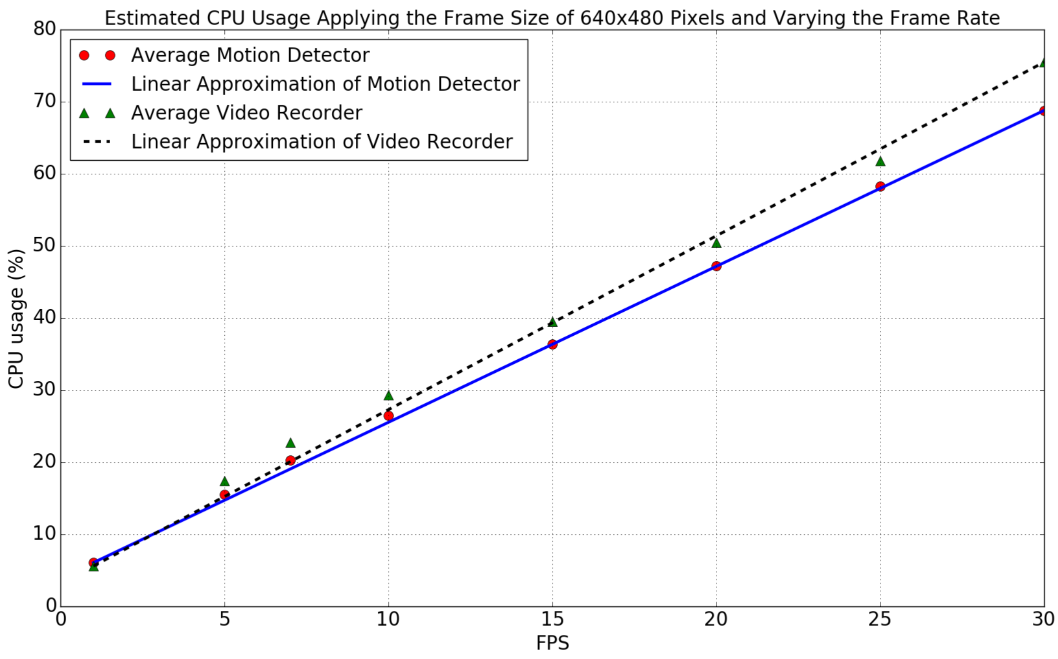


Figure 3.32: Approximating the CPU usage with a linear equation for the CPU AMD-FX, 8 cores, 3.5 GHz

The video processing task exploration can take quite a long time to collect data from the compute nodes, but the execution time can be reduced by exploiting the CPU usage values in the criteria of Figure 3.31. It appears that the average usage is directly related to the frame size and rate. This can then be exploited by determining the linear slope plotted by only taking the sampling points of the video processing tasks from the lowest and highest frame rate or size. For example in Figure 3.32, the frame rates of 1 and 30 FPS are used to approximate the CPU usage, and the result motion detection and video recording equation are presented as Equations (3.1.1) and (3.1.2). Table 3.1 shows the approximated CPU usages from the two equations compared to the real average CPU usages and their absolute errors.

$$\%CPU = 3.90695652x + 2.1626087$$

for motion detection on the CPU AMD-FX, 8 cores, 3.5 GHz (3.1.1)

where x is the frame rate

$$\%CPU = 3.19016492x + 2.40983508$$

for video recorder on the CPU AMD-FX, 8 cores, 3.5 GHz (3.1.2)

where x is the frame rate

Table 3.1: Comparison between the average CPU usage and its approximation for motion detection and video recording at different frame rates for the CPU AMD-FX, 8 cores, 3.5 GHz

Frame rate	Motion detection			Video recorder		
	Average CPU usage	Approxi-mation	Absolute error	Average CPU usage	Approxi-mation	Absolute error
1	6.070	6.070	0.000	5.600	5.600	0.000
5	15.478	14.720	0.758	17.417	15.239	2.178
7	20.243	19.045	1.198	22.809	20.059	2.750
10	26.512	25.533	0.979	29.355	27.289	2.066
15	36.396	36.346	0.050	39.568	39.338	0.230
20	47.237	47.159	0.077	50.462	51.387	0.925
25	58.270	57.972	0.297	61.809	63.436	1.627
30	68.785	68.785	0.000	75.485	75.485	0.000

Table 3.1 shows that the errors when applying Equations (3.1.1) and (3.1.2) are small, with the maximum absolute error about 2.75 %. By applying these equations, the execution time of the video processing exploration task can be reduced from 96 test cases to eight ($2 \times 2 \times 2$) which come from two video processing types (motion detection and video recording), two frame rates (1 and 30 FPS) and two frame sizes (160x120 and 1120x840 pixels), taking the total time of 16 minutes for each machine specification. The linear equation model can only be applied to the same machine. Different specifications may result in different slopes due to different CPU architectures.

3.4 Nokkhum Scheduling

Resource management and the video processing task scheduler are the key Nokkhum VSaaS modules for the availability and resource provision of tasks. This section describes a suitable task scheduler based on the real workloads presented in Section 3.3.

3.4.1 Scheduling Overview

The task scheduler picks a video processing task from a queue, and asks the resource predictor to estimate the computing resource requirements for the task. After the predictor has returned an available computing node, the scheduler calls the task controller to spawn the video processing task. The Nokkhum scheduling process is shown in Figure 3.33.

The Nokkhum scheduler evaluates which compute node worker is suitable to run a video analysis task. At first, it gets a video processing task from the task queue, finds an available compute node worker, and passes it to the resource predictor. The predictor

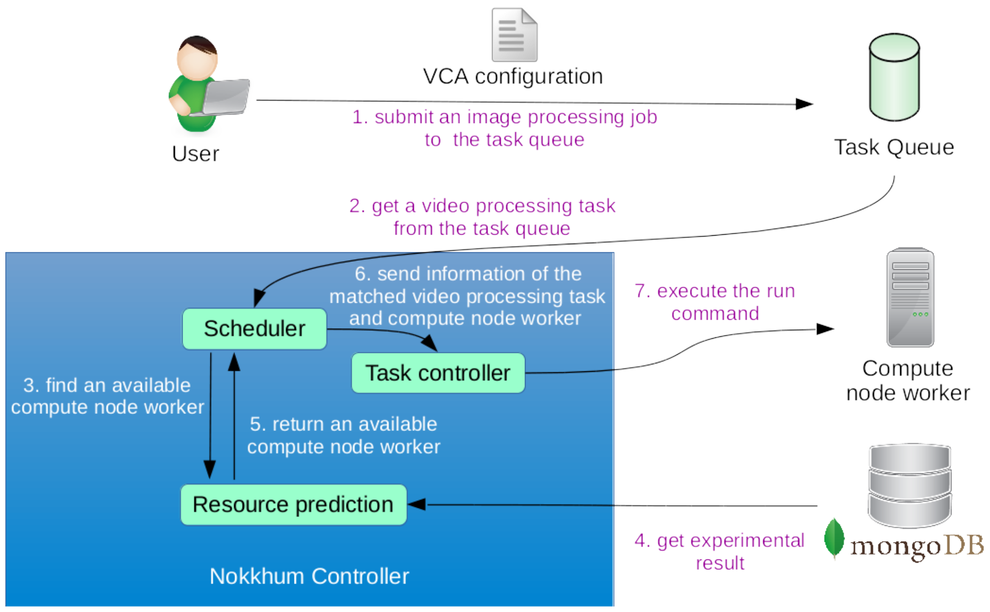

VCA configuration
User
1. submit an image processing job to the task queue
Task Queue
2. get a video processing task from the task queue
6. send information of the matched video processing task and compute node worker
7. execute the run command
Scheduler
Task controller
Compute node worker
3. find an available compute node worker
5. return an available compute node worker
Resource prediction
4. get experimental result
mongoDB
Nokkhum Controller

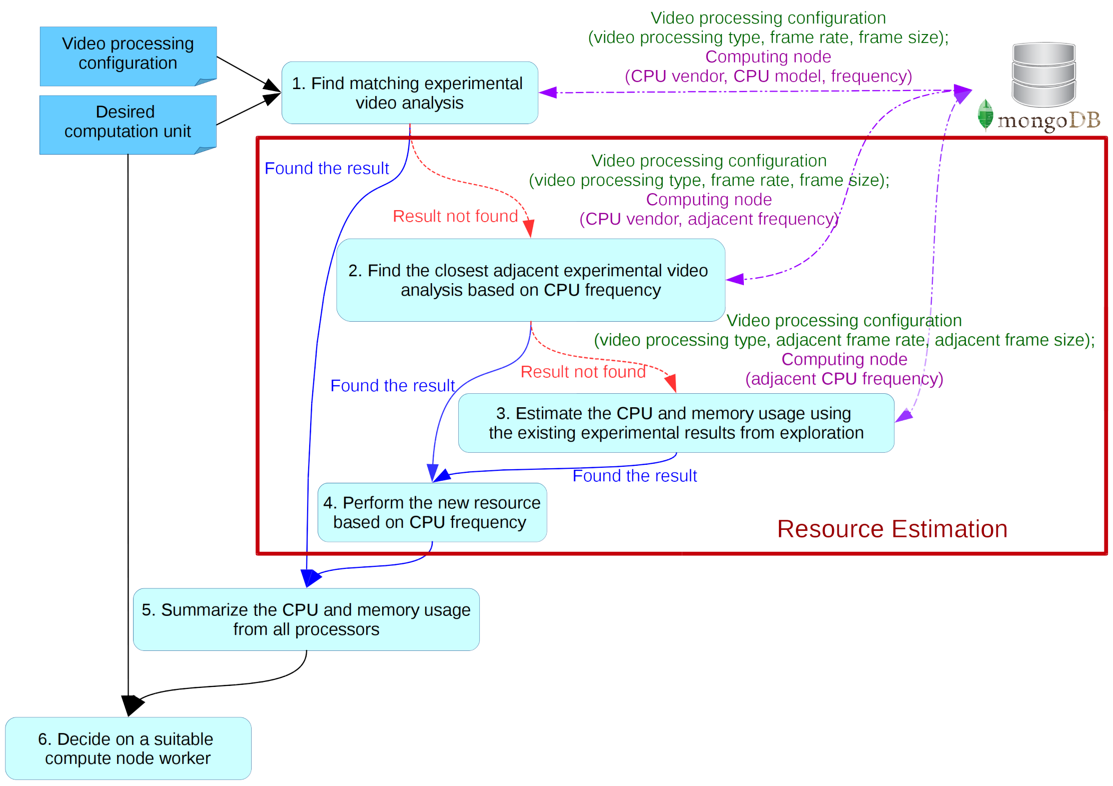
Figure 3.33: Overview Nokkhum scheduling.

then looks for the most suitable experimental workloads from the database and returns the best fit compute node worker to the scheduler. After that, the scheduler sends the video processing configuration and compute node worker information to the task controller. Finally, the controller communicates with the compute node worker to run the video processing task. The Nokkhum resource decision process is described in details in Section 3.4.2.

Resource prediction is part of the compute node controller. It responds to the task scheduler by providing an available compute node worker suitable for the video processing task. It performs resource estimation based on data from the previously mentioned experimental results, especially the CPU and memory usage information. This estimation is used to determine which compute node is suitable for running the required video processing task. The experimental results can be collected from the task exploration, which can be run on any machine specification, and are stored in a database for the next round of scheduling. The resource prediction process is shown in Figure 3.34.

3.4.2 Resource Decision Description

The Nokkhum resource decision module for placing a video processing task on a suitable computing unit is shown in Figure 3.34. The module requires both the desired computation unit and the video processing configuration. The resource decision steps are described as follows.



Video processing configuration
Desired computation unit
1. Find matching experimental video analysis
Video processing configuration (video processing type, frame rate, frame size); Computing node (CPU vendor, CPU model, frequency)
mongoDB
Found the result
Result not found
Video processing configuration (video processing type, frame rate, frame size); Computing node (CPU vendor, adjacent frequency)
2. Find the closest adjacent experimental video analysis based on CPU frequency
Video processing configuration (video processing type, adjacent frame rate, adjacent frame size); Computing node (adjacent CPU frequency)
Found the result
Result not found
3. Estimate the CPU and memory usage using the existing experimental results from exploration
Found the result
4. Perform the new resource based on CPU frequency
Resource Estimation
5. Summarize the CPU and memory usage from all processors
6. Decide on a suitable compute node worker

Figure 3.34: The Nokkhum resource management decision process.

Step 1) Find matching experimental video analysis

The resource decision module finds a video processing task with a matching video processing type, frame rate, and frame size from the previously allocated computation unit (of the same CPU vendor, model, and frequency). If it finds a result, it skips Steps 2 - 4 and goes to Step 5.

Step 2) Find the closest adjacent experimental video analysis based on CPU frequency

If the resource decision module cannot find an exactly matching experimental result in the database, it requests a new experimental result with an adjacent CPU frequency and ignores the CPU model of the desired compute node, but the video processing type, frame rate, and frame size must still match. The example in Figure 3.35 shows that it always chooses the next available CPU with a higher frequency than the desired one. If a result is found, it will skip Step 3.

Step 3) Estimate the CPU and memory usage using the experiment results

This step looks for an experimental result in the database which has the same video processing type, and passes the result to Step 4. The result may not be accurate,

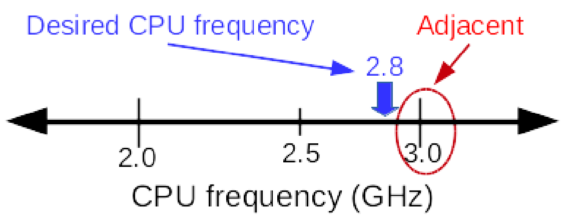


Figure 3.35: Selecting a CPU frequency adjacent to the desired frequency.

and may not fit the task well, but it is better than scheduling with no estimation at all.

Step 4) Perform the new resource estimation based on CPU frequency

Step 4 analyzes the experimental results shown in Figures 3.24, 3.25, and 3.26. The task's CPU usage depends on the CPU frequency. A higher clock frequency means a lower CPU usage by the task, depending on the CPU model and specification, although memory usage is rather consistent. The resource decision module estimates the CPU and memory usages from the experimental result equivalent to the desired computation unit's CPU frequency. It applies a scaling factor (SF) as described by Equation (3.2) to the CPU usage as shown in Equation (3.3.1).

$$SF = \begin{cases} \dfrac{F_{desired}}{F_{experiment}} & \text{if } F_{experiment} >= F_{desired} \\[2em] \dfrac{F_{experiment}}{F_{desired}} & \text{otherwise} \end{cases} \tag{3.2}$$

where SF is the CPU scaling factor;

$F_{desired}$ is the CPU frequency of the desired computation unit;

$F_{experiment}$ is the CPU frequency of the computation unit running the experiment.

Equation (3.2) calculates a scaling factor using the ratio of the compute node CPU frequency and the adjacent frequency determined from the experimental results in the database. It returns different ratios whether the adjacent frequency is higher or lower than the CPU frequency of the desired compute node. The scaling factor is used to estimate adjacent CPU usages, according to the resource usage criterion shown in Equations (3.3.1) and (3.3.2). The estimated memory usage is set to the memory usage of the experiment, as in Equation (3.3.2). The CPU and memory usages are derived from the resource usage criteria as presented in Section 3.3.3. The estimated CPU and memory usages ($CPU_{estimated}$ and $memory_{estimated}$) are used in the next step.

$$CPU_{estimated} = SF \times CPU_{experiment}[criterion] \tag{3.3.1}$$

where $CPU_{experiment}[criterion]$ is the CPU usage, collected from the experiment, selected by the $criterion$; $criterion$ is a resource usage policy identified by the administrator.

$$memory_{estimated} = memory_{experiment}[criterion] \tag{3.3.2}$$

where $memory_{experiment}[criterion]$ is the memory usage from the experiment selected by applying the $criterion$.

Step 5) Summarize the CPU and memory usages from all processing tasks

The Nokkhum video processor components are sequentially connected to each other. The resource decision module summarizes the CPU and memory usages of all the video tasks from the previous steps to aid decision making in the final step. For example, a motion recorder consists of a motion detector and a video recorder; the resource decision module sums the CPU and memory usages from both processors for consideration in the next step.

Step 6) Decide the suitable compute node worker

The final step decides which compute node is suitable. The resource decision module acquires the real workload of the compute node and approximates the current capacity of the running task. The approximated current resource usage of the target compute node and the task information from Step 5 are used to decide if it is suitable. The module will return an appropriate compute node to the task scheduler to start the task. The calculations are presented by Equations (3.4.1), (3.4.2) and (3.4.3).

$$T_{CPU} = \sum_{n=1}^{N} CPU_{e_{(n)}} + CPU_{e_p} \tag{3.4.1}$$

where T_{CPU} is the summation of $CPU_{estimated}$ of all the video processor tasks running on the compute node C;

$CPU_{e_{(n)}}$ is the estimated CPU usage of the video processing task n;

CPU_{e_p} is $CPU_{estimated}$ of the requested video processing task p that will be executed by the compute node C;

N is the number of video processing tasks contained in the compute node C.

$$T_{memory} = \sum_{n=1}^{N} memory_{e_{(n)}} + memory_{e_p} \tag{3.4.2}$$

where T_{memory} is the summation $memory_{estimated}$ of all the video processor tasks running on the compute node C;

$memory_{e_{(n)}}$ is the estimated memory usage of the video processing task n;

$memory_{e_p}$ is $memory_{estimated}$ of the requested video processing task p that will be executed by the compute node C;

N is the number of video processing tasks contained in the compute node C.

$$D(C) = \begin{cases} True & \text{if } T_{CPU} < C.cpu\_core * 100, \\ & \text{and } T_{memory} < C.total\_memory \\ \\ False & \text{otherwise} \end{cases} \tag{3.4.3}$$

where $D(C)$ is the decision function for the compute node qualification;

C is the desired compute node.

Equations (3.4.1), (3.4.2) and (3.4.3) determine a compute node suitable for executing the video processing task. T_{CPU} and T_{memory} are summations of the estimated CPU and memory usages ($CPU_{estimated}$ and $memory_{estimated}$) of all the video processing tasks (N) containing the candidate compute nodes and selected video processing tasks (CPU_{e_p} and $memory_{e_p}$). The decision function (D) of compute node C employs T_{CPU} and T_{memory} to determine whether the compute node C can run the video processing task p. T_{CPU} must be lower than the number of cores multiplied by 100%, and T_{memory} must be lower than the maximum memory of the desired compute node. If there is no compute node with the desired specification, the decision function will return $False$. The resource decision algorithm for placing tasks is also presented as pseudocode in Algorithm 1.

Algorithm 1 presents three functions for task scheduling: GetSuitableComputeNode, EstimateComputingResources, and GetVPTExperiment. GetSuitableComputeNode validates whether a compute node is suitable for starting a task using a compute node information and a video processing configuration. It checks an available compute node using EstimateComputingResources which gathers approximated computing resources from the current task running on the candidate compute node. The approximated computing resource is compared with the estimated video processing resource taken from the video processing task configuration. EstimateComputingResources summarizes the CPU and memory usage sums from all the video processing configurations using GetVPTExperiment. It queries the experimental VCA from the database following the steps shown and described in Figure 3.34 and Section 3.4.2 respectively.

The scheduler orders the tasks based on real resource usage, but it is difficult to evaluate the computing resources for such data. Therefore, the decision algorithm is driven by resource utilization, so the system administrator can guide the scheduler with the resource usage criteria described in Subsection 3.3.3.

```
input  : A set of available compute nodes
input  : A VCA configuration
output: A suitable compute node
Function GetSuitableComputeNode (compute-nodes, p)
    foreach c in compute-nodes do
        cEstimateCPU ← 0; cEstimateMemory ← 0;
        foreach cp in c.processors do
            subcpu, submemory= EstimateComputingResources
              $(c, cp)$;
            cEstimateCPU = cEstimateCPU + subcpu;
            cEstimateMemory = cEstimateMemory + submemory;
        end
        processorCPU, processorMemory =
          EstimateComputingResources (c, p);
        if processorCPU $+$ cEstimateCPU $<$ 100 $\times$ c.cpucore and
          processorMemory $+$ cEstimateMemory $<$ c.maxmeory then
          return $c$ ;
    end
    return None
End
Function EstimateComputingResources (c, p)
    totalCPU ← 0; totalMemory ← 0;
    foreach p in p.processors do
        experiment = GetVPTExperiment $(c, p)$;
        totalCPU ← totalCPU + experiment.cpu[criterion];
        totalMemory ← totalMemory + experiment.memory[criterion];
        if "processors" in p then
            subcpu, submem = EstimateComputingResources
              $(c, p.processors)$;
            totalCPU ← totalCPU + subcpu; totalMemory ←
              totalMemory + submem;
        end
    end
    return totalCPU, totalMemory
End
Function GetVPTExperiment (c, p)
    cpu ← $c.CPU$;
    experiment ← GetVPTExFromDatabase (cpu.vender, cpu.model,
      cpu.frequency, p.vcatype, p.framerate, p.imagesize);
    if experiment == None then
        experiment ← GetVPTExFromDatabase (cpu.vender,
          ∼cpu.frequency, p.vcatype, p.framerate, p.imagesize);
        // ∼cpu.frequency is the selected CPU frequency that is an adjacent
          target CPU frequency
    end
    if experiment == None then
        experiment ← GetVPTExFromDatabase (∼cpu.frequency,
          p.vcatype, p.framerate, ∼p.imagesize);
        // ∼p.imagesizes is the selected image size that is adjacent target
          image size
    end
    return experiment
End
```

Algorithm 1: Nokkhum task scheduling

3.5 Summary

This chapter presented the system design and implementation of Nokkhum VSaaS. The Nokkhum system consists of five components performing different functions which can be distributed are several servers. It can deploy all the elements in a single server for a small business, or many servers to support a large number of IP cameras. Moreover, video processing workloads are analyzed applying various factors including the kind of video processing, video frame rate, video frame size, and type of computing units. The analysis results are used to guide video task scheduling for placing a new task into a suitable compute node. The task scheduler ensures the compute node sufficiently consumes the computing resources.

CHAPTER 4

RESULTS AND DISCUSSIONS

This chapter mainly presents and discusses the results of the Nokkhum VSaaS testing and video task scheduling. First, Infrastructure setting presents essential specification and environment setting for this experimental results. Second, Nokkhum VSaaS system testing includes the scalability and flexibility checklist, and system response time. The next section shows the video task scheduling results, varying the number of video processing tasks placed on a computing unit with various video processing configurations.

4.1 Infrastructure Setting

The general specification and cloud environment setting for testing the flexibility and scalability of Nokkhum system and video processing task scheduling are presented in this section. The general specification is a common infrastructure for all experiments. The cloud environment setting describes the essential software and physical machine for providing cloud environment to evaluate the system's flexibility and scalability.

4.1.1 General Specification

This section presents physical and virtual machine specification, camera specification, video record format, and software using in this thesis as follows:

1. Six physical machines including three AMD and three Intel CPU as shown in Table 4.2 as computing machines. Two gigabit Ethernet interface is employed for both internal and public networks.

2. Six VMs related to physical machines employing 2-core CPU and 2 GB of RAM as shown in Table 4.3

3. OpenStack Icehouse [61] with KVM [42] as the hypervisor for IaaS infrastructure. The VM image format is QEMU Copy On Write (QCOW2) [62].

4. Camera and video inputs shown in Table 4.1.

5. Video processing exploration using the VIRAT video data set with following parameters.

- Eight frame rates: 1, 5, 7, 10, 15, 20, 25, and 30 FPS
- Six frame sizes: 160x120, 320x240, 640x480, 800x600, 960x720, and 1120x840 pixels.
- Video processings: motion detection, video recording, and motion recording.
- Ogg codec used as the default codec for video output.

6. RabbitMQ [63][64] is MOM for providing the OpenStack and Nokkhum system.

Table 4.1: IP camera vendors and video input formats

Vendor	Model	Video Size (pixels)	Frame Rate (FPS)
D-Link	DCS-930L	160x120	1, 5, 7, 15, 20, 30
		320x240	1, 5, 7, 15, 20, 30
		640x480	1, 5, 7, 15, 20
	DSC-2102	160x120	5, 10, 15, 30
		302x240	5, 10, 15, 30
		640x480	5, 10, 15, 30
AXIS	215 PTZ	640x480	10
	210	640x480	10
	211M	640x480	10

Table 4.2: Physical machines used in the experiments.

PM Code	CPU			Total Memory (GB)	Total Disk (GB)
	Model	Frequency (Hz)	Cores		
AMD-2.8-PM	AMD Phenom(tm) II X6 1055T Processor	2800	6	8.11	98.29
AMD-3.5-PM	AMD FX(tm)-8320 Eight-Core Processor	3500	8	16.55	98.29
AMD-3.8-PM	AMD A10-5800K APU with Radeon(tm) HD Graphics	3800	4	16.27	1850.37
Intel-2.6-PM	Intel(R) Core(TM)2 Quad CPU Q9400 @ 2.66GHz	2670	4	4.15	285.31
Intel-2.8-PM	Intel(R) Xeon(R) CPU X3360 @ 2.83GHz	2834	4	4.15	226.78
Intel-3.4-PM	Intel(R) Core(TM) i7-2600 CPU @ 3.40GHz	3800	4	4.13	473.86

Table 4.1 presents IP camera vendors and video formats used in this thesis. In this thesis, the experiments emphasize on the video sizes of 320x240 and 640x480 pixels,

Table 4.3: Virtual machines used in the experiments.

VM Code	PM Code	CPU			Total Memory (GB)	Total Disk (GB)
		Model	Frequency (Hz)	Cores		
AMD-2.8-VM	AMD-2.8-PM	AMD Opteron 23xx (Gen 3 Class Opteron)	2812.792	2	2.10	20.09
AMD-3.5-VM	AMD-3.5-PM	AMD Opteron 63xx class CPU	3515.784	2	2.10	20.09
AMD-3.8-VM	AMD-3.8-PM	AMD Opteron 63xx class CPU	3793.102	2	2.10	20.09
Intel-2.6-VM	Intel-2.6-PM	Intel Core 2 Duo P9xxx (Penryn Class Core 2)	2666.362	2	2.10	20.09
Intel-2.8-VM	Intel-2.8-PM	Intel Core 2 Duo P9xxx (Penryn Class Core 2)	2833.530	2	2.10	20.09
Intel-3.4-VM	Intel-3.4-PM	Intel Xeon E312xx (Sandy Bridge)	3391.448	2	2.10	20.09

and the frame rates of 10, 15 and 30 FPS. Table 4.2 shows physical machine specifications and their codes. The machine codes are used in Tables 4.3 and referred to in the computing resource usage report in Table 4.9. Table 4.3 shows the VM specifications based on the physical machines from Table 4.2.

4.1.2 Cloud Environment Setting

For the cloud testing environment, this thesis employs the OpenStack service family on generic PCs (AMD-2.8-PM and AMD-3.5-PM). For VM providers, OpenStack Nova and KVM are chosen to work as the hypervisor. The physical cloud structure is implemented by a PC acting as a cloud controller (AMD-2.8-PM), and many PCs (AMD-2.8-PM and AMD-3.5-PM) utilized as cloud compute nodes. The connection is done by using two Ethernet interfaces. The first interface is for internal communication, and the other for public use via a 10/100/1000 Mb Ethernet. The public interface connects all the cloud controllers, cloud compute node workers, cloud storage, and IP cameras. The cloud environment setting is shown in Figure 4.1.

Nokkhum VSaaS is designed to run on a cloud infrastructure, and all the possible topology configurations shown in Figures 3.12 - 3.18 have been tested successfully in the cloud infrastructure testing system. The system's response time for a medium sized VSaaS (the scenario of Figure 3.15) which supports 10-30 VMs of compute node workers, depending

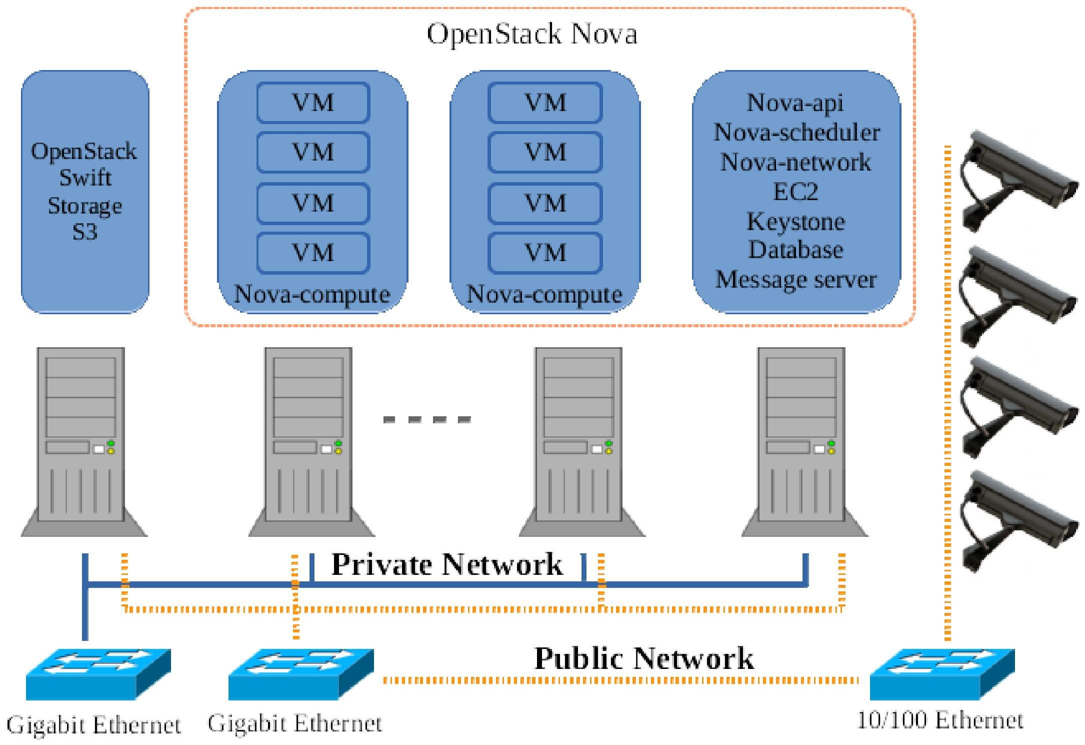


Figure 4.1: Cloud environment setting using OpenStack.

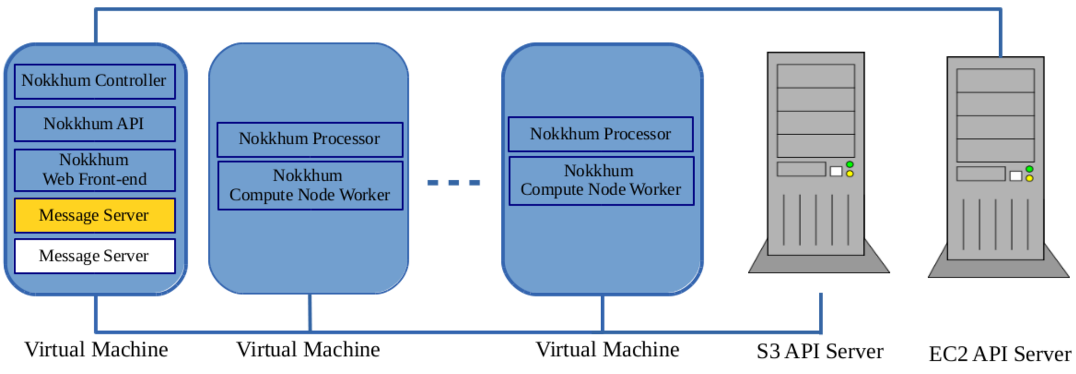


Figure 4.2: Nokkhum VSaaS on cloud infrastructure.

on the capability of the CPUs and memory of the controller node, have been thoroughly investigated. Its deployment is shown in Figure 4.2. The Figure 3.15 scenario has been emphasized because it is a good representative of the topology configurations typically used by many organizations, departments, and universities for providing video surveillance. Figure 4.2 shows a VM as a controller node, containing a message server (RabbitMQ), databases (MongoDB), a web interface, an API and a controller. In addition, each compute node VM image contains a compute node worker and a video processor. The IP address of the message server in the VSaaS system must be identified in the compute node worker configuration. After registering the compute node image in the cloud infrastructure, the compute node image name can be added to the controller and started to run on the VSaaS system.

4.2 Nokkhum VSaaS System Testing

The system testing in this section focuses on the response times of the Nokkhum VSaaS in the cloud environment. This thesis utilizes general-purpose PCs for building the cloud infrastructure with OpenStack, Icehouse version, as the cloud middleware. The PCs used two different CPUs, such as AMD-2.8-PM and AMD-3.5-PM, and their memory range from 8 to 16 GB, with 1 TB of storage. The experiments have been designed to create an underlying performance matrix including the VM acquisition time, waiting time, and processing time for both user and system request commands. The Nokkhum VSaaS is based on the design described in Section 3.2, and its cloud environment set up is explained in Section 4.1.2. All testing descriptions are described as follows:

1. Computing nodes employ AMD-2.8-VM and AMD-3.5-VM as VM templates.
2. Twenty IP cameras are employed including models of the Dlink DSC-930L, Dlink DSC-2102, AXIS 215 PTZ, and AXIS 211M.
3. All input video formats based on the IP cameras which are widely spread in the market including the frame sizes of 160x120, 320x240, and 640x480 and the frame rates of 5, 7, 15, 20, and 30.
4. The experiments involve many users and multiple video processing configurations such as motion detection and video recording.
5. The testing is repeated ten times on a random sequence of video processing suits.

4.2.1 Scalability and Flexibility Checklists

The testing results confirm that the Nokkhum VSaaS system performs according to the system design as shown in the scalability and flexibility check lists in Tables 4.4 and 4.5. In terms of scalability, the system can automatically acquire VMs, provide services for various user groups, support IaaS, and be scaled up by distributing its components across VMs and physical machines. In terms of flexibility, the system can automatically handle different situations concerning camera suspension, and malfunctioning, or unavailable compute node workers, controller, and message server.

Table 4.4: Scalability checklist.

List	Check
Acquire VMs automatically	✓
Provide services for various user groups	✓
Support IaaS with Amazon EC2/S3	✓
Components can scale up and down according to Figures 3.12 - 3.18	
A: compact system on one machine	✓
B: add a compute node worker	✓
C: compute node workers are separated from the controller	✓
D: separate cloud storage service	✓
E: separate database and message server	✓
F: separate front-end node	✓
G: fully distributed system	✓

Table 4.5: Flexibility checklist.

List	Check
System automatically recovers after a suspended IP camera becomes available	✓
System handles compute node worker errors (either electricity or network problem) and acquires a new one	✓
Compute node worker continues running when controller node is unavailable	✓
Controller node can recover its status when it resumes running	✓
Controller and compute node worker can continue running when the message server is unavailable	✓

4.2.2 System Responding Time

The Nokkhum VSaaS is based on the design described in Section 3.2, and its cloud environment set up is explained in Section 4.1.2. The minimum video recording space per camera per day was varied from 65 KB to 210 MB. The total video recording space per camera used each day was varied from 361MB to 19 GB.

The Nokkhum front-end node was executed as a single VM and the Nokkhum compute unit utilized the 3 GB QCOW2 disk format. The virtual hardware template for instance acquisition was equipped with a 2-core CPU, 2.10 GB RAM, and a hard disk of 20.09 GB. The experimental results are shown in Table 4.6 and in Figures 4.3 - 4.8.

Table 4.6: Virtual machine acquisition time.

Activity	Time (s)		
	Min	Max	Average
Instance spawning with image cache	11.150	11.470	11.265
Instance spawning without image cache	104.930	117.011	112.229
Instance booting	13.220	19.379	16.081

Table 4.6 shows the acquisition time for OpenStack Nova to provide instances. The spawning time is the time to transfer a VM image from the OpenStack Glance server to the destination Nova Compute Node Worker, plus the time to prepare the image for booting. The image cache plays an important role in instance spawning. If the Nova Compute Node Worker has previously run an instance, it may cache its image for a later run. This reduces the image transferring time so it can boot much sooner. The instance booting time is the interval from when the VM is first active to when the Nokkhum controller gets its first response from the Nokkhum compute node worker. The VM acquiring average time (spawning and booting time) was 128.31 seconds without image cache, and 27.346 seconds with caching.

In this experiment, there were two types of message commands for controlling the starting and stopping of the Nokkhum processor. Firstly, User Request Command (URC) messages are generated when the user requests the starting or stopping of video processing. Secondly, System Request Command (SRC) messages are generated by the task controller module in the Nokkhum controller when the video processor crashes. The task controller adds an SRC to the command waiting queue to start the video processor. The waiting and command processing times for an SRC indicate the system's ability to serve user requests.

Figures 4.3 and 4.4 show the waiting time histograms for URCs to start and stop video processors. Figures 4.5 and 4.6 show the processing time duration histograms for URCs to start and stop video processors. The command waiting time histograms contain a range of different time distributions due to several causes: the task scheduler is a single thread performing sequential processing, the user submits requests for several sequential actions in a too short time, and the user command action must wait for an available compute node. The waiting time for a URC to start a video processor begins when the system starts loading the video processor binary and finishes when the video processor gets its first image from the video connection. The image acquisition time varies depending on the IP camera type, camera

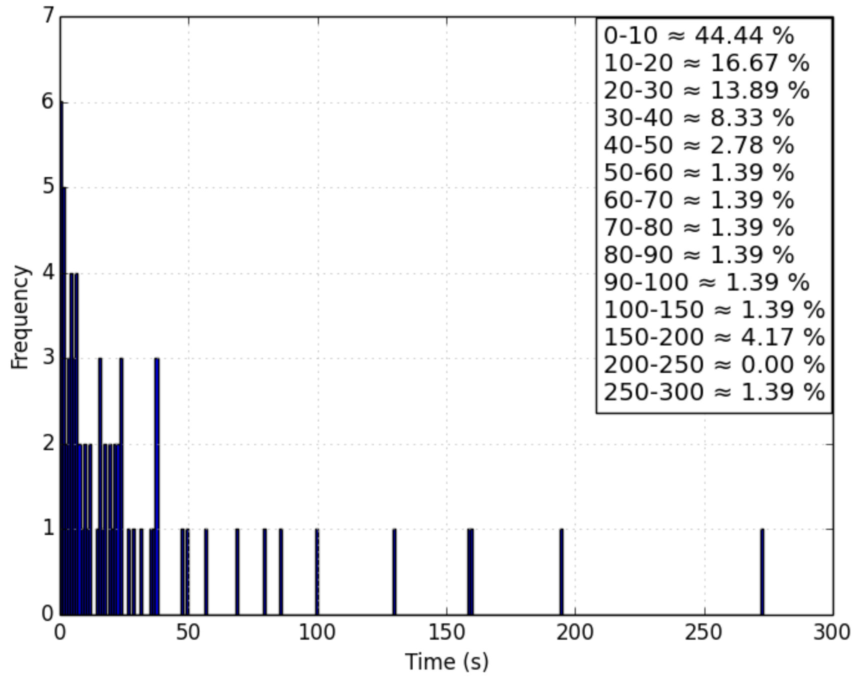


Figure 4.3: URC waiting time for starting video processors.

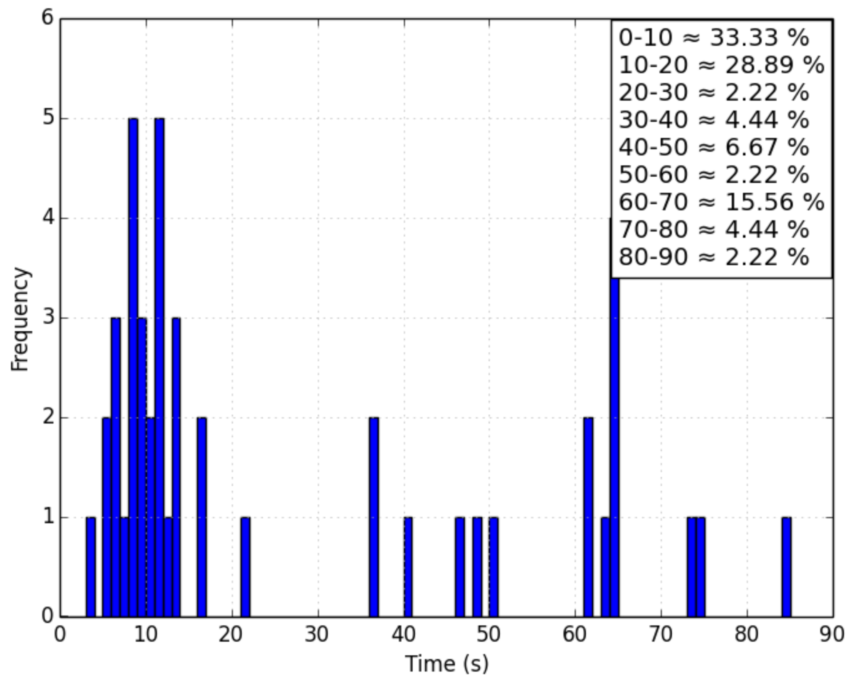


Figure 4.4: URC waiting time for stopping video processors.

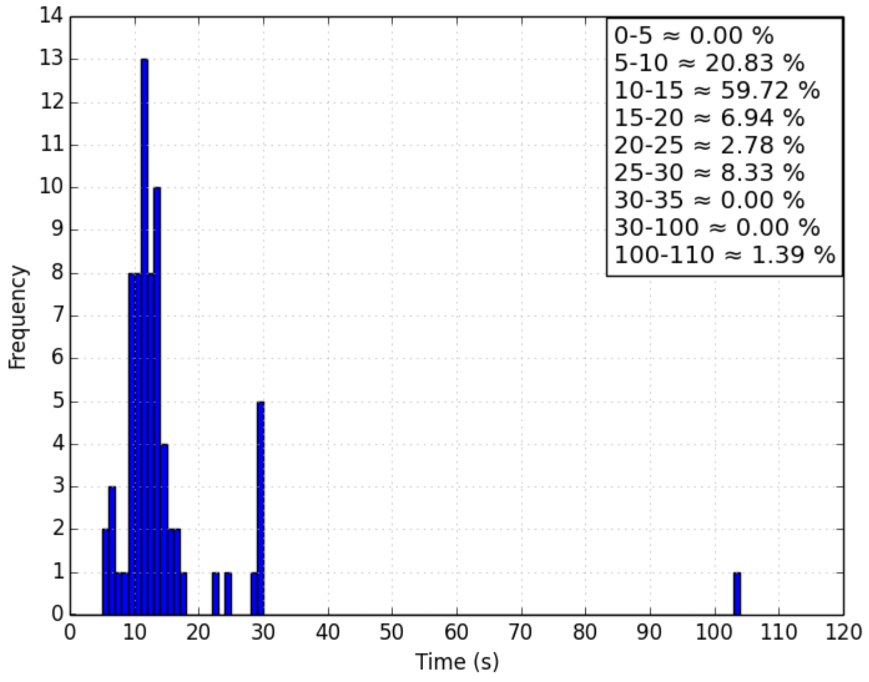


Figure 4.5: URC processing time for starting video processors.

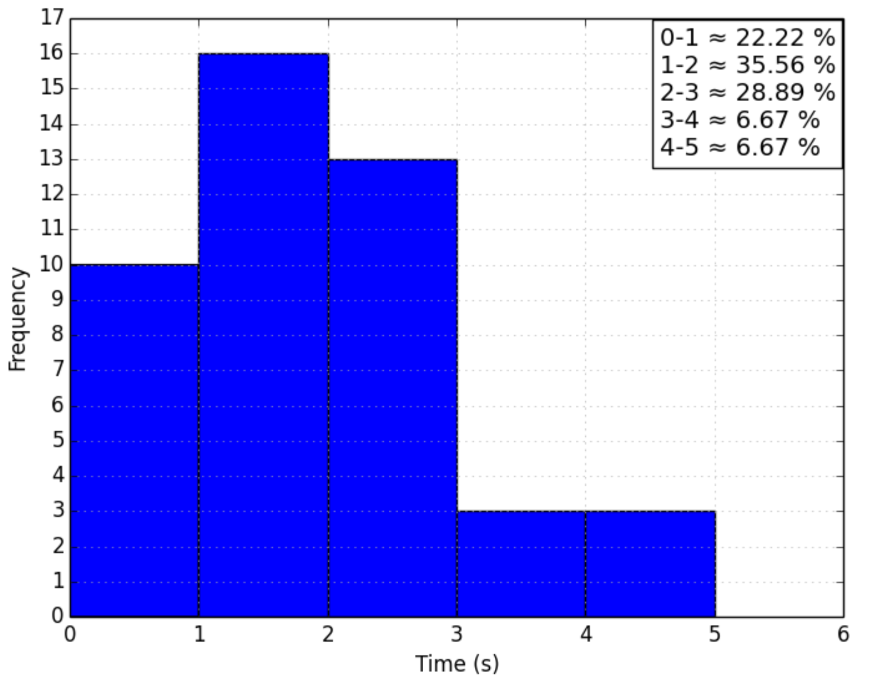


Figure 4.6: URC processing time for stopping video processors.

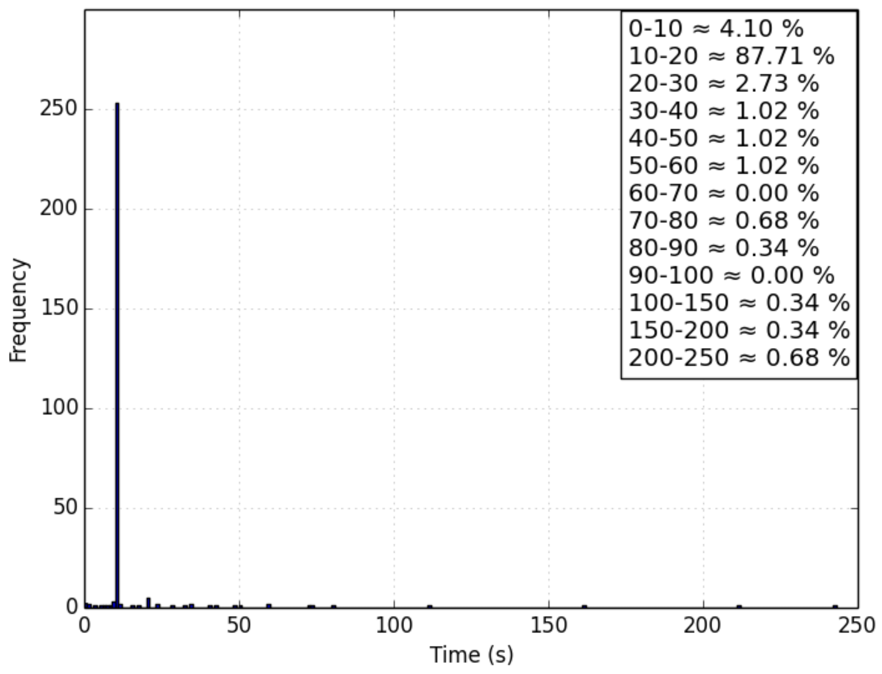


Figure 4.7: SRC waiting time for starting video processors.

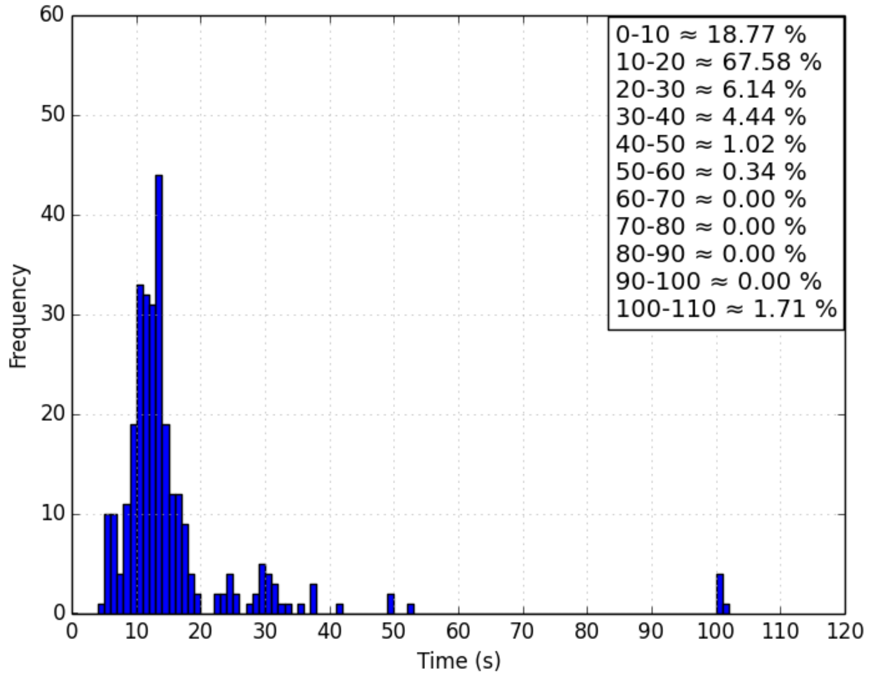


Figure 4.8: SRC processing time for starting video processors.

model, and network topology. Most URC processing times for starting a video processor are in the range of 10-15 seconds (Figure 4.5). Measuring the processing time for a URC to stop a video processor does not involve the binary loading and network traffic when measuring the command waiting time. Most URC waiting times for starting a video processor are in the range of 0-10 seconds (Figure 4.3) because results in long waiting times for acquiring a new compute node worker do not often occur. Most waiting and processing times for the URCs to stop a video processor are in the ranges of 0-10 and 1-2 seconds respectively (Figures 4.4 and 4.6).

Figures 4.7 and 4.8 show the waiting and processing times of SRCs for video processor recovery. The waiting and processing times of SRCs are similar to the waiting and processing times of URCs. However, URC events occur spontaneously while SRC events occur automatically. For example, when a video processor exits with an error because it cannot acquire an image from the video connection, then the compute node worker will detect the suspicious consumption of computing resources. Most waiting and processing times for SRCs to start a video processor have similar characteristics, falling within the range of 10-20 seconds. The response times show that the system works well.

4.2.3 Discussion

In this section, Nokkhum is compared with available video surveillance systems as shown in Tables 4.7 and 4.8. However, direct comparisons are difficult because some of these systems are conceptual designs without complete implementations. Some researches focused on resource allocation for system scalability rather than software architecture. Also, some researches used simulated data rather than the actual one.

As a consequence, some descriptions are unavailable (N/A) or lacking in information (L/I). For example, Nokkhum provides Account/Billing, Authentication, and Authorization (AAA), while the information is unclear for other systems. Nokkhum can more flexibly configure components for different deployment scenarios. For instance, it can utilize resources in a minimal way at first, and then scale them up later. In this way, Nokkhum is not just a particular type of video surveillance system applying cloud computing technology, but it also supports a variety of users in the public cloud, flexibility configurations for different system sizes, and scalability, which are topics that are not addressed in other systems.

Section 4.2.2 includes histograms of VSaaS response times for service delivery to users. Other CVS systems did not provide such response time information. Instead, they focused on video processing performance or network bandwidth usage. Nokkhum system response times are similar to those for the bundled software which come with the IP cameras.

Table 4.7: Compairison of cloud-based video surveillance systems.

System	*Nokkhum*	*P2PCloud*	*VAQACI*
Architecture	Distributed components	Distributed physical nodes	Distributed components
AAA service	Account and Billing, Authentication, Authorization	N/A	Billing
Cloud technology	Nova/EC2, Swift/S3	HDFS	MapReduce, HDFS
Real-time video analysis	Yes	L/I	Only after videos have been recorded and pushed into the HDFS
Bandwidth usage	According to configuration	Low (only when event detectors are used, otherwise as high as others)	High (centralized video controller by Hadoop)

Table 4.8: System scalability compairison.

System	*Nokkhum*	*P2PCloud*	*VAQACI*
Resource allocation	Automatic	N/A	Automatic
Deployment configuration	Supporting many scenarios according to Figures 3.12 - 3.18	Local node and directory node	One scenario
Minimum number of deployed machines	1	Minimum required 2 (central and local)	2

However, many factors can affect the URC processing times for starting a video processor. These include the camera response time for starting video streaming, resource allocation waiting time, and the URC scheduling waiting time. Nokkhum supports real-time video processing while other systems do not fully support it. Nokkhum can vary its network bandwidth usage according to its users' configurations while other systems are not as flexible.

Also, this thesis has tested Nokkhum controller's capability for handling the compute node worker. In the test scenario, the compute node worker has spawned 25 VMs on the OpenStack environment. The VMs include one to two cores of CPU and 512 MB to 2.10 GB of RAM. The controller can comprehensively deal with the all VMs and has a tendency to support more compute node workers. This test scenario uses VMs with 4-core CPU, 8 GB of RAM, and 80 GB of hard disk according to Figure 3.15 configuration.

4.3 Video Task Scheduling Results

This section presents experimental results when testing the Nokkhum scheduler with different resource configurations and machine specifications. The results then become the video processing exploration data for applying scaling factors in later scheduling.

1. Computing node employs physical machine as shown in Table 4.2 and virtual machine as shown in Table 4.3.
   (a) Physical machines: AMD-2.8-PM, AMD-3.5-PM, AMD-3.8-PM, Intel-2.6-PM, Intel-2.8-PM, and, Intel-3.4-PM.
   (b) VMs: AMD-2.8-VM, AMD-3.5-VM, AMD-3.8-VM, Intel-2.6-VM, Intel-2.8-VM, and, Intel-3.4-VM.
2. Using VIRAT video data set as a reference video.
3. Exploring CPU and RAM based on video formats as follows:
   (a) Eight frame rates: 1, 5, 7, 10, 15, 20, 25, and 30 FPS.
   (b) Six frame sizes: 160x120, 320x240, 640x480, 800x600, 960x720, and 1120x840 pixels.
   (c) Ogg codec is applied for video output.
4. Exploring video processing including motion detection and video recording, and motion recording which is a combination of motion detection and video recording.
5. Each exploration results shown in Table 4.9 have been tested ten times.

4.3.1 Video Task Scheduling Experimental Results

Table 4.9 presents the resource usage for the machines described in Tables 4.2 and 4.3, comparing three resource usages criteria: the average (Avg), average maximum (AMax), and average minimum (AMin) of CPU and memory usage. Each run involves two video processing tasks, a video recorder (VR) and motion detector (MD). These results are used as example models for determining task scheduling.

Figures 4.9 and 4.10 present task scheduling results based on a physical machine (AMD-3.5-PM) and a VM (AMD-3.5-VM), involving a number of both types of video tasks

Table 4.9: Video processing workload resource usage results.

Code	Processor	Video		CPU Usage (%)			Memory Usage (MB)		
		Size	FPS	Avg	AMax	AMin	Avg	AMax	AMin
AMD-3.5-PM	MD	320x240	10	9.80	10.47	8.83	46.08	46.82	45.01
		320x240	15	12.44	13.47	11.36	46.16	46.98	44.10
		320x240	30	20.24	21.40	19.26	47.03	47.74	46.19
		640x480	10	26.51	28.45	25.08	71.70	79.00	69.56
		640x480	15	36.40	38.15	34.35	71.90	80.58	68.70
		640x480	30	75.86	123.01	68.24	82.30	93.32	76.19
	VR	320x240	10	11.80	12.67	10.71	41.18	41.69	40.70
		320x240	15	16.50	17.46	15.55	41.51	41.84	40.86
		320x240	30	29.26	30.36	28.42	42.01	42.17	41.58
		640x480	10	29.35	30.86	28.20	54.61	55.36	53.99
		640x480	15	39.57	40.90	37.95	54.98	55.50	54.27
		640x480	30	75.49	77.89	73.28	55.88	56.26	55.11
AMD-3.8-PM	MD	640x480	10	25.45	27.60	24.07	67.26	79.94	64.43
	VR	640x480	10	28.42	29.86	26.95	49.40	49.85	48.63
Intel-2.8-PM	MD	640x480	10	25.22	28.40	23.83	65.65	77.56	62.84
	VR	640x480	10	31.55	32.75	30.45	48.57	48.97	47.79
Intel-3.8-PM	MD	640x480	10	26.93	29.32	25.28	66.81	78.23	64.40
	VR	640x480	10	40.28	41.61	39.19	49.84	50.24	49.20
AMD-3.5-VM	MD	320x240	10	9.97	10.68	8.49	40.14	40.42	37.18
		320x240	15	13.13	14.30	12.58	39.94	40.45	35.96
		320x240	30	22.10	23.28	21.47	40.86	41.63	40.21
		640x480	10	25.24	27.60	23.99	61.79	70.98	59.86
		640x480	15	33.85	35.04	31.80	62.72	73.89	59.46
		640x480	30	65.27	66.75	64.17	66.17	75.79	59.52
	VR	320x240	10	13.37	14.36	12.62	35.92	36.11	35.76
		320x240	15	17.72	18.55	16.56	35.70	35.82	35.57
		320x240	30	29.83	30.67	28.59	36.12	36.24	35.99
		640x480	10	31.27	32.72	30.11	45.35	46.11	44.93
		640x480	15	41.20	42.79	39.93	45.64	46.36	45.26
		640x480	30	78.08	80.20	76.20	45.69	46.49	45.32
AMD-3.8-VM	MD	640x480	10	25.29	27.40	24.16	61.81	71.87	59.82
	VR	640x480	10	29.19	30.88	28.18	45.37	46.02	44.94
Intel-2.8-VM	MD	640x480	10	36.98	39.43	35.09	63.10	73.28	59.35
	VR	640x480	10	32.92	34.27	31.59	45.81	46.56	45.36
Intel-3.8-VM	MD	640x480	10	26.38	29.18	25.00	63.12	75.64	60.48
	VR	640x480	10	39.94	41.58	38.88	45.52	46.25	45.09

MD: Motion detector
VR: Video recorder
Avg: Average
AMax: Average Maximum
Amin: Average Minimum

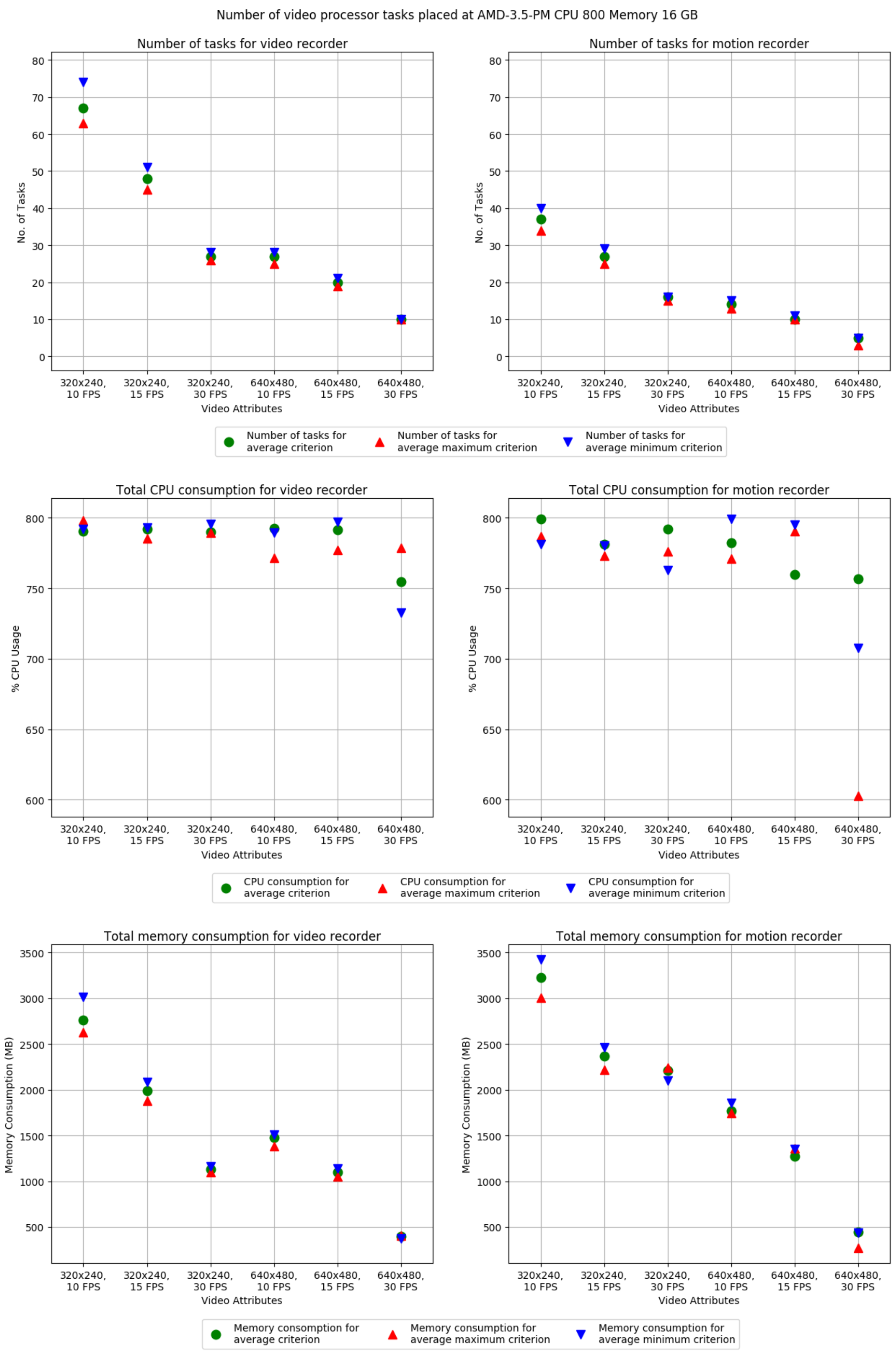


Figure 4.9: Number of tasks, the CPU and memory consumption in the physical machine (AMD-3.5-PM).

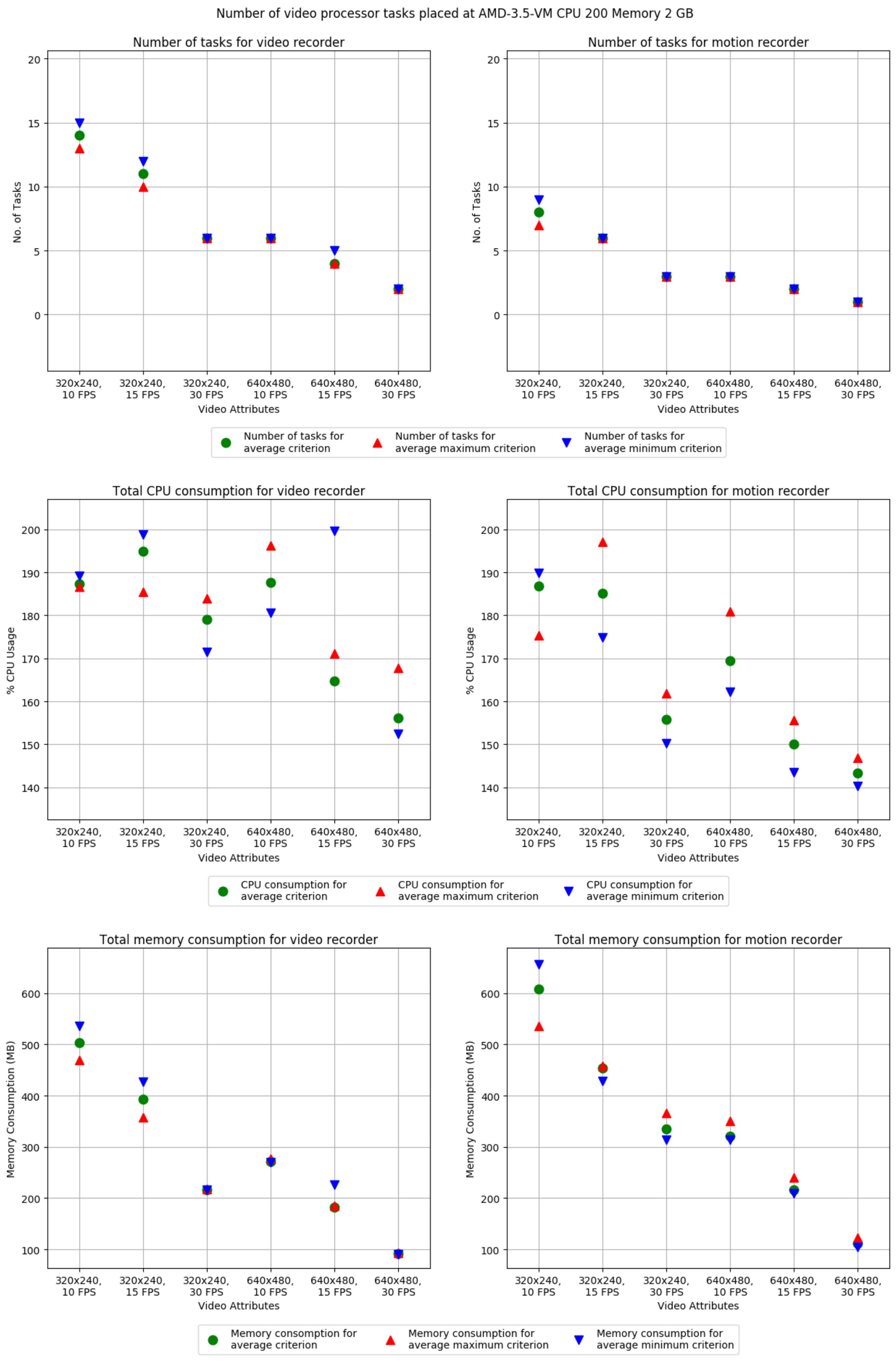


Figure 4.10: Number of tasks, the CPU and memory consumption in the virtual machine (AMD-3.5-VM).

(VR and motion recorder (MR)). AMD-3.5-VM is a VM running on AMD-3.5-PM. Both are very similar, but their CPU specifications are different as shown in Tables 4.2 and 4.3. The important specification parameters are the number of cores and the maximum memory. The video processing tasks used for scheduling are a VR and a MR, whereas MR combines motion detector and video recorder tasks. This means that the MR computing resource usage is a summation of both tasks.

Scheduling results are shown in Figures 4.9 and 4.10. One of the experiments of which the result shown in Figure 4.10 involves a video recorder, using the frame size of 640x480 pixels, and a frame rate of 10 FPS on AMD-3.5-VM. The scheduler can assign six video recorder tasks to this VM when applying the average CPU usage criterion. The video recorder task occupies the CPU usage for 187.59 % and consumes 272.10 MB, where the maximum memory usage is 2000MB and the maximum CPU usage is 200 %. Therefore, there is some space left for placing another task on the VM. For instance, the scheduler cloud can add a video recorder VM, with the video size of 320x240 pixels and frame rate of 10 FPS which would consume 199.39 % CPU usage and 313.28 MB memory. The machine specification can support a these workload, and so they all can run together smoothly. As the video format that recommended by SWGIT is 640x480 pixels at 30 FPS. The scheduler can place ten video recorder on the AMD-3.5-PM and two video recorder on the AMD-3.5-VM.

Figures 4.9 and 4.10 show that the number of tasks placed on a compute node worker depends on the frame rate and size, for both the video and motion recorders. The number of tasks decreases when the frame rate and size increase. This characteristic appears in both physical and virtual machines, and also directly affects memory consumption. Therefore, this thesis can consider the number of tasks in the analysis as reflecting the memory usage. However, the CPU consumption by the video and motion recorders affects the CPU usage percentage, depending on the number of tasks and the compute node capacity. This indicates that CPU consumption is the most significant factor for workload scheduling.

According to the scheduling process, the video processing scheduler's response time in different scenarios are shown in Table 4.10. When the scheduler executed without any experiment dataset in the database, it took the average response time of 0.0427 second, while the maximum response time and minimum response time were 0.2393 and 0.0005 seconds respectively. While the scheduler found the experimental result matched with the desired CPU model, it took longer time than when there were no experimental results in the database. The average, maximum, and minimum response time were 0.0715, 0.9315, and 0.0002 seconds respectively. Part of the execution time was the database querying time. In case that the scheduler could not match the experimental results with the desired CPU model, the scheduler took the average response time of 0.2637 seconds. The average maximum and minimum response times were 1.2503 and 0.0005 second which were more than the querying time from the database. The experimental results show that the scheduling consumed the average scheduling time less than one second. All experimental results have been tested on AMD-3.5-VM specification, and seven physical machines and seven virtual machines including 1344 experimental results (96 test cases on 17 machines).

Table 4.10: Scheduling response time.

Scienario	Response Time (s)			
	Average	Maximum	Minimum	Standard Deviation
No dataset	0.0427	0.2393	0.0005	0.0473
CPU model found	0.0715	0.9315	0.0005	0.1113
CPU model Not found	0.2637	1.2503	0.0005	0.2926

4.3.2 Discussion

The Nokkhum scheduler assigns a video processing task to a compute node by examining adjacent experiment results from the exploration test in order to estimate the resource usage. However, if the scheduler does not have enough resource usage information, it may assign tasks that exceed the compute node's capacity. To avoid this, the Nokkhum scheduler also need to compare the real resource usage and the estimated resource usage. This method is used to prevent system failure until better exploration results are received.

This thesis can estimate the number of video processing tasks per compute node by transforming Equations (3.4.1), (3.4.2) and (3.4.3) into Equation (4.1). It shows that the number of video processing tasks depends on the video frame rate and frame size per compute node. Also, the CPU information is given more weight than the memory. The VSaaS administrator can predict the number of compute nodes for their system by applying Equation 4.1, and also plan the system deployment budget.

$$N_{CPU} = \lfloor \frac{C.cpu_{core} \times 100}{CPU_{estimated}} \rfloor$$

$$N_{memory} = \lfloor \frac{C.total\_memory}{memory_{estimated}} \rfloor$$

$$N_{total} = \begin{cases} N_{CPU} & \text{if } N_{CPU} < N_{memory}, \\ N_{memory} & \text{otherwise} \end{cases} \tag{4.1}$$

where N_{CPU} is the possible number of video processing tasks divided by CPU usage;

N_{memory} is the maximum number of video processing tasks divided by memory usage;

N_{total} is the maximum number of video processing tasks based on a comparison of N_{CPU} and N_{memory}.

4.4 Summary

This chapter addresses that the Nokkhum VSaaS can run on virtual and physical machines based on the OpenStack software for providing the cloud environment. Also, this chapter concludes the checklist for verifying the system scalability and flexibility designed in Section 3.2.3. In addition, the chapter presents the response time for any request command type for testing the VSaaS service work as well as the response quality to the request commands. Moreover, this chapter compares the Nokkhum VSaaS with other available systems in several factors. Finally, the scheduling example results for placing a video processing task in a suitable computing node running a real video processing workload have been presented.

CHAPTER 5

CONCLUSIONS

This chapter presents the thesis contributions and suggestions for further developments of Nokkum VSaaS. It reveals both investigation results on Nokkhum VSaaS architecture and scheduling. The contribution concerns architectural design, development, optimization and testing schemes.

5.1 Thesis Summary

This thesis involves the design and development of a flexible and scalable component-based VSaaS architecture for providing a VSaaS system called Nokkhum. It enables deploying any components separately on both physical servers and VMs IaaS. The Nokkhum VSaaS software architecture can automatically scale the number of VMs to support dynamic user requirements. The Nokkhum applies the Amazon EC2 API for automatically handling VM acquisition, and the Amazon S3 API is used by the storage engine. The designs and benefits of Nokkhum components are described as follows.

The Nokkhum controller component consists of many modules for managing the computing resources, video processing tasks, notifier, and billing. The controller can handle a large number of compute node workers according to the computing unit performance. In a single VSaaS system, the Nokkhum system allows more than one active controller for supporting a large number of compute node workers and concurrently dealing with new video processing tasks. This scheme increases the system availability and relieves the system overloading. Also, the controller includes VM management interface using Amazon EC2. It automatically acquires an additional computing resource, when the compute node controller detects insufficient computing resource for a new video processing task. The compute node controller activates this action when the cloud provider key has been assigned. All above descriptions are made up for the feature functions of the Nokkhum controller.

This thesis shows the design of the Nokkhum compute node worker for reporting computing status, both of the computing unit and video processing task, to the controller, and responding to the controller order. The compute node worker is separately designed from Nokkhum processors so that the compute node workers can support related stream processing by implementing standard input/output interfaces. This designed model is different from other systems. The others only support their video processing software. The Nokkhum compute node worker consists of the data storage interface for storing all data into the cloud object storage using Amazon S3 API. It immediately pushes the output data to the cloud object storage as soon as possible. This data pushing style causes the computing unit does not require a high volume hard disk. However, the hard drive consumption may relate to the CPU capacity for temporarily storing video records.

The Nokkum controller and compute node worker cooperate with each other to manage video processing tasks. They communicate with each other via a message broker using the publish-subscribe messaging pattern. Also, the publish-subscribe model increases fault tolerance for the Nokkhum system. Although, some components terminate when errors occur, the other components still smoothly continue running. When the crashed component restarts working again, the controller will recover the system states and continuously process next operations. Moreover, the publish-subscribe pattern extends the scalability of the system architecture. The Nokkhum VSaaS system can, as a minimum, deploy only one machine for a small-size VSS requirement. For the requirement of a larger VSS, the Nokkhum system can distribute components especially the Nokkhum compute node worker, to another server or more. The scaling component operation can proceed while the system is running, without a need to shut the system down. This potential is explicitly addressed in the mechanism and the design of the Nokkhum architecture but it was implicit and not mentioned in many previous works.

The REST interface of the Nokkhum API server allows the system to support multiple types of clients. In this thesis, a web client working as a proof of concept has been implemented and used in both desktop and mobile platforms. The API server provides a set of functions for managing cameras, video processors, and video records. It applies a token based mechanism for authentication and authorization of the user requests. Then, it composes a controlling message for managing the user's video processing task. The API server conceals the users from the controller in order to protect the controller from direct security attacks. In addition, the API server can concurrently be deployed in many servers.

According to the Nokkhum scalability, this thesis presents the seven possible components deployment topologies as a guideline to the VSaaS administrator. All topologies are aimed for configuring the components corresponding to various provider requirements. The Nokkhum architecture can be deployed using a minimal number of servers at the beginning and can later be scaled up quickly for supporting a growing number of cameras. The minimum number of servers begins with one and the number of Nokkhum compute node workers and processors can be scaled up to sufficiently support added IP cameras. In case that there is a large number of IP cameras, all Nokkhum components can be distributed to many servers in order to help respond to video processing demands.

This thesis has described video processing workload analysis and resource estimation for workload scheduling on the Nokkhum VSaaS, which handles the computing resource consumption of video processing tasks by interlacing the percentages of CPU and memory usage with frame rates and sizes. In this part, the characteristics of video processing workloads based on popular home-use video processing, especially motion detection, and video recorder, have been studied. Based on experiments run on both physical and virtual machines, this thesis has developed video processing that stably consumes computing resources. A typical combination of video processors incurs different characteristics which makes it difficult to model CPU and memory usage. Therefore, the approach for video processing task scheduling utilizes task exploration which collects and records CPU and memory usage in a database. Later, the scheduler uses this data to estimate required computing resources for a new task, to determine which compute node is the most suitable for it. Task exploration is initiated when there is no video processing information for a computing unit in the database, and the unit is idle.

This thesis has also described a method for determining video resource usage when there is no exactly matching information in the database. While waiting for exploration results, the scheduler estimates the new video processing task's resource usage from existing experimental results available in the database. The estimation process employs a scaling factor to adjust the CPU usage as a heuristic in the scheduling process. This resource estimation process enables the scheduler to immediately place a task on an appropriate compute node without waiting for exploration results which may take a long time to be produced.

In addition, this thesis has offered criteria (the average, average maximum, and average minimum resource usage) as guidelines for identifying the resource usage policy. These three criteria allow the administrator to adjust the requested task to the desired compute node. Average resource usage fits all task consumption relative to the computing unit. Average maximum resource usage can force the compute node towards an excessive workload and average minimum resource usage can reduce consumption to allow space to left over.

In conclusion, this thesis proposes the design and implementation of the architecture for a VSaaS system. It enables flexible deployment on both physical machines and VMs. The system scaling process does not require the system to be shut down, unlike in other systems. This scaling operation is clearly defined in this system, while it was not emphasized in other works. Nokkhum VSaaS is flexible enough to tolerate challenging scenarios in which some components are not connected due to various causes such as electricity blackout, loss of network communication, or suspension due to software errors which require a restart. Not only the architecture design, but also the resource usage efficiency is important. This thesis studies the video processing workloads characteristics, especially motion detection, and video recorder with considerable factors, such as the type of video processing, video frame rate, frame size, and computing unit specification. Consequently, this thesis proposes video processing exploration based scheduling for placing any video processing task to a suitable computing node. The scheduler applies the resource usage data from the video processing task exploration to predict a new video processing task consumption. Moreover, this thesis describes a resource decision mechanism for approximating the resource usage of the video processing task when the exploration data is not available. The scheduler also applies the proposed three resource usage criteria to be identified by an administrator for resource usage provision in the scheduling process.

5.2 Claims to Originality

This thesis contributes to the designs, developments and applications of VSaaS, especially the flexible and scalable architecture and video processing task scheduling in a number of ways. The system architecture has been suggested and new approaches have been proposed for organizing and managing the VSaaS. The original contributions are highlighted as below.

- A flexible and scalable component-based VSaaS architecture possibly deployed on both heterogeneous physical servers and VMs of IaaS has been proposed.
- Encapsulating the functionalities of each components, and separating the VM layer to the VSS layer by using components has been presented.
- The idea of employing more than one active controller helps increase the system scalability and relieve the system overloading.
- The concept of having the compute node worker separately designed and work apart from the video processor enables that it can support related stream processing implemented by standard I/O interfaces.
- Storing data into the cloud object storage with an immediate pushing style consumes a lower volume of hard disk of the computing node.

- The publish-subscribe message passing mechanism has been used for the cooperation between the controller and compute node worker, and it contributes to the system fault tolerance and scalability.
- The API server applying the REST interface and a token based mechanism has been designed to support multiple types of clients as well as to protect the controller from direct security attacks.
- A video processing workload analysis has been carried out, and it reveals factors affecting the system resources consumption.
- The video processing task exploration approach has been proposed and conducted to collect the CPU and memory usages into the database to enable the scheduler to compare and estimate the resources for a new video processing task.
- The method and algorithm for estimating the resource usage of a new video processing task have been suggested, employing both data from the video processing task exploitation and CPU scaling factor.
- Three resource usage criteria have been proposed and guided for resource scheduling and prediction.

5.3 Limitations

Currently, the Nokkhum system has been deployed on a physical machine via manual configuration, applying a snapshot of the VM image in the OpenStack environment. It also supports cloud resource management APIs, Amazon EC2 and S3. Then, it can be deployed by the cloud provider which provides those APIs. According to a single machine deployment, the minimum machine specification required is 2.0 GHz of CPU, 4 GB of RAM, 500 GB of hard disk, which can cover 4-6 cameras. In addition, this thesis studies video processing workload characteristics based on only video recorder, motion detection, and motion recorder. Other excluding complex video processing tasks have been excluded in the study.

5.4 Future Work

There still are many issues for further research and development in order to make a better VSaaS, for example:

- A video processing redundant system to archive high availability and fault tolerance
- Billing according to on-demand video processing resources
- Migration of VSaaS computation to other locations for more system availability
- Selection of best VSaaS locations for video processing with low video stream transfer latency and network bandwidth
- All the Nokkhum components can be configured to run on modern lightweight software containers like Docker [65] for reducing time to start and enable automatic deployment.
- Extended types of video processing workloads analysis can be included.

BIBLIOGRAPHY

[1] M. Valera and S. Velastin, “Intelligent distributed surveillance systems: a review,” in Proceedings of the IEE Vision, Image and Signal Processing, vol. 152, no. 2, 2005, pp. 192–204.

[2] B. Georis, X. Desurmont, D. Demaret, S. Redureau, J. Delaigle, and B. Macq, “IP-distributed computer-aided video-surveillance system,” in Proceedings of the IEE Symposium on Intelligence Distributed Surveillance Systems, 2003, pp. 18/1–18/5.

[3] NW Systems Group Limited. (2013) SecurityStation - how VSaaS works and the benefits of VSaaS. http://www.securitystation.com/how-vsaas-works.php. [Online]. Available: http://www.securitystation.com/how-vsaas-works.php

[4] Secure-i. (2013) VCR → DVR → NVR ... and now HVR™. http://www.secure-i.com/learn/technologies. [Online]. Available: http://www.secure-i.com/learn/technologies

[5] Neo IT Solutions Inc. (2013, Feb.) OVS™ online video surveillance as a service | VSaaS | MVaaS | RVMaS | VAS. http://www.neovsp.com/solutions. [Online]. Available: http://www.neovsp.com/solutions

[6] Amazon. (2014) AWS documentation. [Online]. Available: http://aws.amazon.com/documentation/

[7] T. White, Hadoop: The Definitive Guide, 1st ed. O’Reilly Media, Inc., 2009.

[8] Y.-S. Wu, Y.-S. Chang, T.-Y. Juang, and J.-S. Yen, “An architecture for video surveillance service based on P2P and cloud computing,” in Proceedings of the 2012 9th International Conference on Ubiquitous Intelligence Computing and 9th International Conference on Autonomic Trusted Computing (UIC/ATC), Sep. 2012, pp. 661–666.

[9] J. Lee, T. Feng, W. Shi, A. Bedagkar-Gala, S. Shah, and H. Yoshida, “Towards quality aware collaborative video analytic cloud,” in 2012 IEEE 5th International Conference on Cloud Computing (CLOUD), 2012, pp. 147–154.

[10] C.-F. Lin, S.-M. Yuan, M.-C. Leu, and C.-T. Tsai, “A framework for scalable cloud video recorder system in surveillance environment,” in Proceedings of the 2012 9th International Conference on Ubiquitous Intelligence Computing and 9th International Conference on Autonomic Trusted Computing (UIC/ATC), 2012, pp. 655–660.

[11] M. Hossain, M. Hassan, M. Qurishi, and A. Alghamdi, “Resource allocation for service composition in cloud-based video surveillance platform,” in Proceedings of the 2012 IEEE International Conference on Multimedia and Expo Workshops (ICMEW), 2012, pp. 408–412.

[12] M. A. Hossain and M. A. Hossain, “Framework for a cloud-based multimedia surveillance system, framework for a cloud-based multimedia surveillance system,” International Journal of Distributed Sensor Networks, International Journal of Distributed Sensor Networks, vol. 2014, p. e135257, 2014. [Online]. Available: http://www.hindawi.com/journals/ijdsn/2014/135257/abs/,http://www.hindawi.com/journals/ijdsn/2014/135257/abs/

[13] D. Rodriguez-Silva, L. Adkinson-Orellana, F. Gonz’lez-Castano, I. Armino-Franco, and D. Gonz’lez-Martinez, “Video surveillance based on cloud storage,” in Proceedings of the 2012 IEEE 5th International Conference on Cloud Computing (CLOUD), 2012, pp. 991–992.

[14] X. Nan, Y. He, and L. Guan, “Optimal resource allocation for multimedia cloud based on queuing model,” in 2011 IEEE 13th International Workshop on Multimedia Signal Processing (MMSP), 2011, pp. 1–6.

[15] D. Miao, W. Zhu, C. Luo, and C. W. Chen, “Resource allocation for cloud-based free viewpoint video rendering for mobile phones,” in Proceedings of the 19th ACM International Conference on Multimedia, ser. MM ’11. ACM, 2011, pp. 1237–1240. [Online]. Available: http://doi.acm.org/10.1145/2072298.2071983

[16] B. Abreu, L. Botelho, A. Cavallaro, D. Douxchamps, T. Ebrahimi, P. Figueiredo, B. Macq, B. Mory, L. Nunes, J. Orri, M. Trigueiros, and A. Violante, “Video-based multi-agent traffic surveillance system,” in Proceedings of the IEEE Intelligent Vehicles Symposium, 2000. IV 2000., 2000, pp. 457–462.

[17] C. Jaynes, S. Webb, R. M. Steele, and Q. Xiong, “An open development environment for evaluation of video surveillance systems,” Proceeding of the Third International Workshop on Performance Evaluation of Tracking and Surveillance, PETS’ 2002, vol. 1, pp. 32–39, 2002. [Online]. Available: http://citeseerx.ist.psu.edu/viewdoc/summary?doi=10.1.1.3.6890

[18] J. San Miguel, J. Bescos, J. Martinez, and A. Garcia, “DiVA: a distributed video analysis framework applied to video-surveillance systems,” in Proceedings of the Ninth International Workshop on Image Analysis for Multimedia Interactive Services, 2008. WIAMIS ’08., 2008, pp. 207–210.

[19] N. Suvonvorn, “A video analysis framework for surveillance system,” in Proceedings of the 2008 IEEE 10th Workshop on Multimedia Signal Processing, 2008, pp. 867–871.

[20] Triornis Ltd. (2013, Jan.) ZoneMinder - main documentation. http://www.zoneminder.com/wiki/index.php/Documentation. [Online]. Available: http://www.zoneminder.com/wiki/index.php/Documentation

[21] X. Yuan, Z. Sun, Y. Varol, and G. Bebis, “A distributed visual surveillance system,” in Proceedings of the IEEE Conference on Advanced Video and Signal Based Surveillance, 2003., Jul. 2003, pp. 199–204.

[22] R. G. J. Wijnhoven, E. G. T. Jaspers, and P. H. N. de With, “Flexible surveillance system architecture for prototyping video content analysis algorithms,” in Proceedings of the SPIE, vol. 6073, no. 1, Jan. 2006, pp. 60 730R–60 730R–9.

[23] J. Han, N. Choi, T. Chung, T. Kwon, and Y. Choi, “A target-centric surveillance system based on localization and social networking,” Multimedia Tools and Applications, pp. 1–25, 2012. [Online]. Available: http://dx.doi.org/10.1007/s11042-012-1285-8

[24] A. C. Nazare Jr. and W. R. Schwartz, “A scalable and flexible framework for smart video surveillance,” Computer Vision and Image Understanding, vol. 144, pp. 258–275, 2016. [Online]. Available: http://www.sciencedirect.com/science/article/pii/S1077314215002349

[25] R. Vezzani and R. Cucchiara, “Video surveillance online repository (ViSOR): an integrated framework,” Multimedia Tools and Applications, vol. 50, no. 2, pp. 359–380, 2010. [Online]. Available: http://link.springer.com/article/10.1007/s11042-009-0402-9

[26] G. Gualdi, A. Prati, and R. Cucchiara, “Video streaming for mobile video surveillance,” IEEE Transactions on Multimedia, vol. 10, no. 6, pp. 1142–1154, Oct. 2008.

[27] Y. Huang, “The design and implementation on a new generation of remote network video surveillance system,” in Proceedings of the 2010 3rd International Conference on Advanced Computer Theory and Engineering (ICACTE), vol. 2, Aug. 2010, pp. V2–294–V2–297.

[28] A. Karimaa, “Video surveillance in the cloud: Dependability analysis,” in Proceedings of the Fourth International Conference on Dependability, DEPEND 2011., 2011, pp. 92–95.

[29] Y. L. Chen, T. S. Chen, L. C. Yin, T. W. Huang, S. Y. Wang, and T. C. Chieuh, “City eyes: An unified computational framework for intelligent video surveillance in cloud environment,”

in 2014 IEEE International Conference on Internet of Things (iThings), and IEEE Green Computing and Communications (GreenCom) and IEEE Cyber, Physical and Social Computing (CPSCom), 2014, pp. 324–327.

[30] M. A. Hossain and B. Song, "Efficient resource management for cloud-enabled video surveillance over next generation network," Mobile Networks and Applications, pp. 1–16, 2016. [Online]. Available: http://link.springer.com/article/10.1007/s11036-016-0699-3

[31] X. Yang, H. Zhang, H. Ma, W. Li, G. Fu, and Y. Tang, "Multi-resource allocation for virtual machine placement in video surveillance cloud," in Human Centered Computing, ser. Lecture Notes in Computer Science, Q. Zu and B. Hu, Eds. Springer International Publishing, 2016, no. 9567, pp. 544–555, DOI: 10.1007/978-3-319-31854-7_49. [Online]. Available: http://link.springer.com/chapter/10.1007/978-3-319-31854-7_49

[32] A. Alamri, M. S. Hossain, A. Almogren, M. M. Hassan, K. Alnafjan, M. Zakariah, L. Seyam, and A. Alghamdi, "QoS-adaptive service configuration framework for cloud-assisted video surveillance systems," Multimedia Tools and Applications, pp. 1–16, 2015. [Online]. Available: http://link.springer.com/article/10.1007/s11042-015-3074-7

[33] M. S. Hossain, "QoS-aware service composition for distributed video surveillance," Multimedia Tools and Applications, vol. 73, no. 1, pp. 169–188, 2013. [Online]. Available: http://link.springer.com/article/10.1007/s11042-012-1312-9

[34] B. Song, Y. Tian, and B. Zhou, "Design and evaluation of remote video surveillance system on private cloud," in 2014 International Symposium on Biometrics and Security Technologies (ISBAST), 2014, pp. 256–262.

[35] C. M. Sharma and H. Kumar, "Architectural framework for implementing visual surveillance as a service," in 2014 International Conference on Computing for Sustainable Global Development (INDIACom), 2014, pp. 296–301.

[36] T. S. Chen, M. F. Lin, T. c. Chieuh, C. H. Chang, and W. H. Tai, "An intelligent surveillance video analysis service in cloud environment," in 2015 International Carnahan Conference on Security Technology (ICCST), 2015, pp. 1–6.

[37] R. Ananthanarayanan, K. Gupta, P. Pandey, H. Pucha, P. Sarkar, M. Shah, and R. Tewari, "Cloud analytics: do we really need to reinvent the storage stack?" Proceedings of the 2009 conference on Hot topics in cloud computing, pp. 15–15, 2009, ACM ID: 1855548. [Online]. Available: http://portal.acm.org/citation.cfm?id=1855533.1855548

[38] R. Buyya, C. S. Yeo, S. Venugopal, J. Broberg, and I. Brandic, "Cloud computing and emerging IT platforms: Vision, hype, and reality for delivering computing as the 5th utility," Future Gener. Comput. Syst., vol. 25, pp. 599–616, 2009, ACM ID: 1529211. [Online]. Available: http://portal.acm.org/citation.cfm?id=1528937.1529211

[39] B. Rimal, E. Choi, and I. Lumb, "A taxonomy and survey of cloud computing systems," in INC, IMS and IDC, 2009. NCM '09. Fifth International Joint Conference on, 2009, pp. 44–51.

[40] D. Neal and S. M. Rahman, "Video surveillance in the cloud-computing?" in 2012 7th International Conference on Electrical Computer Engineering (ICECE), 2012, pp. 58–61.

[41] D. J. Neal and S. Rahman, "Video surveillance in the cloud?" International Journal on Cryptography and Information Security, vol. 2, no. 3, pp. 1–19, 2012. [Online]. Available: http://arxiv.org/abs/1512.00070

[42] A. Kivity, Y. Kamay, D. Laor, U. Lublin, and A. Liguori, "kvm: the linux virtual machine monitor," in Proceedings of the Linux Symposium, vol. 1, 2007, pp. 225–230.

[43] D. L. Baggio, S. Emami, D. M. Escrivá, K. Ievgen, N. Mahmood, J. Saragih, and R. Shilkrot, Mastering OpenCV with Practical Computer Vision Projects. Packt Publishing, Dec. 2012.

[44] T. S. W. G. on Imaging Technology (SWGIT). (2016) Recommendations and guidelines for using closed-circuit television security systems in commercial institutions. [Online]. Available: https://www.swgit.org/pdf/Section%204%20Recommendations%20and%20Guidelines%20for%20Using%20Closed-Circuit%20Television%20Security%20Systems%20in%20Commercial%20Institutions?docID=48

[45] Xiph.Org. (2016) Xiph.org: Ogg. [Online]. Available: https://xiph.org/ogg/

[46] S. Oh, A. Hoogs, A. Perera, N. Cuntoor, C.-C. Chen, J. T. Lee, S. Mukherjee, J. K. Aggarwal, H. Lee, L. Davis, E. Swears, X. Wang, Q. Ji, K. Reddy, M. Shah, C. Vondrick, H. Pirsiavash, D. Ramanan, J. Yuen, A. Torralba, B. Song, A. Fong, A. Roy-Chowdhury, and M. Desai, "A large-scale benchmark dataset for event recognition in surveillance video," in Proceedings of the 2011 IEEE Conference on Computer Vision and Pattern Recognition, ser. CVPR '11. IEEE Computer Society, 2011, pp. 3153–3160. [Online]. Available: http://dx.doi.org/10.1109/CVPR.2011.5995586

[47] Q. Mahmoud, Middleware for Communications. John Wiley & Sons, Jun. 2005.

[48] S. Vinoski, "Advanced message queuing protocol," IEEE Internet Computing, vol. 10, no. 6, pp. 87–89, 2006.

[49] R. T. Fielding and R. N. Taylor, "Principled design of the modern web architecture," ACM Transaction on Internet Technol., vol. 2, no. 2, pp. 115–150, May 2002. [Online]. Available: http://doi.acm.org/10.1145/514183.514185

[50] E. Wan and R. Van der Merwe, "The unscented kalman filter for nonlinear estimation," in The IEEE 2000 Adaptive Systems for Signal Processing, Communications, and Control Symposium 2000. AS-SPCC., 2000, pp. 153–158.

[51] Python Software Foundation. (2014, Mar.) Subprocess management - python documentation. [Online]. Available: http://docs.python.org/3.4/library/subprocess.html

[52] D. Crockford, "Json: The fat-free alternative to xml," in Proceeding of XML, vol. 2006, 2006.

[53] L. Masinter, T. Berners-Lee, and R. T. Fielding. (2013) Uniform resource identifier (URI): generic syntax. [Online]. Available: http://tools.ietf.org/html/rfc3986

[54] C. McDonough. (2016, Feb.) The pyramid web application development framework. [Online]. Available: http://docs.pylonsproject.org/projects/pyramid/en/latest/

[55] K. Chodorow and M. Dirolf, MongoDB: The Definitive Guide. O'Reilly Media, Inc., Sep. 2010.

[56] E. Rescorla. (2013) HTTP over TLS. [Online]. Available: https://tools.ietf.org/html/rfc2818

[57] D. Nurmi, R. Wolski, C. Grzegorczyk, G. Obertelli, S. Soman, L. Youseff, and D. Zagorodnov, "The eucalyptus open-source cloud-computing system," in 9th IEEE/ACM International Symposium on Cluster Computing and the Grid, 2009. CCGRID'09. IEEE, 2009, pp. 124–131.

[58] D. Milojičić, I. M. Llorente, and R. S. Montero, "Opennebula: A cloud management tool," IEEE Internet Computing, vol. 15, no. 2, pp. 11–14, 2011.

[59] Apache Software Foundation. (2014, Feb.) Apache cloudstack. [Online]. Available: http://cloudstack.apache.org/

[60] K. Jackson, OpenStack Cloud Computing Cookbook. Packt Publishing, 2012.

[61] OpenStack. (2016) ReleaseNotes/icehouse - OpenStack. [Online]. Available: https://wiki.openstack.org/wiki/ReleaseNotes/Icehouse

[62] KVM. (2016) Qcow2 - KVM. [Online]. Available: http://www.linux-kvm.org/page/Qcow2

[63] Pivotal Software, Inc. (2014) RabbitMQ - messaging that just works. [Online]. Available: http://www.rabbitmq.com/

[64] A. Videla and J. J. W. Williams, RabbitMQ in Action: Distributed Messaging for Everyone. Manning Publications, May 2012.

[65] Docker, Inc. (2016) What is Docker? https://www.docker.com/what-docker. [Online]. Available: https://www.docker.com/what-docker

APPENDIX A

NOKKHUM HANDBOOK

This appendix presents how to install and use the Nokkhum system. The program installation section presents programs, tools, and libraries installation step for a front-end node and a computing node.

A.1 Software Installation

Nokkhum is mainly written in Python in many parts, and in C++ in the Nokkhum processor. The Nokkhum project is hosted on github.com as described in Table A.1. Many components, required libraries and middlewares are shown in Tables A.2, A.3 and A.4.

Table A.1: Nokkhum components on github.com.

Nokkhum components	URL
Nokkhum Controller and Nokkhum Compute Node Workder	https://github.com/sdayu/nokkhum
Nokkhum Processors	https://github.com/sdayu/nokkhum-processor
Nokkhum API	https://github.com/sdayu/nokkhum-api
Nokkhum Web Client	https://github.com/sdayu/nokkhum-web

Nokkhum can be installed applying many topologies as described in Figures 3.12 - 3.18. This appendix shows the installation example identified in Figure 3.15. The deployed system consists of two types of servers. The first type is the front-end node and the second type is the computing node. This installation manual only describes the setup for the Linux operating system. The below steps must be done in the following order.

A.1.1 Front-end Node Installation and Configuration

This section describes how to install and configure the front-end node which consists of a MongoDB, RabbitMQ, Nokkhum API, Nokkhum web, and Nokkhum controller as described in Figure 3.15.

Table A.2: Required Python libraries and tools.

Libraries and tools	Version	URL
Python	>= 3.5	https://www.python.org/downloads/release/python-350/
mongoengine	>= 0.10.0	http://mongoengine.org/
psutil	>= 3.4.0	https://github.com/giampaolo/psutil
amqp	>= 1.4.9	http://github.com/celery/py-amqp
netifaces	>= 0.10.4	https://bitbucket.org/al45tair/netifaces
numpy	>= 1.11.0	http://www.numpy.org
scipy	>= 0.17.0	http://www.scipy.org
boto	>= 2.40	https://github.com/boto/boto/
python-dateutil	>= 2.5.3	https://dateutil.readthedocs.org
pyramid	>= 1.6	http://docs.pylonsproject.org/en/latest/docs/pyramid.html
pyramid_jinja2	>= 2.6.2	https://github.com/Pylons/pyramid_jinja2
pyramid_beaker	>= 0.8	http://docs.pylonsproject.org/projects/pyramid_beaker/en/latest/
wtforms	>= 2.1	http://wtforms.simplecodes.com
pycrypto	>= 2.6.1	http://www.pycrypto.org
requests	>= 2.10.0	http://python-requests.org

Table A.3: Required C++ libraries and tools.

Libraries and tools	Version	URL
G++	>= 5.1	https://gcc.gnu.org/gcc-5/
OpenCV	>= 3.0	http://opencv.org/
CMake	>= 2.8	https://cmake.org/
boost	>= 1.58	http://www.boost.org/
google glog	>= 0.3.4	https://github.com/google/glog
jsoncpp	>= 1.7.2	https://github.com/open-source-parsers/jsoncpp
Poco C++	>= 1.3.6	http://pocoproject.org/

Table A.4: Required middleware and database.

Software	Version	URL
OpenStack	>= 13.0.0	http://www.openstack.org
RabbitMQ	>= 3.5.7	http://www.rabbitmq.com
MongoDB	>= 2.4	https://www.mongodb.com

A.1.1.1 Prerequisites

First of all, update the debian repository. Then, install the RabbitMQ, a MongoDB server and essential tools.

```
# apt update
# apt install rabbitmq-server
# apt install mongodb-server
# apt install python3 python3-dev python3-pip python3-venv  git
```

After having completed the packages installation, the RabbitMQ server requires to add a user to the system and setup permissions.

```
# rabbitmqctl add_user nokkhum NOKKHUM_PASSWD
Creating user "nokkhum" ...
...done.
# rabbitmqctl set_permissions nokkhum ".*" ".*" ".*"
Setting permissions for user "nokkhum" in vhost "/nokkhum" ...
...done.
```

A.1.1.2 Nokkhum API Installation and Configuration

1. Create a Python virtual environment.

   ```
   $ pyvenv nkapi-env
   $ source nkapi-env/bin/activate
   (nkapi-env) $
   ```

2. Clone the Nokkhun API source code from github.com and the initial submodule.

   ```
   (nkapi-env) $ git clone --branch v0.1.0 https://github.com/sdayu
       /nokkhum-api.git
   (nkapi-env) $ cd nokkhum-api
   (nkapi-env) $ git submodule update --init
   ```

3. Setup Nokkhum with an automatic script, it will simultaneously install required libraries identified in the setup script.

   ```
   (nkapi-env) $ python setup.py develop
   ```

4. Copy the sample configuration and setup a new configuration.

   ```
   (nkapi-env) $ cp productoion.ini.sample production.ini
   ```

   Change the red literals according to your own configuration.

```
[app:main]
use = egg:nokkhum-api

pyramid.reload_templates = true
pyramid.debug_authorization = false
pyramid.debug_notfound = false
pyramid.debug_routematch = false
pyramid.default_locale_name = en

mongodb.host = localhost
mongodb.db_name = nokkhum

nokkhum.auth.secret = NOKKHUM-SECRET

# cloud storage
nokkhum.storage.s3.host = S3-HOST-IP
nokkhum.storage.s3.port = 8080
nokkhum.storage.s3.access_key_id = S3-ACCESS-KEY-ID
nokkhum.storage.s3.secret_access_key = S3-SECRET-ACCESS-KEY
nokkhum.storage.s3.secure_connection = false
nokkhum.temp_dir = /tmp/nokkhum-api/cache

nokkhum.api.ip = FRONT-END-IP
```

5. Start the Nokkhum API server.

```
(nkapi-env) $ pserve --reload production.ini
```

A.1.1.3 Nokkhum Web Installation and Configuration

Install an additional front-end javascript package manager

```
# apt install npm
# npm install -g bower
```

1. Create a Python virtual environment.

```
$ pyvenv nkweb-env
$ source nkweb-env/bin/activate
(nkweb-env) $
```

2. Clone the python-nokkhumclient source code from github.com and setup the Nokkhum client with the automatic script.

```
(nkweb-env) $ git clone --branch v0.1.0 https://github.com/sdayu
    /python-nokkhumclient.git
(nkweb-env) $ cd python-nokkhumclient
(nkweb-env) $ python setup.py develop
(nkweb-env) $ cd ..
```

3. Clone the Nokkhun Web source code and setup the Nokkhum client with the automatic script.

```
(nkweb-env) $ git clone --branch v0.1.0 https://github.com/sdayu
    /nokkhum-web.git
(nkweb-env) $ cd nokkhum-web
(nkweb-env) $ python setup.py develop
```

4. Install JavaScript for the front-end framework.

```
(nkweb-env) $ bower install
```

5. Copy the sample configuration and setup a new configuration.

```
(nkweb-env) $ cp productoion.ini.sample production.ini
```

Change the red literals according to your own configuration.

```
[app:main]
use = egg:nokkhum-web

pyramid.reload_templates = true
pyramid.debug_authorization = false
pyramid.debug_notfound = false
pyramid.debug_routematch = false
pyramid.debug_templates = true
pyramid.default_locale_name = en
pyramid.includes = pyramid_tm
                pyramid_jinja2
                pyramid_beaker

jinja2.directories = nokkhumweb:templates

nokkhum.auth.secret = NOKKHUM—SECRET

nokkhum.api.host = PUBLIC—INTERFACE
nokkhum.api.port = 6543
nokkhum.api.secure_connection = false

session.expire = 600
cache.regions = day, hour, minute, second
cache.type = memory
cache.second.expire = 1
cache.minute.expire = 60
cache.hour.expire = 3600
cache.day.expire = 86400
```

6. Start the Nokkhum web server.

```
(nkweb-env) $ pserve --reload production.ini
```

7. Open web browser and goto http://PUBLIC-INTERFACE:6544 presents the Nokkhum web index as shown in Figure A.1.

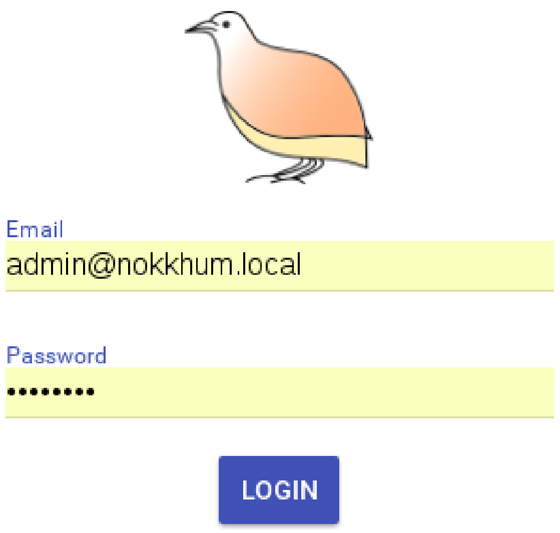

Email
admin@nokkhum.local
Password
••••••••

Figure A.1: Nokkhum web index.

A.1.1.4 Nokkhum Controller Installation and Configuration

1. Create a Python virtual environment.

```
$ pyvenv nkcontroller-env
$ source nkcontroller-env/bin/activate
(nkcontroller-env) $
```

2. Clone the nokkhum source code from github.com and setup the nokkhum controller with the automatic script.

```
(nkcontroller-env) $ git clone --branch v0.1.0 https://github.
    com/sdayu/nokkhum.git
(nkcontroller-env) $ cd nokkhum
(nkcontroller-env) $ python setup.py develop
```

3. Copy the sample configuration and setup a new configuration.

```
(nkcontroller-env) $ cp controller-config.ini.sample controller-
    config.ini
```

Change the red literals according to your own configuration.

```
[DEFAULT]
nokkhum.log_dir = /tmp/nokkhum-log

[controller]
nokkhum.controller.interface = eth0
nokkhum.scheduler.processor.heuristic = avg

mongodb.host = localhost
mongodb.db_name = nokkhum

amq.url = amqp://nokkhum:NOKKHUM-PASSWD@localhost:5672/nokkhum

nokkhum.storage.enable = true
nokkhum.storage.api = s3
nokkhum.storage.s3.host = S3-HOST-IP
nokkhum.storage.s3.port = S3-HOST-PORT
nokkhum.storage.s3.access_key_id = S3-ACCESS-KEY-ID
nokkhum.storage.s3.secret_access_key = S3-SECRET-ACCESS-KEY
nokkhum.storage.s3.secure_connection = false

nokkhum.temp_dir = /tmp/nokkhum-web/cache

nokkhum.vm.enable = false # change to true when using EC2
nokkhum.vm.api = ec2
nokkhum.vm.ec2.host = S2-HOST-IP
nokkhum.vm.ec2.port = EC2-HOST-PORT
nokkhum.vm.ec2.access_key_id = EC2-ACCESS-KEY-ID
nokkhum.vm.ec2.secret_access_key = EC2-SECRET-ACCESS-KEY
nokkhum.vm.ec2.secure_connection = false
nokkhum.vm.ec2.image.name = NOKKHUM-IMAGE
nokkhum.vm.ec2.instance_type = l1.medium, l1.large

nokkhum.information.removal = 30
```

4. Start the Nokkhum controller.

```
(nkweb-env) $ bin/bin/nokkhum-controller controller-config.ini
```

A.1.2 Computing Node Installation and Configuration

A computing node consists of two Nokkhum components. First is a Nokkhum compute node worker and the other is a Nokkhum processor.

A.1.2.1 Prerequisites

First of all, update the debian repository. Then, install essential tools for python and C++.

```
# apt update
# apt install python3 python3-dev python3-pip python3-venv  git
# apt install cmake g++ pkg-config libboost-filesystem-dev libboost-
   program-options-dev libboost-date-time-dev libopencv-dev libgoogle
   -glog-dev libjsoncpp-dev
```

A.1.2.2 Nokkhum Compute Node Installation and Configuration

1. Create Python virtual environment.

```
$ pyvenv nkcompute-env
$ source nkcompute-env/bin/activate
(nkcompute-env) $
```

2. Clone the nokkhum source code from github.com and setup the nokkhum compute node worker with the automatic script.

```
(nkcompute-env) $ git clone --branch v0.1.0 https://github.com/
   sdayu/nokkhum.git
(nkcompute-env) $ cd nokkhum
(nkcompute-env) $ python setup.py develop
```

3. Copy the sample configuration and setup a new configuration.

```
(nkcompute-env) $ cp compute-config.ini.sample compute-config.
   ini
```

Change the red literals according to your own configuration.

```
[DEFAULT]
nokkhum.log_dir = /tmp/nokkhum-log

[compute]
nokkhum.processor.path = PATH-TO-NOKKHUM-PROCESSOR-PROJECT/build
   /nokkhum-processor
```

```
nokkhum.processor.record_path = /tmp/nokkhum-records
nokkhum.compute.interface = eth0

amq.url = amqp://nokkhum:NOKKHUM-PASSWD@FRONT-END-IP:5672/
    nokkhum
```

4. Start the Nokkhum controller.

```
(nkcompute-env) $ bin/bin/nokkhum-compute compute-config.ini
```

A.1.2.3 Nokkhum Processor Installation and Configuration

1. Configure the cmake to build Nokkhum processors

```
$ mkdir build
$ cd build
$ cmake ..
```

2. Compile Nokkhum processors

```
$ make
```

APPENDIX B

PUBLICATIONS

B.1 Conference Proceedings

B.1.1 T. Limna and P. Tandayya, “Design for a flexible video surveillance as a service,” in *proceedings of 2012 5th International Congress on Image and Signal Processing (CISP)*, 2012, pp. 197–201. Chongqing, Sichuan, China, IEEEXplore.
doi:10.1109/CISP.2012.6469742

B.1.2 T. Limna and P. Tandayya, “Video surveillance as a service cost estimation and pricing model,” in *proceedings of 2015 12th International Joint Conference on Computer Science and Software Engineering (JCSSE)*, 2015, pp. 174–179. Hat Yai, Songkla, Thailand, IEEEXplore.
doi:10.1109/JCSSE.2015.7219791

Design for a Flexible Video Surveillance as a Service

Thanathip Limna and Pichaya Tandayya
Department of Computer Engineering
Faculty of Engineering, Prince of Songkla University
Hat Yai, Songkhla, Thailand
thanathip.limna@gmail.com, pichaya@coe.psu.ac.th

Abstract—**Current video surveillance systems are popular on standalone applications. A free video surveillance software usually fixes its configuration on a single desktop. This makes it is hard to scale the system up for supporting many cameras. Moreover, it also needs a one-stop management service method. Our research tries to enhance OpenVSS, an open source video surveillance system, for Video Surveillance as a Service (VSaaS) so that it can be used by multiple users and runs on Infrastructure as a Service of Cloud Computing in the future.**

Keywords-Video Surveillance as a Service, Cloud Computing

I. INTRODUCTION

A video surveillance system [1] provides an important role for security and popularly used to observe indoor events. The video surveillance system needs a video analysis software and storage space to keep records up to the owner's requirement. Currently, the small video surveillance software usually is a standalone application and bundles with a closed circuit television and Internet Protocol cameras (IP cameras). The software that is able to support a lot of cameras is expensive and needs complex system configuration.

The Software as a Service (SaaS) is a famous software model that delivers services to users via the Internet. The users do not need to have high skills in system configuration and it reduces a usability learning curve. We are interested in applying SaaS to a video surveillance system for providing Video Surveillance as a Service (VSaaS). The VSaaS is providing on the Internet must covers massive camera management, different camera configuration, massive storage space, supporting dynamic user requirement, etc. The advantage of the VSaaS is that the user do not need to mind about software maintenance and can choose new image processing solutions as soon as the provider deploys them without new system installation on the user side. The user can have less concern about system software/hardware maintenance, and focus on providing IP-cameras only. On the other hand, the provider may compete with each other to reduce the operation cost and provide best quality image processing to users.

The VSaaS on current market provides simple image processing such as motion detection with specific areas of interest, storage space and alert system. Technologies and VSaaS architecture behind the scene lack of information due to trade secret. Some open video surveillance systems are unsuitable for SaaS models. They are appropriately designed according to the software objectives for best system performance. However, there still are needs to research upon dynamic user request and system scalability. Consequently, we simply develop and research on a VSaaS in terms of software system architecture, system behavior and user requirement. This paper identifies the scalable VSaaS software system architecture that easy to increasing a computation node worker for supporting increasing IP-cameras.

Two main parts of our proposed VSaaS architecture which are a controller and a compute node worker communicate with each other using message passing. Therefore, our VSaaS tasks are easily distributed to multiple computational nodes for supporting elastic requirement. All of video records are collected into the cloud storage and the user can access to playback them via a web site. The video records and images are stored on cloud storage as long as the user requires.

II. VIDEO SURVEILLANCE AS A SERVICE

A. *Current Video Surveillance Application*

At the present, video surveillance systems support Internet Protocol cameras (IP cameras) [2] which are connected to a Network Video Recorder (NVR) or a generic computer for image processing. The video camera management and image processing software are usually bundled with video cameras or sold separately in case of high performance software that can manage a lot of cameras and provide special image processing. Normally, free video management software is specifically limited to only certain types of cameras and cannot combine cameras from different manufacturers. In addition, if anyone wants to install a video surveillance system, one has to pay for the first time installation, maintenance and computer software costs.

There are many free video surveillance software that work on standalone machine such as ODViS [3], OpenVSS [4] and Zoneminder [5]. They have complete video surveillance processing on a single computer. Some software are easy to install, configure and use but most software components are not scalable. An easy way to scale up this kind of systems is adding a new computer worker and re-configure it but when the user requires many cameras, it is hard to manage and maintain the system. The other type of video surveillance system architectures is a distributed system, that its architecture consists of many components and processing layers such as IVSS [6], DiVA [7], CANDELA [8], etc. It is designed for

Thailand Research Fund through the Royal Golden Jubilee Ph.D. Program (Grant No. PHD/0047/2552) and Prince of Songkla University

supporting massive cameras and different configurations. However, the architecture needs best configuration as it separates modules to run on different machine and requires close work cooperation.

B. *Video Surveillace as a Service*

Current VSaaS on market has provided a network video recorder and storage space to collect video records. Normally, the system has motion analysis for applying to the video recorder. The motion analysis filters the event footage for reducing the storage space in collecting video records. The VSaaS has a service plan for customers and the customers choose the plan that is fit to their requirements.

It is well known that a video surveillance system must have appropriate storage space corresponding to the time to store the video record. Therefore, each stored video record from a camera requires a lot of memory. In addition, the maintenance cost for video cameras, computers and administration is next to be considered. To reduce these costs, we need to provide a video surveillance service in terms of SaaS over the Internet. The characteristic of video cameras changes from analog to digital in IP cameras and SaaS on the Internet is widely spread in the market. Therefore, there are attempts to transform video surveillance system models into Internet services like Video Surveillance as a Service (VSaaS). This kind of services can solve the storage space problem, support a lot of video image processing and allow the growth in the number of video cameras in the future. The users have to pay fees upon their organization's requirements. Although there are video surveillance services in the market such as SecurityStation [9], Secure-i Hosted Video Recorder™ [10], and OVS™ [11], the technical issues behind the scenes to provide the services, for example, resource management and cost-effective optimization, still are not disclosed.

III. DESIGN OF VIDEO SURVEILLANCE AS A SERVICE

Our VSaaS called "Nokkhum" is implemented by Python and C++, and divided into four parts: controller, compute node worker, image processor and web front-end. The controller is a daemon process that handles the user requests from a web site and pushes commands to available compute nodes using the message-oriented middleware. The compute node worker gets commands from the controller via the message broker, manages its image processor and monitors its resources for reporting back to the controller. For a basic image processor, we investigate some components of OpenVSS and rewrite the code to fit to our system design based on OpenCV [12].

A. *Component Overview*

The video surveillance system have four sections as shown in Figure 1. Each section is a separate program developed using C++ and Python. The controller pushes message commands to the computer node worker via the message broker and the compute node worker builds the image analysis process. The basic architecture is shown in Figure 2.

- A controller is a task and resource control which is an important part of the system. It is a daemon process that monitors user requirements and system resources.
- A compute node worker is a daemon process that runs on each computation node. It provides computer resource monitoring, camera monitoring and image processing deployment interface. The compute node uses pipes for communicating with image processors and reports information resource updates to the controller using message passing.
- An image processor provides an image processing solution via JSON. It parses JSON information from the compute node pipe as a standard input and constructs image processing threads. The image processor is a computer process controlled by the compute node worker.
- Web front-end is a web user interface for managing and controlling cameras and image processors.

Nokkhum VSaaS architecture in Figure 2 shows a conceptual design. The user composes the camera configuration on web interface then the web site pushes the command to the database. When the task scheduler on a controller is active, it gets the user command, finds available resources and pushes the message to a compute node worker via the message server. The compute node worker gets the message, then builds a video surveillance solution and starts image processors. When the image processors are working and producing the output data (video record, image), then the compute node worker stores the data at the cloud storage. The user can watch, download and delete the video records and images from web site. The controller also monitors and manages the time to storage of video records and images.

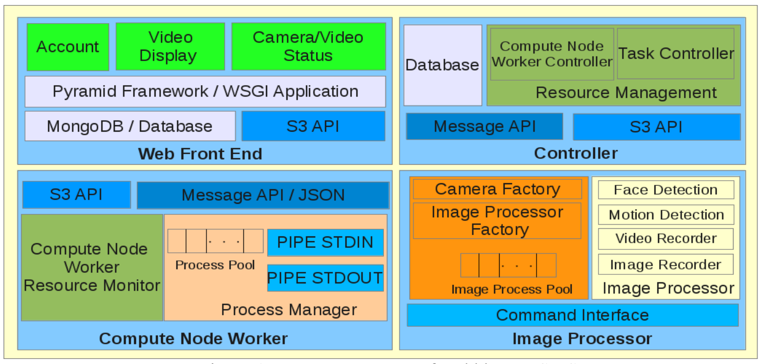


Figure 1: Four components of Nokkhum VSaaS

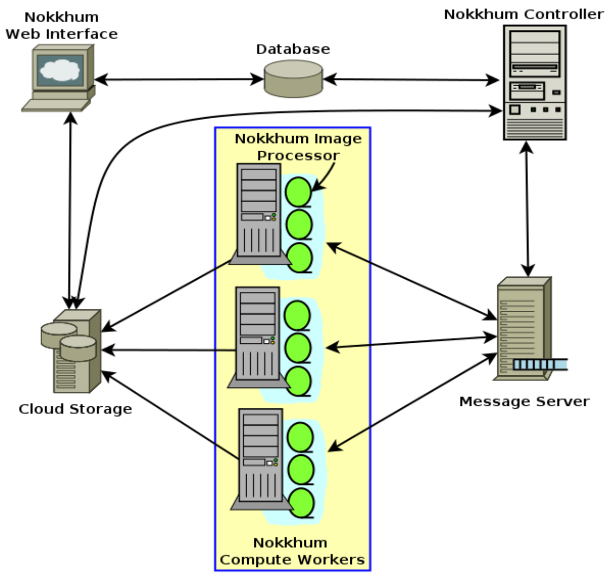


Figure 2: Nokkhum VSaaS architecture

1) Controller

Video surveillance controller and compute node worker communicate via the message passing method. We have implemented the controller to handle users' video surveillance requirements. The controller can call a compute node worker to start a surveillance application. The controller has been developed using Python, MongoDB [13] and connected to the compute node worker using message passing.

At the present, the controller has basic camera monitoring and task scheduling. It can handle resource information from the compute node and push camera action commands to the compute node in order to start, stop and check the camera image processing status. The compute node can handle actions from the controller and trigger order to image processors.

2) Compute Node Worker

The compute node worker consolidates image processing tasks assigned by the controller. The tasks contain many image analytical solutions such as motion detection and face detection. The compute node worker has resource sensors to monitor the computer resource capacity and serve the information for the controller's monitoring.

A compute node worker includes resource sensors for monitoring the resource usage such as CPUs utilization and memory capacity. Then, the resource information will be delivered to the controller via message passing. The last component of a compute node is the task management module which controls and monitors tasks and the task pool. This module manages the task life cycle such as create and terminate states. In addition, the task management module may watch task behaviors and handle shortcomings in order to increase the system availability.

A task runs on a compute node worker consists of image processing solutions for a video surveillance system. The task bundles image processing solutions that apply to the same IP camera. Therefore, we need to reduce the internal network bandwidth for image delivery between compute node workers. In addition, when a compute node worker clashes, then the tasks that run on it will stop without affecting other compute node workers.

3) Image Processor

The video surveillance computation includes image processors that is a set of image analysis build from configuration based on the OpenCV library as shown in Table 1.

Table 1. Image processors and their implementation

Image Analysis	Implementation
Motion detector	OpenCV optical flow
Face detector	OpenCV object detection with cascade classifier
Video recorder	OpenCV video function
Image recorder	OpenCV image function

According to image processor conceptual design, an image analysis is a thread that passes its output to another image analysis via an image queue as shown in Figure 3.

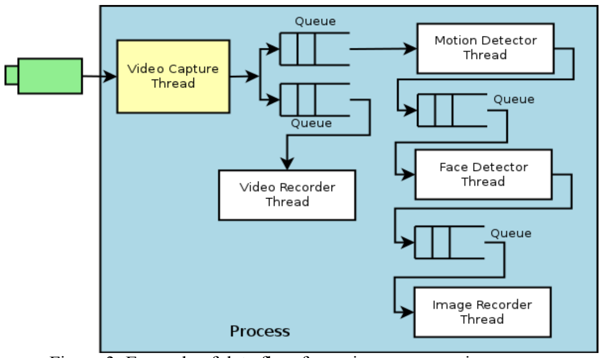


Figure 3. Example of data flow for an image processing process

In Figure 3, the data flow of the image processor starts when an IP camera captures images via the video capturing thread. The capturing thread pushes the image matrix into multiple queues upon image analysis. After the image analysis thread begins data processing, it will then pass the image matrix output to the next image analysis via the queues.

A camera configuration and processor attributes are described using JSON for building an image processing solution as shown in Figure 4. The camera configuration is a dictionary containing all attributes such as name, URL, width, height, etc. The processor attributes are listed identified by "processors" keyword, which contain JSON dictionary of attributes. The list of processor attributes allows nested description for complex configuration.

```
{ "camera" : {
        "fps" : 10,
        "height" : 240, "width" : 320,
        "model" : "Logitech",
        "name" : "Camera 1",
        "url" : "rtsp://url.to.camera/stream",
         },
    "processors" : [ {
            "interval" : 2,
            "name" : "Motion Detector",
            "processors" : [ {
                    "directory" : "./",
                    "fps" : 20,
                    "height" : 240, "width" : 320,
                    "name" : "Video Recorder" } ],
         } ]
}
```

Figure 4. Example of camera and image processors configuration using JSON

4) Web Front End

The web front end is an interface for users and the administrator. It has been developed by Python and HTML to provide camera and system information up to the user group permission. The user can create new camera configuration, compose available image analysis and watch videos and images from the cloud storage. The web interface is developed by Python, Pyramid [14], HTML5 and MongoDB.

B. Inter-component Communication

There are 2 interesting parts of inter-component communication, one includes a controller and compute node workers and the other includes compute node workers and image processor. In this section, we present the communication method between component.

1) Controller and Compute Node Worker

According to Figure 2, the controller and compute node workers communicate with message passing method. There are two types of exchanges in the communication: direct exchange and topic exchange. The controller and the compute node worker employ direct exchange for greeting messages and updating system resources because the compute node does not need to wait for a response to do something. The controller and the compute node worker use topic exchange for synchronizing command messages such as start/stop an image processor and request new greeting information. The topic exchange is designed to wait for responding messages and command confirmation.

When a new compute node worker starts, it will send a greeting message to the message broker (The message broker address must appear in the compute node worker configuration), then the controller will receive a greeting message and push the compute node worker to the resource pool. After receiving the greeting message, the compute node pushes to update system resource usage including running image processors on this compute node worker, CPU utilization and memory usage.

2) Compute Node Worker and Image Processor

A compute node worker and an image processor has communicate with the pipe as the image processor is a process created by the compute node worker. The compute node worker writes a command to the standard input and reads results from the standard output of the image processor. The command and result are written in the JSON format. The compute node worker checks available image processors and reports to the controller via an update system resource message.

IV. EXPERIMENTAL RESULT

We have set up a testing environment, running Nokkhum VSaaS on two physical machines with AMD Phenom™ II X6 1055T Processor, 8 GB DDR RAM, 2 TB hard disk and Linux operating system. The first machines includes message broker (RabbitMQ [15]), database server (MongoDB), Nokkhum controller, Nokkhum compute node worker, Nokkhum processor and Nokkhum web front end. The second machine includes Nokkhum compute node worker and Nokkhum processor for only video processing.

We have 12 available video streams adding to the system with different configurations. The system can assign processing tasks to Nokkhum compute node workers. There are several scenarios as follows:

- In case of the electricity is cut or the network is down at a video camera site, the system can automatically recovery after the IP camera is available.
- When a compute node worker has problem, for example, network, program crash and disconnection to the message broker, the system should provide an available compute node worker for starting a new run.
- When the Nokkhum controller is down and the compute node workers are running, the video processors can continue working without an effect. When the controller resume running, it can recover to the current status by processing message information from the message broker server and database server.
- The system can start with a single machine and scale the system up by adding new worker machines in the future.

As the testing system has been run more than two months, our Nokkhum VSaaS works well on above scenarios. The controller is able to process messages from any compute node worker. However, the limitation of the number of compute node workers supported by the controller is to be further researched. The number of compute node workers is a key to determine whether the system should start a new controller. In our experiment, a controller can process two compute node workers that can support more than 30 video streams according to each camera configuration.

The Nokkhum VSaaS works as expected for serving a video surveillance application on the Internet. It is suitable for a private organization that has many IP cameras. Some more technical issues need to be further researched such as network bandwidth, security, service fee, etc., for its expandable service running on the global Internet.

V. CONCLUSION

Nokkhum is novel design and implementation for VSaaS in order to support dynamic user requirement. Nokkhum components make it easy to distribute image processing tasks to compute node workers via a message broker. This advantage makes the VSaaS can scale up the number of compute node workers in order to handle additional cameras and support unpredictable user requirements. This system helps the administrator find available resources in order to start a surveillance application automatically. It handles common image processing issues and tries to solve them when the conditions are ready (networks, electricity, computing resources). Nokkhum has provided a web interface for users to manage their camera configuration, design their own surveillance processes and arrange image or video records. In future work, Nokkhum will provide some more modules to run on Infrastructure as a Service (IaaS) of Cloud Computing for automatic system scalability. It is easily scale the system according to user requirement and cost-effective optimization.

Nokkhum is distinguished form other surveillance applications as follows: Nokkhum has cloud storage back-end for keeping image and video records. Design components are prepared to run on IaaS a cloud computing environment. All modules require a few resources at the start-up time and acquire more resources when adding more cameras. This advantage affects the operation cost. The administrator does not need a long time to plan for supporting the requirements in the future.

Acknowledgment

The authors are grateful for financial supports from the Thailand Research Fund through the Royal Golden Jubilee Ph.D. Program (Grant No. PHD/0047/2552) and Prince of Songkla University.

References

[1] M. Valera and S. Velastin, “Intelligent distributed surveillance systems: a review,” in *IEE Proceedings Vision, Image and Signal Processing*, vol. 152, no. 2, pp. 192-204, 2005.

[2] Georis, J. F. Delaigle, B. Macq, X. Desurmont, D. Demaret, and S. Redureau, “IP-distributed computer-aided video-surveillance system,” in *COLLOQUIUM DIGEST-IEE*, pp. 18-18, 2003.

[3] C. Jaynes, S. Webb, R. M. Steele, and Q. Xiong, “An Open Development Environment for Evaluation of Video Surveillance Systems,” in *Proceedings of the Third International Workshop on Performance Evaluation of Tracking and Surveillance(PETS'2002)*, vol. 1, p. 32–39, 2002.

[4] N. Suvonvorn, “A video analysis framework for surveillance system,” in *2008 IEEE 10th Workshop on Multimedia Signal Processing*, pp. 867-871, 2008.

[5] Triornis Ltd. (2012, February 21). zoneminder. [Online]. Available: http://www.zoneminder.com

[6] X. Yuan, Z. Sun, Y. Varol, and G. Bebis, “A distributed visual surveillance system,” in *Proceedings IEEE Conference on Advanced Video and Signal Based Surveillance, 2003.*, 2003, pp. 199 - 204.

[7] J. C. San Miguel, J. Bescos, J. M. Martinez, and A. Garcia, “DiVA: A Distributed Video Analysis Framework Applied to Video-Surveillance Systems,” in *Image Analysis for Multimedia Interactive Services, 2008. WIAMIS '08. Ninth International Workshop on*, 2008, pp. 207-210.

[8] R. G. J. Wijnhoven, E. G. T. Jaspers, and P. H. N. de With, “Flexible surveillance system architecture for prototyping video content analysis algorithms,” in *Proceedings of SPIE*, vol. 6073, no. 1, p. 60730R–60730R–9, Jan. 2006.

[9] NW Systems Group Ltd. (2012, February 20). How SecurityStation works. [Online]. Available: http://www.securitystation.com/how-vsaas-works.php

[10] Secure-i. (2012, February 20). Clear and Simple. HVR™ is changing security one camera at a time. [Online]. Available: http://www.secure-i.com/learn/technologies

[11] Neo IT Solutions Inc. (2012, February 20). OVS™ Technology. [Online]. Available: http://www.neovsp.com/solutions/webvsp

[12] Gary B. and A. Kaehler, *Learning OpenCV.* O'Reilly Media, Inc., 2008.

[13] K. Chodorow and M. Dirolf, *MongoDB: The Definitive Guide.* O'Reilly Media, Inc., 2010.

[14] C. McDonough, (2012, February 20). The Pyramid Web Application Development Framework. [Online]. http://media.readthedocs.org/pdf /pyramid/1.3-branch/pyramid.pdf

[15] SpringSource, (2012, February 20). RabbitMQ – Documentation. [Online]. VMware Inc., Available: http://www.rabbitmq.com/ documentation.html

Video Surveillance as a Service Cost Estimation and Pricing Model

Thanathip Limna and Pichaya Tandayya
Department of Computer Engineering
Prince of Songkla University
Hatyai, Songkhla, Thailand 90112
Email: thanathip.limna@gmail.com, pichaya@coe.psu.ac.th

***Abstract*—There were many research works about video surveillance as a service system. Some concerned system architecture. Some identified cost optimization for the deployment of video surveillance systems using cloud computing technology. However, there was not much information about budget calculation for system provision. This paper addresses a cost investigation for video surveillance providers to use as a guideline for budget preparation. In addition, we propose a pricing model for monthly package based on real load resource usage.**

***Keywords*—Cost Estimation, Pricing Model, VSaaS, Real Load**

I. Introduction

Currently, applying cloud computing technology to drive online services becomes a popular trend for modern Software as a Service (SaaS). The SaaS is spreadly used in many software categories such as accounting, office, entertainment, banking, and also video surveillance. Nowadays, there are many video surveillance system providers providing online image analysis and storage space for home customers, being known as Video Surveillance as a Service (VSaaS) [1]. The customer only has to buy Internet Protocol cameras (IP cameras), then register and connect the IP cameras to an online system. The provider accepts the IP camera connectivity and starts the image analysis according to the customer's choice of configuration and service package. The VSaaS paradigm is shown in Fig. 1. Most VSaaS providers charge the customer according to software features and timing. Many VSaaS

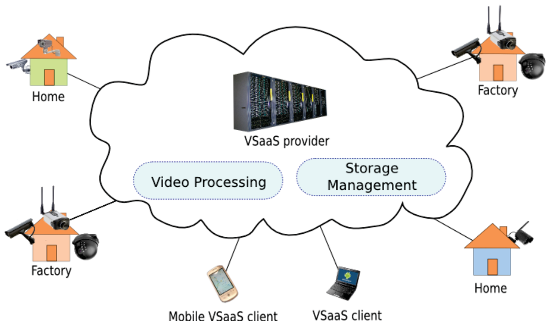


Fig. 1. VSaaS paradigm

providers provide video surveillance services based on their technologies which can be categorized into two types. The first one operates VSaaS services using physical servers, and the other applies Virtual Machines (VMs) and cloud computing technology. The two kinds of implementations infer different costs and also service charges to both service providers and customers. Therefore, we are interested in investigating the costs for VSaaS providers in terms of CPU and Memory usage.

II. Video Surveillance as a Service

Video Surveillance as a Service is well known as video surveillance provided on the Internet. Customers just need to have IP cameras and a connection to the Internet. They do not have to mind about recording hardwares and storage space. They can register with the VSaaS provider and connect their IP cameras to an online software system. After finishing all configurations, the VSaaS system will start image analysis and keep its results in the storage. The customer can access a VSaaS for live monitoring and video record playback via VSaaS client. The customer has to pay for a service charge according to the package or service plan chosen during the registration process. There are advantages using VSaaS as follows.

- The customer does not implicitly pay for mantainance cost for recording hardware or storage space.
- The customer gets a new version of software as soon as it is ready for deployment.
- The customer can add or remove cameras, and apply image analysis as required.

There are serveral cloud-based video surveillance built with Hadoop [2] such as P2PCloud [3], VAQACI [4], and the CVR system [5]. Another approach uses Amazon Web Services (AWS) [6] platform as appearing in many researches such as works conducted by Hossain *et al.* [7], Rodriguez-Silva *et al.* [8], and Nokkhum [9]. Most example systems focus on video surveillance system architecture more than cost analysis. While the cost for providing VSaaS is necessary for VSaaS providers in estimating budget and pricing the service charge. In this paper, we use our Nokkhum system as a reference architecture, because we can measure the image analysis processing part that helps with cost estimation and pricing service charges.

A. Nokkhum VSaaS

Nokkhum is a flexible component-based video surveillance architecture [9] that can deploy any components on both physical servers and virtual machines using the Amazon EC2 API. It applies the Amazon S3 API for storing pictures and video records. Nokkhum can flexibly configure one or more servers so that its provisions are suitable for different organizations and business sizes. It has API interfaces server for providing a REpresentational State Transfer (REST) over HTTP [10] to support any client platform to control, view, or manage image analysis, camera configuration, and video recording. The Nokkhum architecture paradigm is shown in Fig. 2. Fig. 2 shows the components of Nokkhum architecture

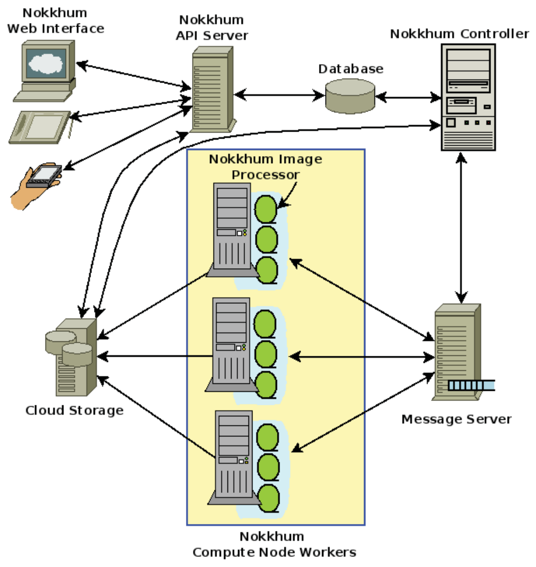


Fig. 2. Nokkhum VSaaS architecture [5]

which includes web interface, APIs, controller, compute node worker, and image processor. The scalability of Nokkhum benefits from a message oriented middleware that enables Nokkhum to distribute its components to run on many servers. The image processor component contains only image analysis and its computation starts with JSON configuration. In this paper, we observe and monitor the resource usage of image analysis of Nokkhum services.

III. VSaaS Price Investigation

Currently, the VSaaS market has many providers. Examples are shown in Table I. Many providers focus on VSaaS infrastructure implementation and the prices are up to the resellers to deal with the customers. Some providers point their positioning on only enterprise solutions or sell the software in bundle with IP cameras. Small groups of existing VSaaS system provide services to customers directly and quote fee rates per their convenient minimum period, mostly in month. This section shows our investigation on current VSaaS infrastructure costs per month and all possible costs per month.

A. Infrastructure Costs

In general, a VSaaS system can be deployed onto three different platforms: first is on a physical server, second is on virtual machines, and third is using cloud provision. This

TABLE I. Examples of VSaaS in the market

Name	Home page	Service plan
Alarm.com	https://www.alarm.com/productservices/video_monitoring.aspx	Bundled with IP camera
AXIS Video Hosting System (AVHS)	http://www.axis.com/hosting	Bundled with IP camera
Brivo	http://www.brivo.com/	Reseller
byRemote Surveillance Solutions	http://www.byremote.net	Reseller
CameraManager	https://www.cameramanager.com/website/en/plans	Monthly cost/camera
CheckVideo	https://www.checkvideo.com	Enterprice, Reseller
Connexed	http://www.connexed.com/pricing.html	Month cost/Camera, Quarterly cost/Camera, Annually cost/camera
Dropcam	https://www.dropcam.com/store	Annually cost/camera

section investigates on current infrastructure costs for deploying a VSaaS system on all three mentioned platforms. We first normalize the infrastructure cost into a monthly rate and categorize into computation cost per month (C_{cpm}) and storage cost per month (C_{spm}).

1) Physical Server Cost: A physical server is a generic server available in the market. When providers deploy their VSaaS, they have to pay for the physical server (C_{server}). The server will be placed in a data center for reasons concerning stability, security, and reliability of network bandwidth, electricity, etc. Then, the VSaaS providers also have to pay for colocation data center service charge ($C_{colocation}$) as an additional cost. Apart from the colocation cost, many data center providers also charge the first time set-up price (C_{fts}) for a new server, which is an extra cost. The monthly computation cost (C_{cpm}) and storage cost (C_{spm} can be shown in Equations (1) and (2).

$$C_{cpm} = \frac{C_{server}}{life\ time \times 12} + \frac{C_{colocation}}{month} + C_{fts} \quad (1)$$

A storage server is a server that provides high volumn storage, including harddisk drive (HDD), solid-state drive (SSD), or network-attached storage (NAS). Therefore, the storage cost (C_{spm}) includes storage costs (C_{HDD}, C_{SSD}, and C_{NAS}) and a base server cost.

$$\begin{aligned} C_{spm} = & \frac{C_{server}}{life\ time \times 12} + \frac{C_{colocation}}{month} + C_{fts} \\ & + \frac{C_{HDD} + C_{SSD} + C_{NAS}}{life\ time \times 12} \end{aligned} \quad (2)$$

In the market, servers have various specifications. For example, a decent server specification of Intel Xeon E3-1220 3.1GHz, 8 GB memory, 500 GB HDD, 3-year warranty, costs about 1359 USD [11]. Regarding this server specification, the price has a 3-year warranty, we then take three years as the hardware lifetime. If the matchine lasts longer than three years, the rest income will be the profit. The other cost is a colocation service charge of which range is various in the market. It depends on the specification such as network bandwidth, IP address, redundant power wire, etc. The colocation cost is about 75 USD per month [12]. We get the physical server cost about 112.75 USD per month for C_{cpm}. For an addition storage of HDD 7200 RPM serial ATA, 6 TB, 2-years warranty, the cost of C_{spm} is about 126.50 USD per month.

2) Virtual Machine Cost: The VM cost (C_{VM}) is a service charge when VSaaS providers deploy their system using VMs. Normally, the providers rent VMs from a virtual private server (VPS) provider, and then deploy a VSaaS system on the VMs. The VPS providers periodically charge customers (monthly, quaterly, semiannually, or annually) according to VM specification. We can calculate the cost as shown in Equation (3).

$$C_{cpm} = C_{spm} = \frac{C_{VM}}{month} \tag{3}$$

For instance, the cost of renting a VM with 4-core CPU, 8 GB memory, 80 GB SSD disk is 80 USD per month [13]. Then, C_{cpm} and C_{spm} cost about 80 USD per month. The advantage of using virtual machines is that the VSaaS provider does not have to buy any hardware. Also, the provider incurs no risks from hardware failure and maintenance costs.

3) Cloud Computing Cost: There are several kinds of cloud computing services for different proposes. In the VSaaS paradigm, the system applies computing and storage services as a minimal requirment. Two services are rented with different costs. Therefore, we separate the cloud computing cost into two kinds: computing rental cost ($C_{compute}$) and storage rental cost ($C_{storage}$). The two costs have different service charging schemes according to the cloud providers' pricing. Therefore, Equations (4) and (5) present different costs comparing to previous cases.

$$C_{cpm} = C_{compute} \tag{4}$$

$$C_{spm} = C_{storage} \tag{5}$$

In the cloud computing market, there are many service packages for providing cloud services to customers. In this section, we survey a package that is similar to the physical server and virtual machine costs. For computing instance, a system of 4-core CPU, 15 GB memory, 80 GBSSD disk costs 0.28 USD per hour [14] or 146 USD per month (no upfront). For storage, the cost is about 0.03 USD per GB per month [15]. If we choose the storage cost of 1000 GB per month, then the storage cost is about 30 USD per month. Therefore, C_{cpm} can be summed up to 146 USD per month and C_{spm} is about 30 USD per month.

B. Monthly Cost

Apart from the infrastructure cost, the VSaaS provider may have to spend for another additional cost per month (C_{apm}) which comes from, for example, salaries, various equipment, and accessory. Moreover, some providers may have to get a loan for starting up their company. As a consequence, the monthly cost also includes the interest per month (I_m). In summary, the monthly cost (C_m) can be summed up as shown in Equation (6).

$$C_m = \sum_{i=1}^{\alpha} C_{cpm_i} + \sum_{j=0}^{\beta} C_{spm_j} + C_{apm} + I_m \tag{6}$$

$$\text{where} \quad \alpha \text{ is number of computing servers}$$
$$\beta \text{ is number of storage servers}$$

According to Equation (6), the VSaaS provider has a monthly cost including the summation of computing server costs from 1 to α. However, a single computing server can be deployed as a VSaaS system, and the storage can be combined in the computing servers storage. The summation of costs of storage servers will appear in the equation when separate storage servers are used. If the provider deploys their VSaaS system on a single server, this term will disapear. However, in image analysis the monthly cost term is difficult to calculate resource usage and video record storage, because we cannot easily separate the terms between computation and storage space. Therefore, the monthly cost can be reformed into C_{cpm} and C_{spm} by dividing the C_{apm} and I_m with α and β as shown in C_{epm}, in Equation (7).

$$\begin{aligned} C_{epm} &= \frac{C_{apm} + I_m}{\alpha + \beta} \\ C_{ncpm} &= C_{cpm} + C_{epm} \\ C_{nspm} &= C_{spm} + C_{epm} \end{aligned} \tag{7}$$

Equation (7) presents new computational cost (C_{ncpm}) and storage cost (C_{nspm}) per month. However, different providers deal with different additional costs and interests. Therefore, this paper only focuses on determining the infrastructure cost ($C_{epm} = 0$) for simplification.

IV. Real Load VSaaS Resource Usage

This section shows the resource usage for video analysis and video recording using a Nokkhum image processor. The experiment in this section is conducted for cost estimation, using the capacity of one camera on one machine. Furthermore, this criteria is used for caculating the resource-usage pricing model proposal in Section V-B.

A. CPU and Memory Usage

Serveral workload types occur while a VSaaS is working. For example, VSaaS middleware workload, database workload, and image analysis workload. The highest resource usage among the mentioned workloads is that of image analysis, especially when an image of a big size is processed as shown in Fig. 3 to 8. Video encoding used in the experiments is Theora [16]. These experiments were tested on AMD A10-5800K Trinity Quad-Core 3.8 GHz, Memory 16 GB. Fig. 3

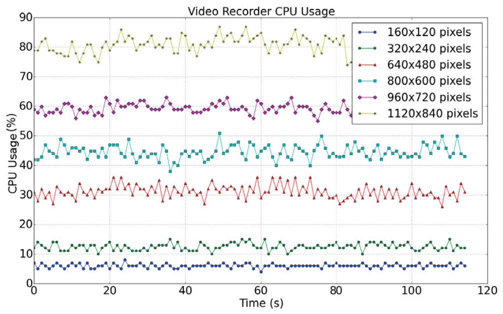


Fig. 3. Video recorder CPU usage using the frame rate of 10 FPS

and 4 show the results of CPU and memory usages in video recording when using various image sizes and the frame rate of 10 frames per second (FPS). The graphs of CPU and memory usages of all image sizes are quite linear, and depend on the video codec used. Fig. 5 and 6 show resource usages when

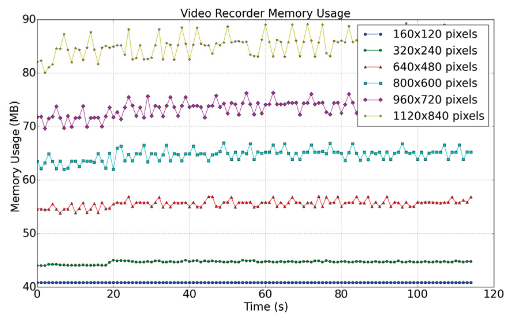


Fig. 4. Video recorder memory usage using the frame rate of 10 FPS

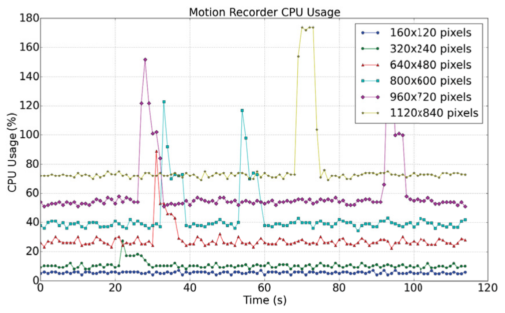


Fig. 5. Motion recorder CPU usage using the frame rate of 10 FPS

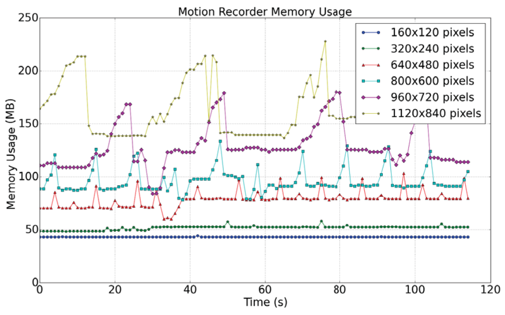


Fig. 6. Motion recorder memory usage using the frame rate of 10 FPS

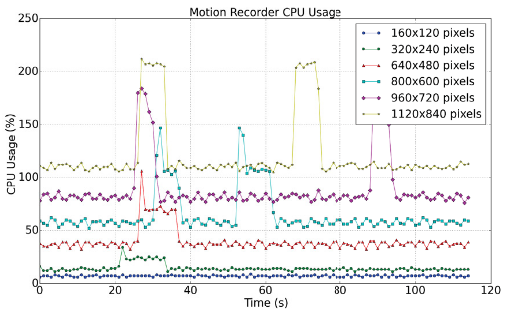


Fig. 7. Motion recorder CPU usage using the frame rate of 15 FPS

combining motion detection and video recording. Resource usages in Fig. 5 and 6 have a different characteristic from those of Fig. 3 and 4, as they appear in different periods. The

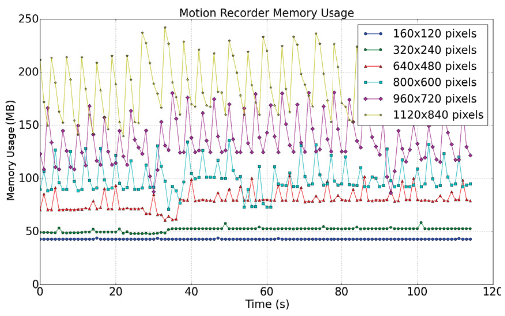


Fig. 8. Motion recorder memory usage using the frame rate of 15 FPS

shoots occur when the motion detector detects a sequence of motion, then it passes the sequence to the video recorder. This characteristic generally appears in results when using various frame rates. It is also seen in the result when using the frame rate of 15 FPS as shown in Fig. 7 and 8.

B. *Storage Space*

Storage space is an important part of VSaaS which stores video records. The size of storage space has to be sufficient to store up to the customers' requirement. The size of a video record grows following the video frame rate, video image size, and video encoder types. Table II shows video record sizes of images of 800x600 pixels with the frame rate of 10 FPS used in real situations.

TABLE II. EXAMPLES OF VIDEO RECORD SIZES USED IN REAL SITUATIONS

Type of image analysis	**FPS**	**Min**	**Max**	**Storing period (day)**		
				1	**7**	**30**
Video Recorder	10	82.1 M	199.1 M	19 G	133 G	570 G
Video Recorder	15	92.9 M	209.8 M	32 G	224 G	960 G
Motion Recorder	10	1 M	12.5 M	2.3 G	16.1 G	69 G
Motion Recorder	15	1.3 M	26 M	2.9 G	20.3 G	87 G

V. Cost Estimation and Pricing

Pricing models appear in the current market where there are providers lists in Table I. The least unit of measurment in the market is per camera per day but the invoice is normally quoted once per month. Another package charges the user per camera per month, per quarter, and per year. This section presents a pricing model for VSaaS revealed in the market, and infrastructure cost approximation in details.

A. *Package Model*

A package model is well known in the market and it is widely used in the current pricing. The provider offers a package per camera that the invoice is issued monthly, quarterly, or annually. In general, cost per quarter and per year are cheaper respectively as the payment is done upfront. In this section, we present the cost estimation of two package models that may be useful for VSaaS providers.

1) Cost Analysis: This section presents a computational cost for image analysis that can be divided into two situations. Image analysis heavily comsumes CPU more than memory, and the storage usage corresponds to the frame rate used and the video or image size. However, image analysis comsumes a lot of memory when dealing with a large image size and a fast frame rate. The computatonal cost used for a camera is shown in Equation (13).

$$C_{ccam_{s,f}} = \begin{cases} \dfrac{C_{ncpm}}{\lfloor \dfrac{N_{CPU} \times 100}{\%\ CPU_{s,f}} \rfloor}, & \text{if interested in CPU usage} \\ \dfrac{C_{ncpm}}{\lfloor \dfrac{Memory_{MAX}}{Memory_{s,f}} \rfloor}, & \text{if a big image size is used} \end{cases} \quad (8)$$

Equation (13) shows a computational cost per camera with video size and frame rate ($C_{ccam_{s,f}}$), which depends on the image size and frame rate. The C_{ncpm} is divided by the number of CPUs divided by % CPU usage of image analysis ($\lfloor \frac{N_{CPU} \times 100}{\%\ CPU_{s,f}} \rfloor$). According to the physical server cost as shown in Section III-A1 and resource usage of image analysis in Section IV-A, the monthly computational cost is about 112.75 USD, when using a camera with the image size of 800x600 pixels, 10 FPS, and simple video recording. The average CPU usage is 46.2%. Therefore, we get the $C_{ccam_{800x600,10}}$ as about 14.09 USD ($112.75/\lfloor \frac{4 \times 100}{46.2} \rfloor$).

$$C_{spd} = \frac{C_{nspm}}{capacity \times 30} \quad (9)$$

$$C_{scam_{s,f}} = C_{spd} \times S_{d_{s,f}} \times day \quad (10)$$

For storage space, we normalize the C_{nspm} with the available space capacity, into a number of giga-bytes per day (C_{spd}) for flexibly estimating the cost in different periods as shown in Equation (9). Therefore, the cost for storing video records per one camera with specific video size and frame rate ($C_{scam_{s,f}}$) can be calculated by the cost for storing a video record multiplied by the maximum storage size ($S_{d_{s,f}}$) and the number of days the storage is kept as shown in Equation (10). For example, according to the physical server cost shown in Section III-A1 and the storage space usage for video record described in Section IV-B, the monthly storage server cost is about 126.50 USD, when using a camera with the image size of 800x600 pixels, 10 FPS, and simple video recording. The average storage size for a camera is about 570 GB per month (storing 570 GB per day). We get the cost per GB per day about 0.0007 USD ($126.50/(6000 \times 30)$), and the $C_{scam_{s,f}}$ for 30 days is about 11.97 USD ($570 \times 30 \times 0.0007$). Then, we conclude the cost per camera per month about 26.06 USD per month (14.09+11.97).

$$C_{cam_{s,f}} = C_{ccam_{s,f}} + C_{scam_{s,f}} \quad (11)$$

A VSaaS provider can apply Equation (10) for different storing periods. In the matket, VSaaS providers offer common periods for pricing starting from 7 to 30 days. However, customers may want to keep their video records longer in some sensitive areas or situation.

2) Pricing: Monthly pricing for VSaaS is like another issue as the provider needs to manipulate both costs and excepted profit (ρ). Regarding the monthly pricing model shown in Equation (12), VSaaS providers can acquire for as much profit as they want. However, pricing must also relate to the competitors' prices. In Equation (12), price per camera($P_{cam_{s,f}}$) depends on the video frame rate and size. However, it is difficult to exactly determine prices for all kinds of providers due to different causes such as system reliability, quality of service, image analysis quality in each system.

$$P_{cam_{s,f}} = C_{ccam_{s,f}} + C_{scam_{s,f}} + \rho \quad (12)$$

B. Pay-as-use Model

This section presents an ondemand resource usage model for image analysis with different storage periods. In the maket, VSaaS providers provide packages for hosting video analysis and storing video records. The minimum period they generally offer price is a day. In some situations, VSaaS systems may fail, for example, the network connectivity is broken, IP cameras are not working, or customers can not fully utilize their rental resources. A pay-as-use model that breaks the price into small details seems fair and worthy for customers.

1) Cost Analysis: According to the infrastructure cost in Section III-A, we break down the monthly computational cost into smaller periods such as week, day, hour, minute, as shown in Table III. The storage space cost in Equation (9) is suitable for determining in this analysis. Table III shows the same costs broken down in various periods. Each period cost (C_{period}) is divided by % CPU usage of image analysis. In Equation (13), the cost in each period comes from the maximum of CPU usage $Max(\%\ CPU_{s,f_{sp}})$ reported by the image analysis sensor.

TABLE III. RESOURCE USAGE COSTS

Server Type	Month (C_{ncpm})	Cost (USD)			
		Week	Day	Hour	Minute
physical server	112.75	26.3083	3.7583	0.1392	0.0023
VM	80	18.667	2.6667	0.0988	0.0016
cloud	146	34.0667	4.8667	0.1802	0.0030

$$C_{ccam_{s,f}} = \frac{C_{minute}}{N_{CPU} \times 100} \times Max(\%\ CPU_{s,f_{sp}}) \quad (13)$$

Suppose that a VSaaS provider deploys a VSaaS system on a physical server. A customer applies a motion video recorder with a 800x600 pixel and 10 FPS 46.2% CPU usage. It is about 0.0002 USD ($\frac{0.0023}{4 \times 100} \times 46.2$) per one minute. If the CPU usage significantly differs from the previous measurement all the time, the cost can also different every minute.

2) Pricing: Pricing for a pay-as-use model is shown in Equation (14). It is like Equation (12), but the first part is the summation of $C_{ccam_{s,f}}$ calculated every minute. The provider can identify static profit (ρ) for every camera. However, different video sizes lead to different CPU usages in image analysis. The provider can basically set for a higher profit when providing a larger video size.

$$P_{cam_{s,f}} = \sum_{t=0}^{\tau} C_{ccam_{s,f_i}} + C_{scam_{s,f}} + \rho \quad (14)$$

VI. Discussion

Two previous pricing models are suitable for some specific situations. A monthly package model is easy for calculation and starting up a small VSaaS business. Providers can provide pricing according to the video size or frame rate. For instance, the provider may offer a service with the video size of 800x600 pixels and the frame rate of 10 FPS. In this case, customers can apply for video analysis equaling to or smaller than 800x600 pixels with the frame rate of 10 FPS. Also, they can ask for 800x600 pixels with 5 FPS, or 320x240 pixels with 10 FPS. They can reduce their income risks, when the VSaaS system is not fully utilized. Example costs for a video recorder with 10 FPS, 800x600 pixels is shown in Table IV. Table IV shows

TABLE IV. Examples of monthly package cost per camera

Server type	Monthly cost	Monthly cost per camera	Storage per GB	GB per day	Storing period (day)		
					7	30	60
physical server	112.75	14.09	0.0007	19	14.74	26.06	61.97
VM	80	10.00	0.03	19	37.93	523.00	2062.00
cloud	146	18.25	0.03	19	46.76	531.25	2070.25

that deploying VSaaS on a physical server seems cheaper than using other platforms. However, in this paper, we cut off additional costs from the infrastructure cost. In the VM cost, cloud storage is usually used as an additional storage backend, because a rental VM normally has a very low capacity of storage space. The cloud computing cost seems the highest in deploying VSaaS, but it guarantees highly available quality of service.

The experimental costs in the pay-as-use model is shown in Table V. The cost per camera per month in Table V is cheaper than the monthly package cost model, because it is a summation of costs per minute. Therefore, VSaaS providers always have to manage their systems to achieve full utilization, otherwise they will losse their profit. According to Equation (14), a provider can increase the profit part for remuneration before pricing. Tables IV and V show that the hightest cost

TABLE V. Examples of pay-as-use costs per camera

Server type	Cost per minute	Monthly cost per camera	Storage per GB	GB per day	Storing period (day)		
					7	30	60
physical server	0.0023	11.479	0.0007	19	12.13	23.45	59.36
VM	0.0017	8.485	0.03	19	36.42	521.49	2060.49
cloud	0.0030	14.973	0.03	19	42.9	527.97	2066.97

of VSaaS system is from storage space, because VSaaS keeps a lot of video records for long periods. If VSaaS providers want to reduce this cost, they have to add for higher profit or discount service package charge.

VII. Conclusion

This paper presents cost investigation for deploying a VSaaS system on various platforms and pricing models for providers. The providers can use the models as a guide line for estimating their costs and pricing service charges. We propose serveral pricing models for various situations suitable for providers with different business sizes based on a real load investigation.

However, the cost estimation and pricing models appearing in this paper do not include the network characteristics and additional costs for VSaaS. We will include them into our analytical models in the future work.

Acknowledgment

The authors are grateful for financial supports from the Thailand Research Fund through the Royal Golden Jubilee Ph.D. Program (Grant No. PHD/0047/2552) and Prince of Songkla University.

References

[1] A. Prati, R. Vezzani, M. Fornaciari, and R. Cucchiara, "Intelligent video surveillance as a service," in *Intelligent Multimedia Surveillance*, P. K. Atrey, M. S. Kankanhalli, and A. Cavallaro, Eds. Springer Berlin Heidelberg, 2013, pp. 1–16. [Online]. Available: http://dx.doi.org/10.1007/978-3-642-41512-8_1

[2] T. White, *Hadoop: The Definitive Guide*, 1st ed. O'Reilly Media, Inc., 2009.

[3] Y.-S. Wu, Y.-S. Chang, T.-Y. Juang, and J.-S. Yen, "An architecture for video surveillance service based on P2P and cloud computing," in *Proceedings of the 2012 9th International Conference on Ubiquitous Intelligence Computing and 9th International Conference on Autonomic Trusted Computing (UIC/ATC)*, Sep. 2012, pp. 661–666.

[4] J. Lee, T. Feng, W. Shi, A. Bedagkar-Gala, S. Shah, and H. Yoshida, "Towards quality aware collaborative video analytic cloud," in *2012 IEEE 5th International Conference on Cloud Computing (CLOUD)*, 2012, pp. 147–154.

[5] C.-F. Lin, S.-M. Yuan, M.-C. Leu, and C.-T. Tsai, "A framework for scalable cloud video recorder system in surveillance environment," in *Proceedings of the 2012 9th International Conference on Ubiquitous Intelligence Computing and 9th International Conference on Autonomic Trusted Computing (UIC/ATC)*, 2012, pp. 655–660.

[6] Amazon, "AWS documentation," 2014. [Online]. Available: http://aws.amazon.com/documentation/

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

[8] D. Rodriguez-Silva, L. Adkinson-Orellana, F. Gonz'lez-Castano, I. Armino-Franco, and D. Gonz'lez-Martinez, "Video surveillance based on cloud storage," in *Proceedings of the 2012 IEEE 5th International Conference on Cloud Computing (CLOUD)*, 2012, pp. 991–992.

[9] T. Limna and P. Tandayya, "A flexible and scalable component-based system architecture for video surveillance as a service, running on infrastructure as a service," *Multimedia Tools and Applications*, pp. 1–27, 2014. [Online]. Available: http://dx.doi.org/10.1007/s11042-014-2373-8

[10] R. T. Fielding and R. N. Taylor, "Principled design of the modern web architecture," *ACM Transaction on Internet Technol.*, vol. 2, no. 2, pp. 115–150, May 2002. [Online]. Available: http://doi.acm.org/10.1145/514183.514185

[11] Dell, "Poweredge r220 rack server," 2015. [Online]. Available: http://www.dell.com/us/business/p/poweredge-r220/fs

[12] Mosaic Data Services, "Colocation pricing," 2015. [Online]. Available: https://mosaicdataservices.com/colocation/colocation-pricing/

[13] DigitalOcean, "Simple pricing," 2015. [Online]. Available: https://www.digitalocean.com/pricing/

[14] Amazon Web Services, "Amazon ec2 pricing," 2015. [Online]. Available: http://aws.amazon.com/ec2/pricing/

[15] ——, "Amazon s3 pricing," 2015. [Online]. Available: http://aws.amazon.com/s3/pricing/

[16] Theora, "Theora video compression," 2015. [Online]. Available: http://www.theora.org/

B.2 Journals

B.2.1 T. Limna and P. Tandayya, “A flexible and scalable component-based system architecture for video surveillance as a service, running on infrastructure as a service,” *Multimedia Tools and Application*, vol. 75, no. 4, pp. 1765–1791, Feb. 2016. WoS. doi:10.1007/s11042- 014-2373-8

B.2.2 T. Limna and P. Tandayya, “Workload Scheduling for Nokkhum Video Surveillance as a Service,” *Multimedia Tools and Application*, pp. 1–27, Jan. 2017. WoS. doi:10.1007/s11042-016-4225-1

Multimed Tools Appl (2016) 75:1765–1791
DOI 10.1007/s11042-014-2373-8

A flexible and scalable component-based system architecture for video surveillance as a service, running on infrastructure as a service

Thanathip Limna · Pichaya Tandayya

Received: 25 April 2014 / Revised: 23 September 2014 / Accepted: 7 November 2014 /
Published online: 31 December 2014

Abstract There are many proposals for moving traditional video surveillance systems into the cloud, commonly known as Video Surveillance as a Service (VSaaS). Most systems use Hadoop technology for storing video records and distributing video analysis tasks. However, Hadoop is more appropriate for video retrieval services than real time video analysis. Also, existing systems offer neither flexible deployment plans, nor are they capable of automatically minimizing the number of required servers (whether they are physical or virtual machines). Our proposal involves the design and implementation of a component-based VSaaS running on Infrastructure as a Service (IaaS). This paper focuses on the design concepts and component functions that provide solutions for the availability and scalability of VSaaS. Our system can easily scale from one server up to a more complex cluster to support the varying requirements of users. It accesses cloud services via Amazon EC2 for computing services and Amazon S3 API for object storage services, since they are supported by many cloud computing IaaS providers. We also present a components deployment that is suitable for any size and type of system, which combines both physical and virtual machines. Experiments show that the system performs well, and can tolerate difficult scenarios.

Keywords Video surveillance as a service · Infrastructure as a service · Cloud computing · System design architecture · Flexibility · Scalability · Fault toleration

1 Introduction

Video surveillance systems have become a standard security tool, preventing unpleasant events from occurring and reducing damage when unusual events do occur. Video

Financial support from the Thailand Research Fund and Prince of Songkla University through the Royal Golden Jubilee Ph.D. Program (Grant No. PHD/0047/2552).

T. Limna · P. Tandayya (✉)
Department of Computer Engineering, Faculty of Engineering, Prince of Songkla University, Hat Yai, Songkhla, Thailand
e-mail: pichaya@coe.psu.ac.th

T. Limna
e-mail: thanathip.limna@gmail.com

surveillance systems are widely used [34], including in residential areas, offices, factories, and traffic control systems. The cameras and related equipment are inexpensive and easily installed. However, video surveillance systems costs involve more than just hardware, including software installation, configuration, operation, and maintanance (to validate and verify system availability). Good preparation and administration are essential for running a video surveillance system.

Video surveillance applications on the Internet generally support online video recorders, surveillance image processors, and online video storage. Cloud-based video surveillance systems, known as Video Surveillance as a Service (VSaaS) are growing in popularity. The main advantage of VSaaS is that the users can be less concerned about system software/hardware maintenance, instead focusing on providing Internet Protocol (IP) cameras. Also, users can choose new image processing solutions as soon as the provider deploys them on the system, without system reinstallation. In addition, provider competition reduces operation cost and provides best-quality image processing for users. Most current VSaaSs provide simple image processing, such as motion detection for specific areas of interest, storage space, and an alert system. Unfortunately, there is a lack of information about the technologies and VSaaS architectures due to trade secrets, and some open video surveillance systems are unsuitable for Software as a Service (SaaS) models, because they are designed to support specific applications. These include car tracking in parking areas and trafic monitoring, which may not be applicable to home or residential areas. Moreover, it is difficult to modify some types of software to run in a distributed manner to support SaaS due to the close cooperative work among the software's components.

A recent proposal suggested the use of a cloud-based video surveillance system built with Hadoop [39]. A Hadoop MapReduce distributes image processing tasks and employs the Hadoop Distributed File System (HDFS) for storing video records. Examples include: P2PCloud [41], VAQACI [16], and the CVR system [18]. Another approach uses cloud object storage via Amazon S3 [1] for storing video records, as described by Hossain *et al.* [10] and Rodriguez-Silva et al. [29]. However, this research did not reveal deployment information, such as how components are placed, how to scale the computing units, the minimum number of servers that can deploy the system, front-end component arrangement for user access, and the provision of the video/camera management services. Neither groups exploited the benefit of the full capability of computing units on both Virtual Machine (VM) and object storage; they either employ one type or the other. As a consequence, their deployment can be rather expensive and complex.

We have developed a VSaaS which works on IaaS, focusing on software system architecture, system behaviors, and user requirements. The system's architecture is based on traditional cloud services: the compute service employs Amazon EC2 and the object storage service utilizes the Amazon S3 API. Our scalable VSaaS software system architecture can easily add new computation node workers to support increasing numbers of IP cameras, and also saves time and energy in planning and controlling system performance.

2 Video surveillance system

Video surveillance systems currently use two types of cameras: closed-circuit television (CCTV) and IP cameras. Many CCTV systems do not only record video, but can also perform functions such as motion detection and record motion video footage. However, CCTV systems have limitations concerning camera usage distance, storage capacity, and installation and reconfiguration problems when adding new processing algorithms. IP camera systems avoid these limitations with longer working distances and wider areas. They can also be easily plugged into any computer network and so distribute data across the Internet. IP camera systems normally employ computers capable of video processing and recording, allowing the systems to be scaled up to support many IP Cameras and offer new image processing features. Currently, video surveillance systems are moving away from standalone applications toward software as a service, in order to provide video processing to a larger number of users. In this section, we compare traditional and modern video surveillance which motivate the improvements proposed in our design.

2.1 Traditional video surveillance systems

Traditional video surveillance systems supporting IP cameras [7] were often connected to a Network Video Recorder (NVR) or a generic computer to provide Video Content Analysis (VCA). In smaller systems, the video camera management and VCA software were usually bundled with IP cameras. The user had to install software on a computer which allowed it to connect to the IP cameras. Normally, the vender software was limited to their own type of cameras and could not interface with cameras from different venders. High performance software that could manage a larger number of cameras and provide special VCA was usually sold separately as additional software. Usually, this kind of software was very expensive and required complex configuration. Anyone wanting to set up a video surveillance system, would incur many expenses apart from the cost of the IP cameras, including the cost of first time installation, network equipment, maintenance and additional computing software.

There are many free and open source video surveillance packages that run on standalone computers, such as ODViS [13], OpenVSS [32], and Zoneminder [33], which provide video camera management, VCA, and storage. Most packages are easy to install, configure, and use, but their software components are rather atomic and hard to scale. One solution is to add a new computing node and re-configure the system. However, it becomes harder to manage and maintain the system, as the number of cameras and different VCA configurations increases.

The other type of video surveillance system architecture is distributed, utilizing a software architecture consisting of many components and processing layers, such as IVSS [42], DiVA [30], CANDELA [40], and S-VDS [9]. These distributed architectures are designed to support massive numbers of cameras and configurations. The components can be distributed across many computers to provide higher performance VCA and system availability. However, the architecture must be carefully configurated as it includes different modules that run on separate machines, and requires tightly coupled work cooperation. Moreover, these kinds of systems require a specialist to determine which components need to be scaled up.

Exploiting the Internet, some video surveillance software provides online system access for video camera monitoring, video storage, and software configuration. Examples include ViSOR [35], S-VDS [9], and MoSES [8]. Although, this makes it easier to access the system, users must budget a large amount for hardware, software, and service charges.

2.2 Video surveillance as a service

Online video storage enhances traditional video surveillance systems by increasing user convenience, and has influenced research on remote video surveillance recorders and VCA on the Internet [11]. One advantage of this architecture is that it can serve any IP camera from any location to users anywhere across the Internet. Users do not have to own recording and storage hardware, instead renting an NVR or a VCA from a video surveillance provider. The provider offers an Internet based recorder and storage space via a service package which matches the user's requirements. However, it is hard for the video surveillance provider to predict the required supply of computing and storage hardware for on-demand processing.

A new video surveillance platform has emerged from cloud computing technology, especially Infrastructure as a Service (IaaS) such as Amazon Web Service (AWS) [1], Hadoop, and virtualization technology. It provides more system availability, security, reliability, and maintainability for cloud-based video surveillance than traditional software [14]. Cloud technology changes remote video surveillance into Video Surveillance as a Service (VSaaS) which is a Software as a Service (SaaS) model. This kind of architecture avoids limitations of computing resources, such as storage space, computing unit, and network management. Cloud computing enables a remote video surveillance provider to start with a small video surveillance system which can grow according to the users' demands. Consequently, the provider does not need a high budget for starting the business and does not need a complex hardware supply plan for supporting user requirements. Current VSaaS in the market provide an NVR, a simple VCA, and storage space for collected video records. The VSaaS providers offer various service plans which their customers choose based on their requirements. Examples of VSaaS currently on the market include SecurityStation [25], Secure-i Hosted Video Recorder [31], and OVS [23].

Many cloud-based video surveillance systems can be classified into two groups by the cloud technology they use. The first group implements their video surveillance architecture based on Hadoop technology which includes MapReduce and the Hadoop Distributed File System (HDFS). Example systems in this group are P2PCloud [41], VAQACI [16], and CVR systems [18]. They utilize the HDFS to store video records and to provide high performance video retrieval. In addition, MapReduce can automatically distribute processing tasks to Hadoop computing nodes. This ability means that a massive number of video records can be distributed as VCA tasks to Hadoop nodes.

The second group implements IaaS by employing Amazon AWS as a video surveillance tool. Examples in this group include the systems developed by Hossain et al. [10] and Rodriguez-Silva et al. [29]. Normally, this kind of architecture uses the Amazon Simple Storage Service (Amazon S3) to collect video records and the Amazon Elastic Compute Cloud (Amazon EC2) to provide virtual machines as computing units.

Both groups of architectures easily transform the task of video surveillance work in cloud-based technology because the software does not depend on any development framework. However, the resulting architecture design in [10] and [29] is rather general. For example, it is unclear how issues such as the compute node number for a first-time installation, or component distribution during deployment and system scalability, are handled.

2.3 Nokkhum component-based system

We propose a component-based video surveillance architecture for VSaaS called the Nokkhum system [17], a flexible design that treats video surveillance as a service. Nokkhum includes components for providing a VSaaS on physical servers, which can be ported to IaaS using the Amazon S3 API for image and video record storage. Nokkhum includes virtual machine management, but previously it only used a web interface for providing user access. Although, this made it easy to use via a web browser, it was difficult to add support for other types of clients, such as mobile devices and tablets. For instance, some mobile operating systems need to execute a native application in order to offer a good user experience and performance. Therefore, the current Nokkhum system can deploy components on a hybrid system of physical servers and virtual machines via the Amazon EC2 API, engendering system portability so that the VSaaS system is suitable for different business sizes. In addition, our API server supports interfaces from any client by utilizing a REpresentational State Transfer (REST) style architecture via HTTP [6]. In summary, we propose a flexible and scalable system architecture, and components design and implementation, for providing VSaaS on IaaS.

3 Nokkhum video surveillance as a service system

The “Nokkhum” (a Thai word for “Asian quail”) VSaaS consists of five components (see Fig. 1): a controller, a compute node worker, an image processor, an Application Program Interface (API), and a client. VSaaS users can access and control their surveillace processes via a web interface connected to the API server. The controller is a daemon process that handles user requests from an API server and directs commands to available compute nodes using the message-oriented middleware [19]. The compute node worker receives

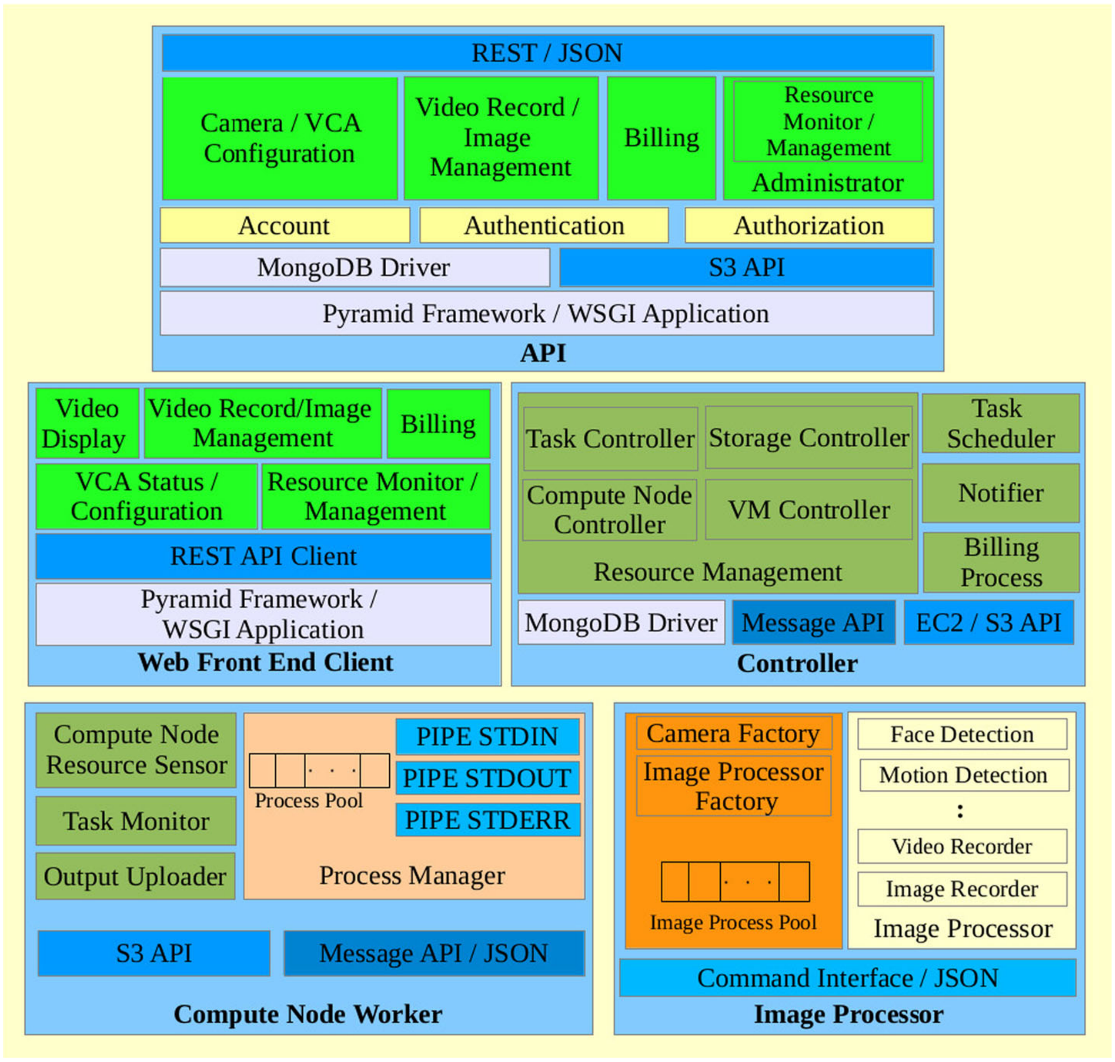


Fig. 1 All components and modules of the Nokkhum VSaaS

commands from the controller via a message broker, manages its image processors, monitors its resources, and reports status details back to the controller. In this work, our image processors are implemented using OpenCV [3].

3.1 Nokkhum design concept

According to the Video Surveillance as a Service (VSaaS) paradigm, a provider offers VCA and storage service to its users. Users must register with the provider and can then use the services provided by the VSaaS system on the Internet. The provider must prepare the VSaaS system and provides services which are available to users at all times. The system must handle dynamic user requests and start required image processing processes when appropriate. Many video surveillance systems are designed to support a single organization with groups of users, but VSaaS tends to be more complex, including various organizations and groups.

The benefits of cloud computing for system availability, dynamic resources provision, and pay as use, enable providers to offer their software at a low cost. Also, they can dynamically scale the system resources according to user requirements. The service platform approach saves on both the software and hardware maintainance costs by offering services with a small software solution at first, but allowing a smooth transition to bigger packages later. The Nokkhum architecture is a VSaaS that runs on top of a cloud computing infrastructure. Both the system size and the business size can be easily scaled to match the customer's requirements. Moreover, the architecture is designed to run on physical machines if necessary, in order to offer the best image processing performance for a small VSaaS system.

3.1.1 Scalability design

The Nokkhum VSaaS architecture is designed for scalability by using the following assumptions:

1. The video surveillance system runs on a cloud infrastructure as a service.
2. The architecture automatically acquires virtual machines to support the customer's changing requirements.
3. The VSaaS provides a private cloud for an organization and a public cloud for user groups.
4. The VSaaS fully supports IaaS with the Amazon EC2 and Amazon S3 APIs, and is compatible with a system that uses only a physical server.
5. The VSaaS starts as a small system, but can easily scale up to match dynamic user requirements.
6. The design aims for a system that utilizes a broadband network throughout.

Following the above assumptions, Nokkhum VSaaS is a highly scalable architecture composed of several components. A controller manages system resources and task scheduling for passing image processing tasks to compute nodes. Consequently, each compute node includes a system resource reporter and an image processing task starter. Message-oriented middleware [37] is employed to exchange information between the main distributed components, and allows the components to be distributed across different computing platforms (both generic personal computers and virtual machines).

3.1.2 Flexibility design

The message-oriented middleware increases system availability, because if some components crash, the other components in the system can still continue running. Nokkhum VSaaS is designed to flexibly tolerate difficult scenarios such as:

1. When the electricity is cut off, or the network is out of service at a video camera site, the system can automatically recover after the camera becomes available again.
2. If a compute node worker has a problem, such as the network being out of service, or a program crashes or a message broker disconnects, then the system can provide an alternative compute node worker.
3. If the Nokkhum controller does not respond, but the compute node workers are still running, then the video processors can continue working. When the controller resumes, it can recover its previous state by processing the information collected in the message broker server and database server.
4. If the message broker server does not respond, then the compute node worker and the controller can continue running with the last available information. After the message broker server resumes, the compute node worker and the controller will start commuicating again.
5. The system can start on a single machine and scale up by adding worker machines.

3.2 Nokkhum components

Nokkhum consists of five components developed with C++ and Python as shown in Fig. 1. Each offers specific functions and work cooperation for enabling system availability.

As shown in Figs. 2 and 3, the user initially creates a camera configuration and image processing solution on a Nokkhum client, which is then passed to a Nokkhum API that stores the command in a database. Once the task scheduler on the Nokkhum controller becomes active, it receives the user's command. The compute node resource predictor module on the Nokkhum Controller finds available resources and directs the command to a compute node worker via the message server. When the compute node worker receives the message, it builds a video surveillance solution and starts the related image processors. As the image processors produce output data (video records and/or images), the compute node worker stores the data in cloud storage. The user can watch, download, and delete the video records and images via the Nokkhum client. The controller also monitors and manages storage usage history for video records and images.

The following subsections describe the five components, namely the Nokkhum Controller, Nokkhum Compute Node Worker, Nokkhum Image Processor, Nokkhum API, and Nokkhum client shown in Fig. 1.

3.2.1 Nokkhum controller

The Nokkhum controller is a daemon process made of multiple modules and sub-controllers which perform many tasks. The most important is resource management, which is handled by several sub-controllers:

– The task controller is an image processor which handles VCA tasks via a task monitor and a task manager. The task monitor checks all running image processor tasks. If one

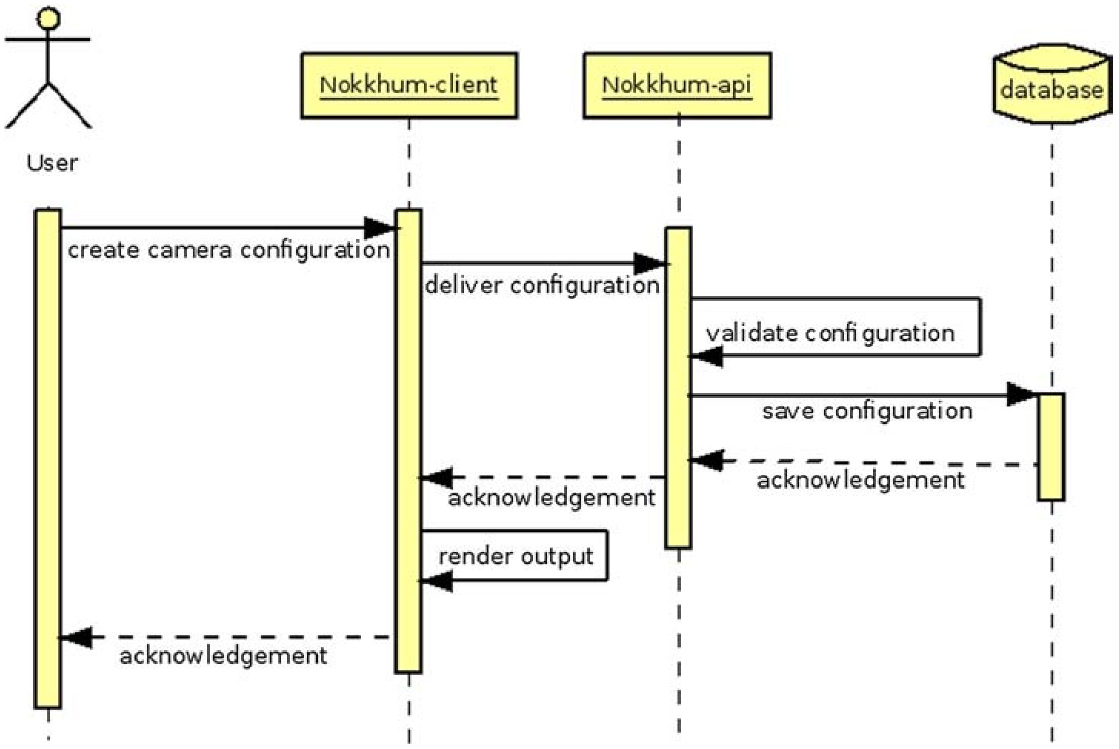


Fig. 2 The VSaaS user configuration sequence

does not respond, then the task monitor will send it a command request to restart the processor task. The task manager provides an interface for controlling image processor activities, and is mainly used by the task scheduler.

– The compute node controller is a compute node worker manager which processes every resource status report. These are delivered from Nokkhum compute node worker sensors via the message broker server. A resource status report includes total CPU usage, total memory usage, and total harddisk usage of a particular compute node worker. It also includes image processing resource usage data, such as CPU usage, memory usage, important image processor results, and the availability status of the image processor. If the resource status report shows that an image processor is unavailable, then the compute node controller will call the task controller to restart the image processor. In addition, when the task scheduler or the VM controller requests a compute node worker's resource information, the compute node controller will provide that information, and predict resource usage for every compute node worker. Currently, this prediction utilizes a Kalman filter [38] applied to the last 20 compute node resource status reports. The compute node controller supports both physical servers and virtual machines.
– The VM controller enables Nokkhum to control virtual machines using the Amazon EC2 API. It acquires a resource prediction from the compute node controller to decide whether to terminate a virtual machine or to start a new one. The VM controller will acquire a new virtual machine when there is a command waiting in the processor command queue and there is no computing resource to handle it.
– The storage controller manages video records and images. Part of its duties are to remove expired video records of any image processor.

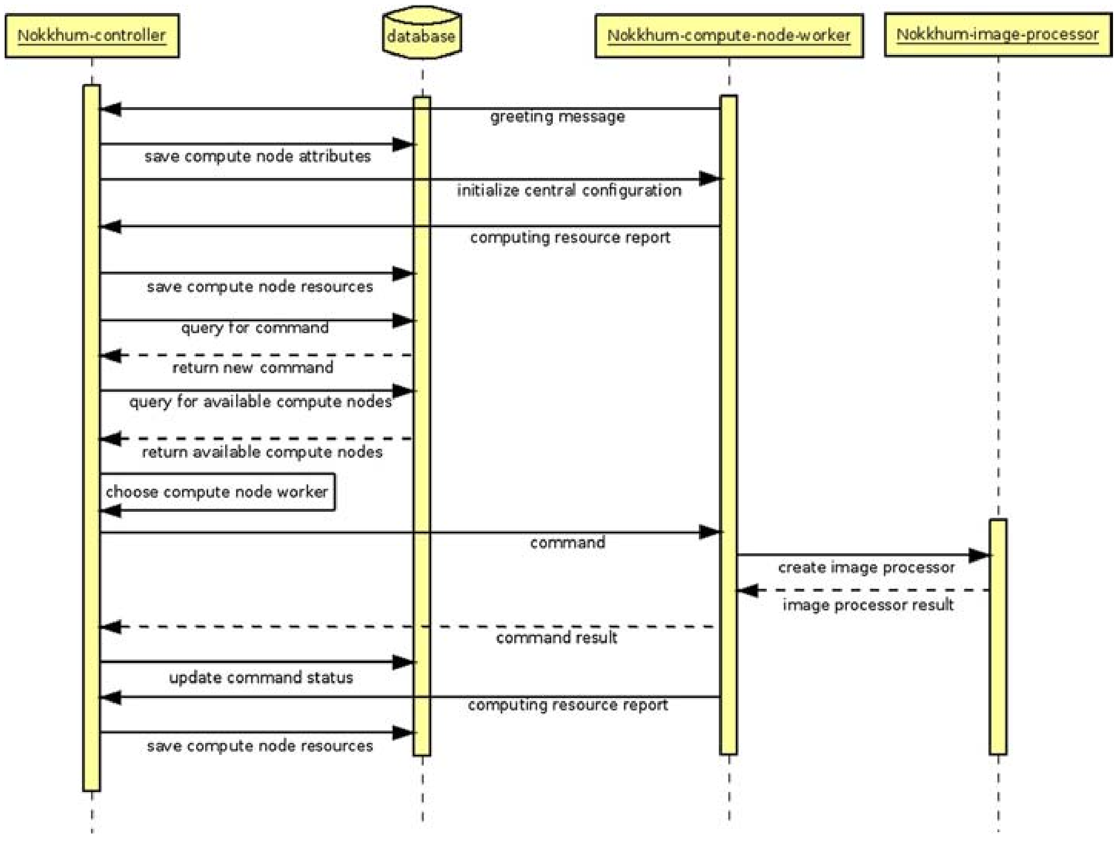

Nokkhum-controller
database
Nokkhum-compute-node-worker
Nokkhum-image-processor
greeting message
save compute node attributes
initialize central configuration
computing resource report
save compute node resources
query for command
return new command
query for available compute nodes
return available compute nodes
choose compute node worker
command
create image processor
image processor result
command result
update command status
computing resource report
save compute node resources

Fig. 3 Interactions among system components of the VSaaS during initialization

Apart from resource management, the Nokkhum controller also includes a task scheduler, a notifier, and a billing process module:

- The task scheduler allocates image processors or VCA tasks to the most suitable Nokkhum compute node workers. It acquires resource information from the compute node controller and starts image processing tasks via the task controller. This module has a command waiting queue that collects processing commands from the user and recovers commands from the task controller.
- The notifier module provides alert messages to users that have activated this module via an image processing task. This module is activated when the compute node controller receives a resource status report that includes a notification message.
- The billing controller handles billing processing for the Nokkhum VSaaS.

The Nokkhum controller deals with the users' video surveillance requirements and calls compute node workers to start surveillance applications via message passing methods. It starts work by retrieving image processing action commands from the database and directs them to available compute node workers. In this way, it can start and stop image processing tasks, and check image processing status details. The controller directs message commands to computer node workers using the message broker server. The compute node workers then build the related image analysis processes.

3.2.2 Nokkhum compute node worker

A compute node worker is a daemon process that runs on each computation node. It provides computer resource monitoring, image processing task monitoring, and an image processing deployment interface. The compute node worker uses PIPEs [27] for communicating with the image processors, and reports information resource updates to the controller using message passing. The compute node worker consolidates the image processing tasks assigned by the Nokkhum controller. The tasks include image analytical solutions such as motion and face detection. The compute node worker includes resource sensors for monitoring the availability and capacity of computer resources and collects information about the controller's monitoring. The compute node worker also includes resource sensors for monitoring usage requirements, such as the CPU utilization, and memory and storage space. This resource information is delivered to the Nokkhum controller.

One important module in the compute node worker is the image processing task manager which controls and monitors tasks and the task pool. This module manages the task life cycle involving the creation and termination of tasks. In addition, the manager can watch task behaviors and handle shortcomings in order to promote system availability. The image processing task runs on a compute node worker containing image processing solutions for the video surveillance system. When the image processing task obtains logging messages, this module sends the messages to the controller via a resource status report. The output uploader module uploads images and video records to cloud storage via the Amazon S3 API. After the image processor output has been completely uploaded, the module can release it from the harddisk. In addition, if a compute node worker crashes, the tasks running on it will be terminated and restarted on another compute node worker without affecting other workers.

3.2.3 Nokkhum image processor

A Nokkhum image processor provides a VCA as a set of image analysis modules using the OpenCV library. The camera configuration and image processor attributes are described using JavaScript Object Notation (JSON) [5] for building an image processing process. The camera configuration is a dictionary containing attributes such as a name, a Uniform Resource Identifier (URI) [20], width, and height. The image processor attributes are listed and identified by "image_processors" keywords which contain the JSON attributes. The list of image processor attributes allows the use of nested descriptions for complex configurations. The Nokkhum image processor parses JSON information from the compute node pipe that works as a standard input and constructs image processing threads. The image processor is a computing process controlled by the compute node worker.

The Nokkhum image processor is flexible because users can design their VCA via JSON configurations. We implement this component by employing threads and queues of image objects. Each image analysis is a running thread which connects to others via an image queue. Each image analysis thread generally includes two queues for image input and output. The image analysis thread receives an image object from the input queue and processes it. Afterwards, it puts the object into the output queue for another thread. A configuration example for the Nokkhum image processor is shown in Fig. 4.

Figure 4 contains several image analysers: a motion detector, a face detector, a video recorder, and an image recorder. When a configuration is sent to a Nokkhum image processor, it translates that configuration, creating related image queues.

3.2.4 Nokkhum API

A popular way of managing a distributed system is by using a single API server and multiple clients. One advantage is that the developer can carefully implement a central API

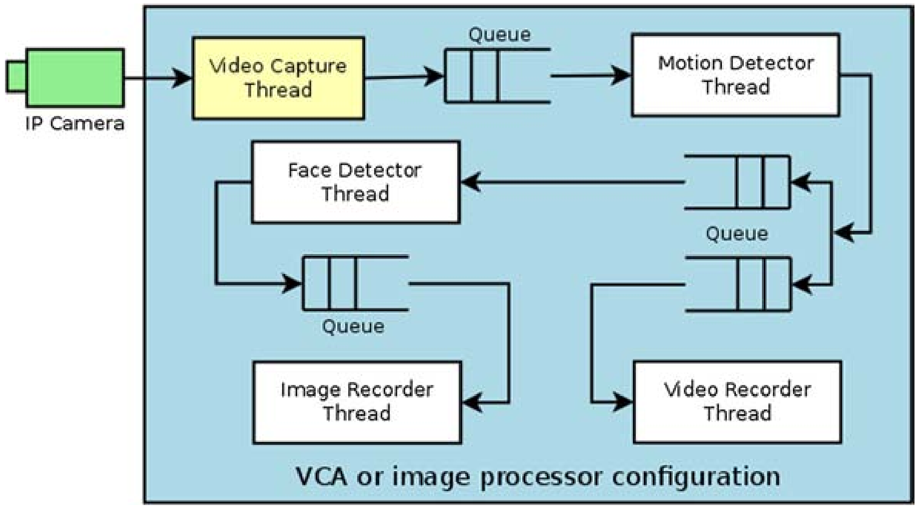


Fig. 4 An example Nokkhum image processor configuration

server, so that the client implementation is light weight, and can support various operating system platforms. The Nokkhum API provides for camera management, image processing task control, video play-back, and media management for basic users. For administrators, the API server provides system monitoring and high-level permission management of the functions that are provided to the users.

The API was developed using Python, Pyramid [21], and MongoDB [4], and uses a REST style architecture via HTTP [6]. Client are identified with token-based authentication when they connect to the server. A token is generated by the server during the authentication, and used in the reply sent to the client. However, the API requires a secure connection for protecting the privacy of user requests. A simple way is to switch to Hypertext Transfer Protocol Secure (HTTPS) [28].

3.2.5 Nokkhum client and web front-end

A Nokkhum client could be inplemented in several ways—as a web front-end, a mobile application, or as a desktop application, because it connects to the Nokkhum API server using HTTP and a REST architecture. We prefer a web front-end client since it can be used via a web browser on both desktop and mobile devices. Moreover, a web-based interface can be displayed on many platforms, and is more easily developed and maintained than native programs. The web front-end interface is implemented with Python and HTML, and utilized by both users and the administrator to provide camera and system information according to user role permissions. Users can create new camera configurations, compose image analyses, and watch videos or images obtained from the cloud storage.

Although the web front-end can be used on many platforms, the users may have to adjust the device's screen resolution which is somewhat inconvenient. Consequently, it is a burden for the developer who develops the native client to adjust it for best user experience and satisfaction.

3.3 Nokkhum architecture

Nokkhum has a scalable VSaaS architecture by using message passing, implemented with the Advanced Message Queuing Protocol (AMQP) [37], for connecting the Nokkhum controller and the compute node workers. This design provides many options for system deployment and the exploitation of mixed computating resources (both virtual and physical) for improved performance. All of Nokkhum's cooperating components are shown in the architecture overview in Fig. 5. The inter-component communication and data format are described in the following sections.

3.3.1 Running nokkhum on infrastructure as a service

The cloud infrastructure that provides physical or virtual machines, and other computer resources, is known as Infrastructure as a Service (IaaS). IaaS supplies computing resources on demand according to the user's requirements. IaaS plays an important role in start-up businesses by replacing high computer hardware purchasing costs with lower, pay-as-use resource rental fees. Moreover, this scheme allows large businesses to

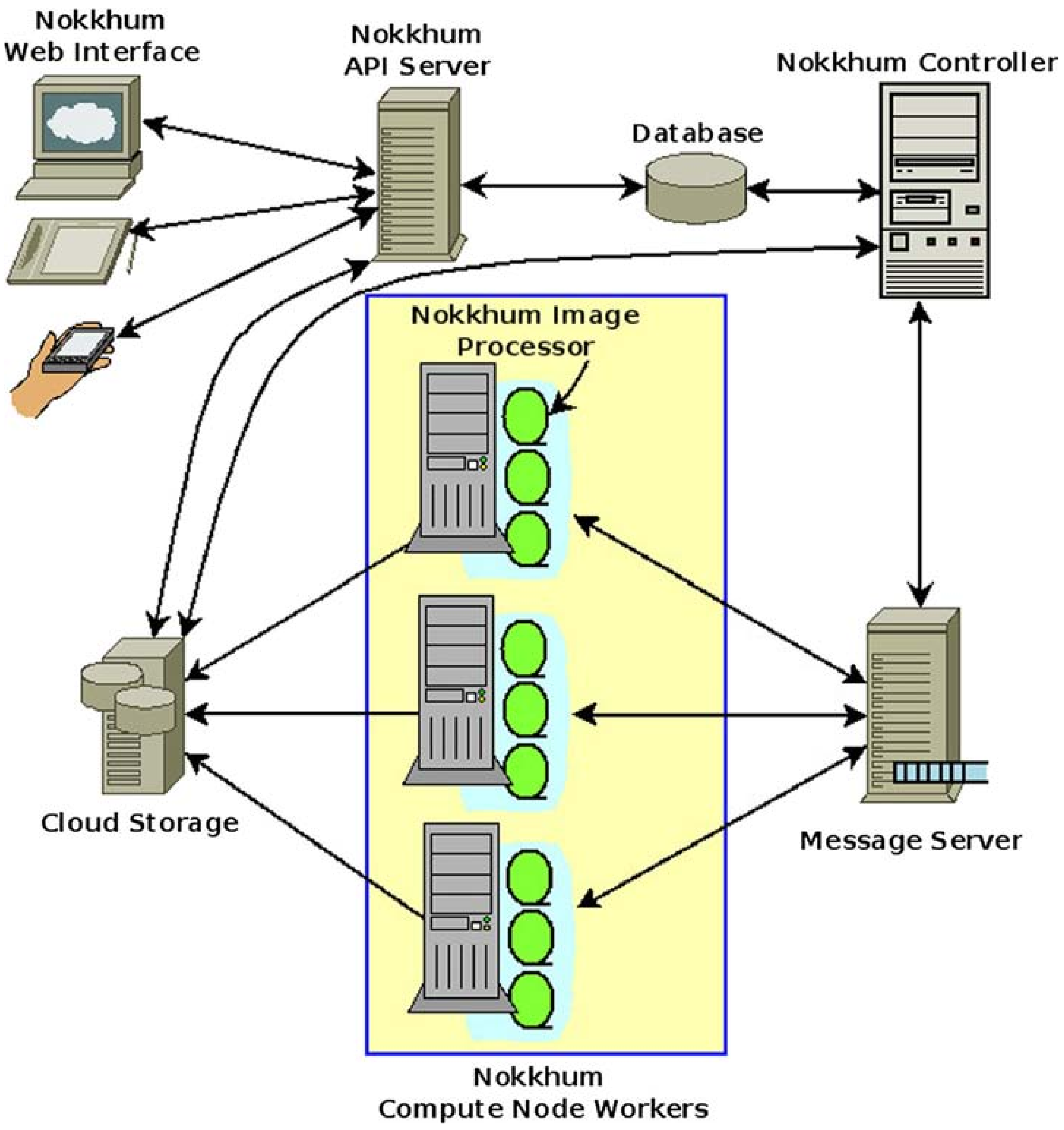


Fig. 5 The overview of the Nokkhum architecture

reduce their hardware maintenance and administrative costs. As a result, many organizations are currently transforming their IT platforms into cloud computing services. However, there are few available details about the implementation of Software as a Service (SaaS).

IaaS providers typically implement their own APIs to access and manage computing resources by exploiting a compatible version of the AWS API. AWS provides many services, but the most popular are Amazon EC2 for computing and Amazon S3 for storage. There are many open source IaaS products that support the AWS API, including Eucalyptus [24], OpenNebula [22], CloudStack [2], and OpenStack [12]. Since IaaS can provide a private cloud infrastructure for an organization, Nokkhum is designed using the Amazon EC2 and Amazon S3 APIs so it can run on most IaaS software providers.

3.3.2 Inter-component Communication

Nokkhum employs two main kinds of inter-component communication for exchanging information and controlling the system's behavior. The communication between the Nokkhum controller and Nokkhum compute node workers uses message-oriented middleware for distributing commands and reporting resource status details to its physical and virtual machines. Program-to-program communication on the same computing machine is utilized between compute node workers and image processors. These two approaches are explained in more details below.

1. Controller and Compute Node Workers Communication

   As shown in Fig. 5, the controller communicates with compute node workers using message passing. Two types of communication pattern are used: direct exchange and topic exchange. The controller and the workers employ direct exchange for greeting messages and updating worker resources and image processing status or behavior. This allows the workers to send these kinds of messages without waiting for a response. The controller and the workers use topic exchange for synchronizing command messages, such as to start or stop image processor processes, or to request greeting information. Topic exchange communication is designed to wait for a response message and command confirmation.

   As shown in Fig. 3, when a new compute node worker starts and finishes the boot state, it will send a greeting message to the message broker server. Then the controller will receive a greeting message and add the worker to the resource pool. After receiving the greeting message, the controller sends configuration details to the worker, such as cloud storage configuration that includes a comnunication protocol for accessing the storage. Therefore, Nokkhum VSaaS requires central configuration from the controller, which distributes it to every compute node involved. After the compute node worker receives configuration information, it will update its system resource usage details, run the required image processors, and report CPU utilization and memory usage to the controller.

2. Compute Node Workers and Image Processors Communication

   A compute node worker communicates with an image processor using a pipeline. An image processor is created by a worker when it receives a starting control message. The worker outputs all the commands for controlling the image processor's behavior via its standard input. When the image processor generates message output, the compute node worker will read it from its standard output. The commands and results are written in JSON format to facilitate processing. The worker checks the availability of all the image processors by monitoring their pipelines. Then, the compute node worker will report to the controller via a resource update system message.

3.3.3 Data format

JSON is utilized as the default data format for cameras, image processors, and inter-component messages because it is both lightweight and available in several computer languages. Cameras and image processor attributes are described via JSON objects. A camera object includes image size, frame rate (frames per second) for video, camera manufactory information, and a username and password to access the camera. An image processor object includes the name and attributes of the image processor. Inter-component message passing uses a JSON object for describing the command property. Listing 1 shows a JSON description for starting video surveillance commands composed by the Nokkhum controller.

```
{
  "action" : "start",
  "attributes" : {
    "cameras" : [
      {
        "id" : "52779ae724b5b108e243649e",
        "password" : "",
        "model" : "DCS-930L",
        "width" : 640,          // video width
        "height" : 480,         // video height
        "fps" : 10,             // frames per second
        "name" : "camera-01",
        "audio_uri" : "http://example.com/audio.cgi",
        "video_uri" : "http://example.com/video/mjpg.cgi?.mjpg",
        "image_uri" : "http://example.com/image/jpeg.cgi",
        "username" : ""
      }
    ],
    "image_processors" : [
      {
        "name" : "Motion Detector",
        "wait_motion_time" : 5, // wait for motion in 5 second
        "interval" : 3,         // process every 3 image
        "sensitive" : 95,       // motion sensitivity 0 - 100
        "image_processors" : [
          {
            "height" : 480,     // image height
            "fps" : 10,         // frames per second
            "width" : 640       // image width
            "name" : "Video Recorder",
            "record_motion" : true, // enable motion recording

          }
        ]
      }
    ]
  }
}
```

Listing 1 JSON description for starting the image processor command

4 Nokkhum system scalability

Nokkhum VSaaS components are divided into five groups, with each providing specific functions. These components can be distributed across many computers connected to the message broker server, which enables Nokkhum to easily scale to support dynamic user requirements. Moreover, Nokkhum can start running on just one machine, providing a small VSaaS system, and easily transition to supporting more cameras and image processing tasks, involving more computing machines. This section describes possible topologies for Nokkhum VSaaS in several scenarios.

As seen in Fig. 6, Nokkhum's components and other infrastructure software can be distributed across many types of computing machines. Figure 6a shows the smallest system type, where all the components run on a single machine supporting a small VSaaS. In order to extend the system for processing more cameras, the computing unit (Nokkhum compute node worker and image processor) can restart congested tasks on another computing machine added to the controller node in Fig. 6b. In Fig. 6c, more computing units are added to deal with an increased number of cameras, and to handle the expanded image processing requirements. In this configuration, the computing units and the controller unit are separated parts. Figure 6d is similar to Fig. 6c, but the cloud storage is moved to another machine for easier storage management and to reduce the computation load of the controller node. This configuration is suitable for a medium sized VSaaS.

The system topologies in Figs. 6e, 6f, and 6g are preferable when a highly scalable system is needed because the Nokkhum components and infrastructure software tasks are distributed across many computing machines. In Fig. 6e, MongoDB and the message server are moved to run on different machines to increase system availability. The message server and MongoDB are shared services for supporting computing units and controller nodes. If the Nokkhum components and the message server are not connected, it is likely that the system will fail. Therefore, the extracted parts, MongoDB, and the message server, are designed to closely work with the controller node in all proposed topologies to increase system availability. In Fig. 6f, the Nokkhum controller is separated from the front-end machine. The rationale is that the controller component can run on its own without affecting other components and if the controller component runs on a private network, it also increases security. The architecture is also designed to support a complete distributed system of which components are located on many computing machines as shown in Fig. 6g, thereby providing a large scale VSaaS system. This scenario is appropriate for a public VSaaS provider, and offers a high level of system availability. On the other hand, it involves many computing machines and consumes many system resources.

The Nokkhum architecture addresses system scalability across various scenarios. The VSaaS providers can freely choose configuration patterns for supporting their requirement. Moreover, the Nokkhum component configuration can combine both physical and virtual machine servers according to the organization's economy and IT proficiency. The organization can choose their security policy whether running many components in a private network or providing context components in a public network.

5 System testing

The system testing in this section focuses on the response times of the Nokkhum VSaaS in the Cloud environment. We utilized general-purpose PCs for building the cloud infrastructure, with OpenStack as the cloud middleware. The PCs used several different CPUs, such as AMD Phenom II X6 1055T and AMD FX-8320, and their memory range from 8 to 16 GB, with 1 TB for storage. The experiments were designed to create a basic performance matrix including the virtual machine acquisition time, waiting time, and processing time for both user and system request commands.

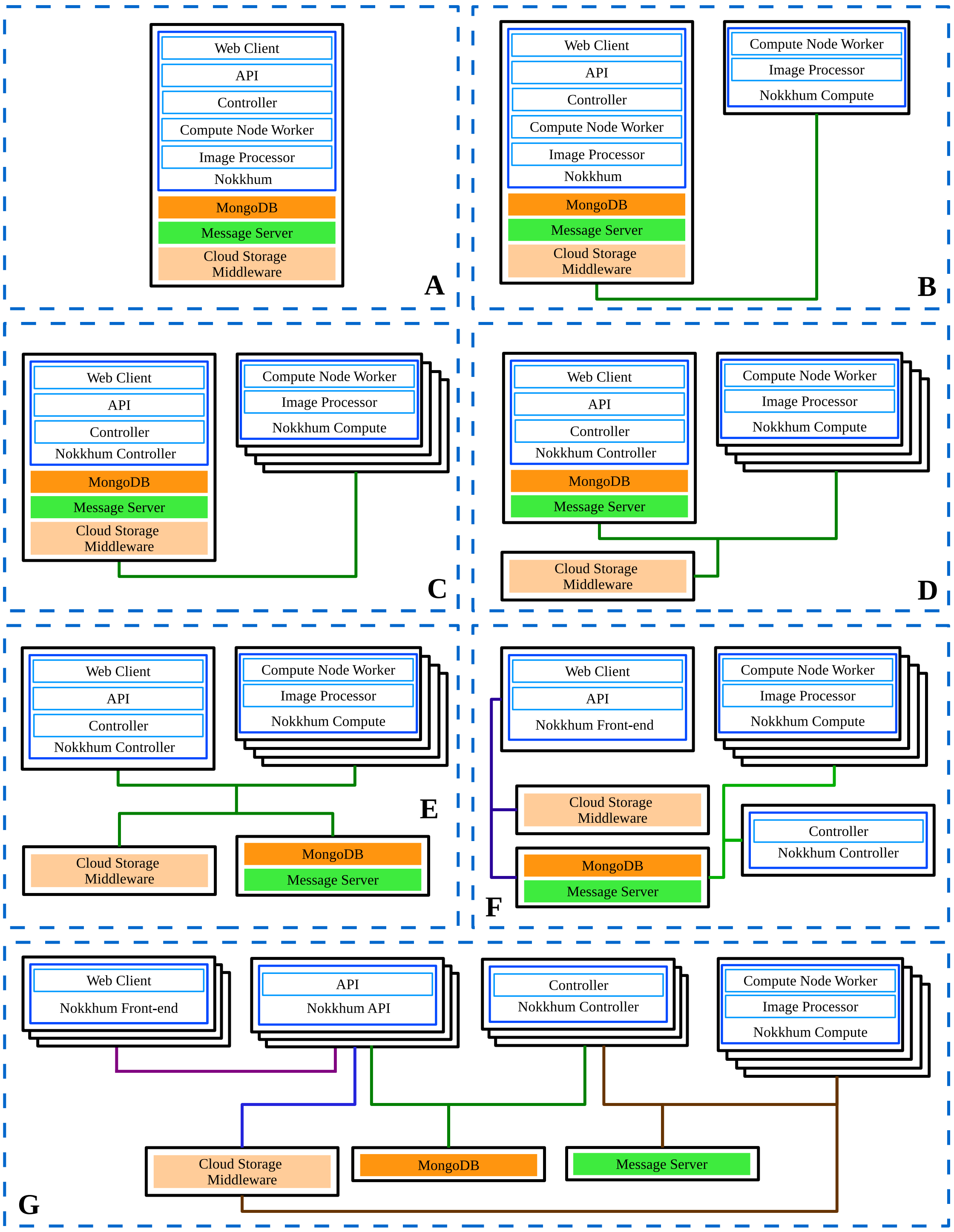


Fig. 6 Nokkhum topology configurations

5.1 Cloud environment setting

In our cloud testing environment, we employed the OpenStack service family on generic PCs. For virtual machine providers, we chose OpenStack Nova and KVM [15] as a Hypervisor. A physical cloud structure is implemented by a PC acting as a cloud controller, and many PCs utilized as cloud compute nodes, and connected using two Ethernet interfaces. The first interface is for internal communication, and the other for public use via a 10/100 Mb Ethernet. The public interface connects all the cloud controllers, cloud compute node workers, cloud storage and IP cameras. The cloud environment setting is shown in Fig. 7.

Nokkhum VSaaS is designed to run on a cloud infrastructure, and all the possible topology configurations shown in Fig. 6 have worked successfully in our cloud infrastructure testing system. In this paper, we will only investigate the system's response time for a medium sized VSaaS (the scenario in Fig. 6d) which supports 10-20 VMs of compute node workers depending on the capabilities of the CPUs and memory of the controller node; Its deployment is shown in Fig. 8. We chose to emphasize the Fig. 6d scenario because it is a good representative of the topology configurations typically used by many organizations, departments, and universities for providing video surveillance. Fig. 8 shows a virtual machine as a controller node, containing a message server (RabbitMQ [26, 36]), databases (MongoDB), a web front-end, an API and a controller. In addition, each compute node virtual machine image contains a compute node worker and a image processor. The IP address of the message server in the VSaaS system must be identified in the compute node worker configuration. After

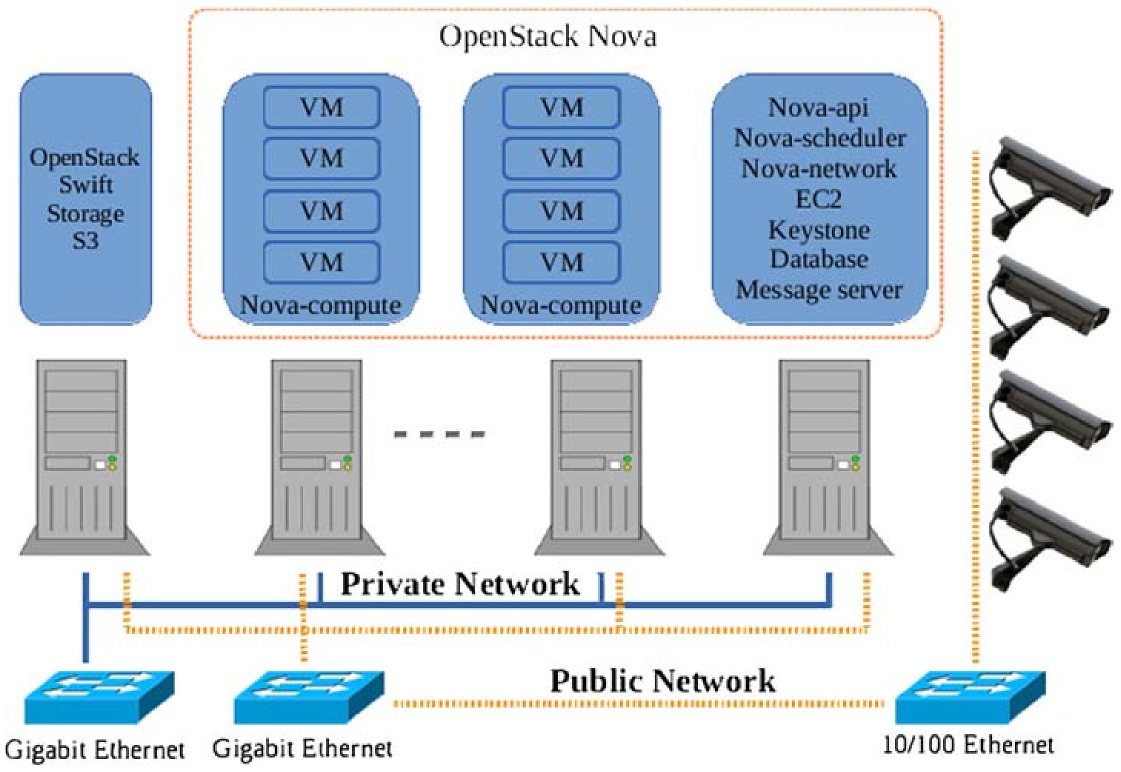


Fig. 7 Cloud environment setting using OpenStack

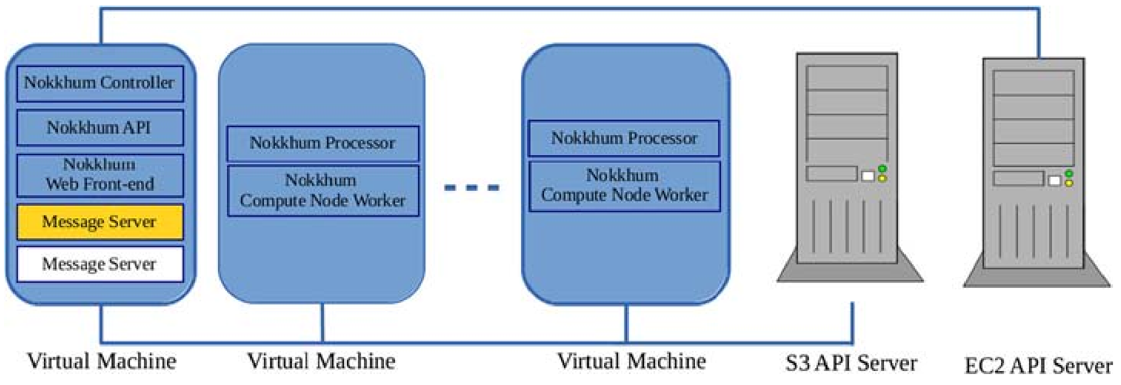


Fig. 8 Nokkhum VSaaS on a cloud infrastructure

registering the compute node image in the cloud infrastructure, the compute node image name is added to the controller and the VSaaS system can start.

5.2 Scalability and flexibility check lists

Scalability and flexibility check lists confirming that the Nokkhum VSaaS system is performing according to the system design appear in Tables 1 and 2. In terms of scalability, the system can automatically acquire VMs, provide for various user groups, support IaaS, and be scaled up by distributing its components across VMs and physical machines. In terms of flexibility, the system can automatically handle different situations concerning suspended cameras, and malfunctioning or unavailable compute node workers, controller, and message server.

5.3 Experimental results

The Nokkhum VSaaS is based on the design described in Section 3, and its cloud environment set up as explained in Section 5.1.

Table 1 Scalability check list

List	Check
Acquire VMs automatically	✓
Provide for various user groups	✓
Support IaaS with Amazon EC2/S3	✓
Components can scale according to Fig. 6	
A: compact system on one machine	✓
B: add a compute node worker	✓
C: compute node workers are separated from the controller	✓
D: separate cloud storage service	✓
E: separate database and message server	✓
F: separate front-end node	✓
G: fully distributed system	✓

Table 2 Flexibility check list

List	Check
System automatically recovers after a suspended IP camera becomes available	✓
System handles compute node worker errors (either electricity or network problems) and acquires a new one	✓
Compute node worker continues running when controller node is unavailable	✓
Controller node can recover its status when it resumes running	✓
Controller and compute node worker can continue running when the message server is unavailable	✓

The experiments involved many users and various image processing configurations such as motion detection, face detection, video recording, and image recording. They utilized about 20 IP cameras including models such as the Dlink DSC-930L, Dlink DSC-2102, AXIS 215 PTZ, and AXIS 211M. About 20 IP cameras were distributed at the Robot Building, Department of Computer Engineering, Prince of Songkla University. The video processing experiments employed a frame rate of 10 – 15 frames per second, and image size of 640x480 pixels. The minimum video recording space per camera per day varied from 65 KB to 210 MB. Total video recording space per camera used each day varied from 361MB to 19 GB.

The Nokkhum front-end node was executed as a single VM and the Nokkhum compute unit utilized a 3 GB QEMU Copy On Write (QCOW2) disk format. The virtual hardware template for instance acquisition was equipped with a 2-core CPU, 2048 MB RAM, and a 40 GB harddisk. The resulting experimental data is shown in Table 3 and in Figures 9– 14.

Table 3 shows the acquisition time for OpenStack Nova to provide instances. The spawning time is the time to transfer an image from the OpenStack Glance server to the destination Nova Compute Node Worker, plus the time to prepare the image for booting. The image cache plays an important role in instance spawning. If the Nova Compute Node Worker has previously run an instance, it may cache its image for a later run. This reduces the image transfering time so it can boot much sooner. The instance booting time is the interval between when the VM is first active to when the Nokkhum controller gets its first response from the Nokkhum compute node worker. VM acquiring average time (spawning and booting time) is 128.31 seconds without image cache, and 27.346 seconds with caching.

In this experiment, there are two types of message commands for controlling the starting and stopping of the Nokkhum image processor. Firstly, User Request Command (URC)

Table 3 Virtual machine acquisition time

Activity	Time (s)		
	Min	Max	Average
Instance spawning with image cache	11.150	11.470	11.265
Instance spawning without image cache	104.930	117.011	112.229
Instance booting	13.220	19.379	16.081

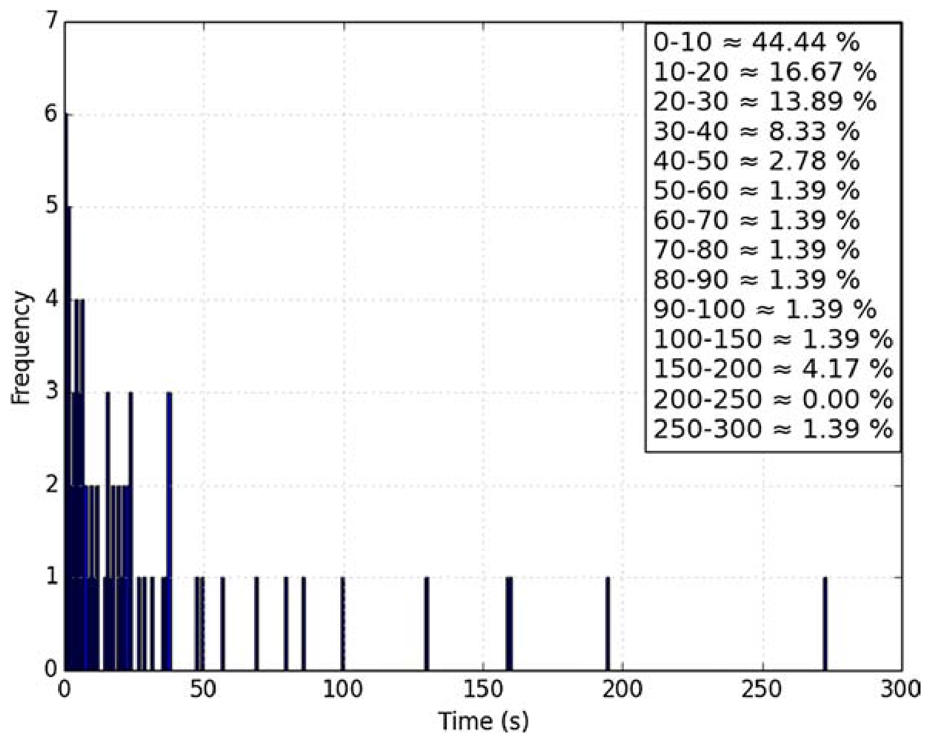


Fig. 9 User request command (URC) waiting time for starting image processors

messages are generated when the user requests the starting or stopping of video processing. Secondly, System Request Command (SRC) messages are generated by the task controller module in the Nokkhum controller when the image processor crashes. The task

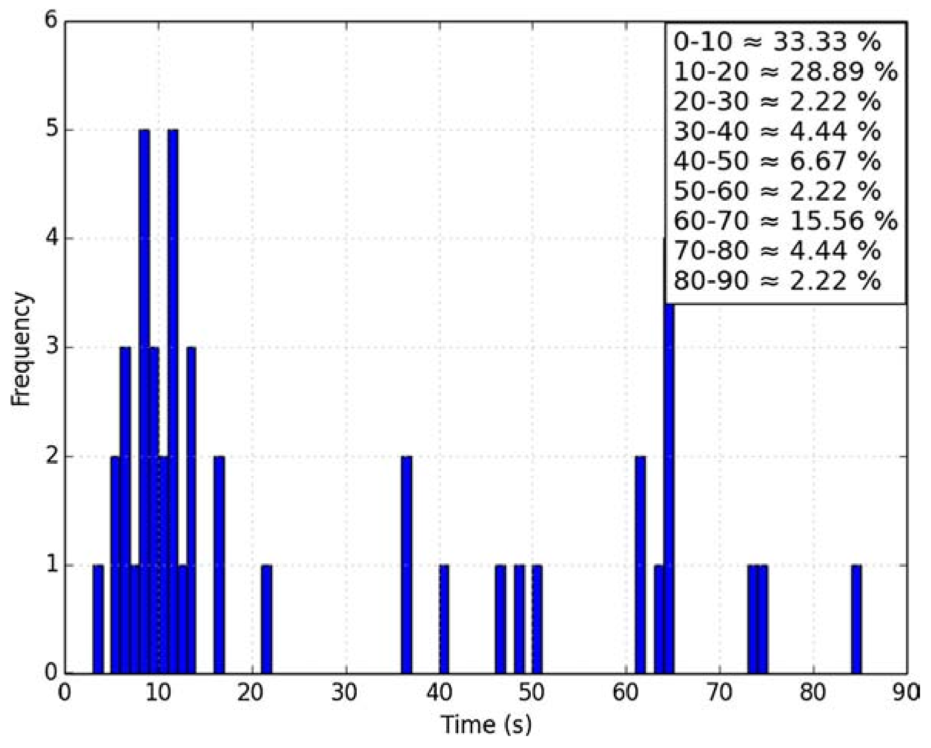


Fig. 10 User request command (URC) waiting time for stopping image processors

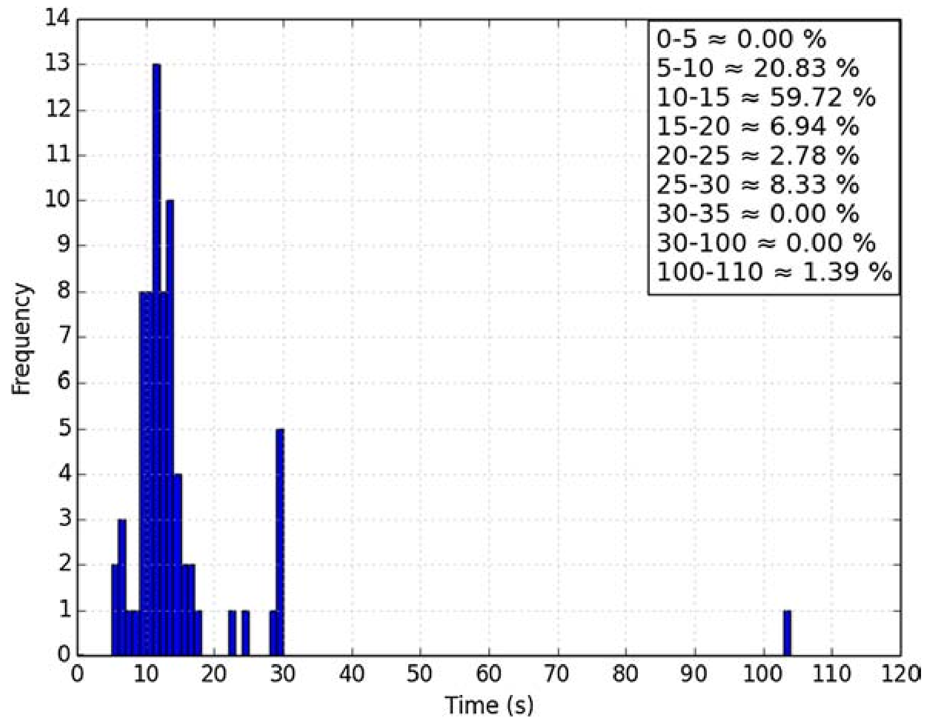


Fig. 11 User request command (URC) processing time for starting image processors

controller adds a SRC to the command waiting queue to start the image processor. The waiting and command processing times for a SRC indicates the system's ability to serve user requests.

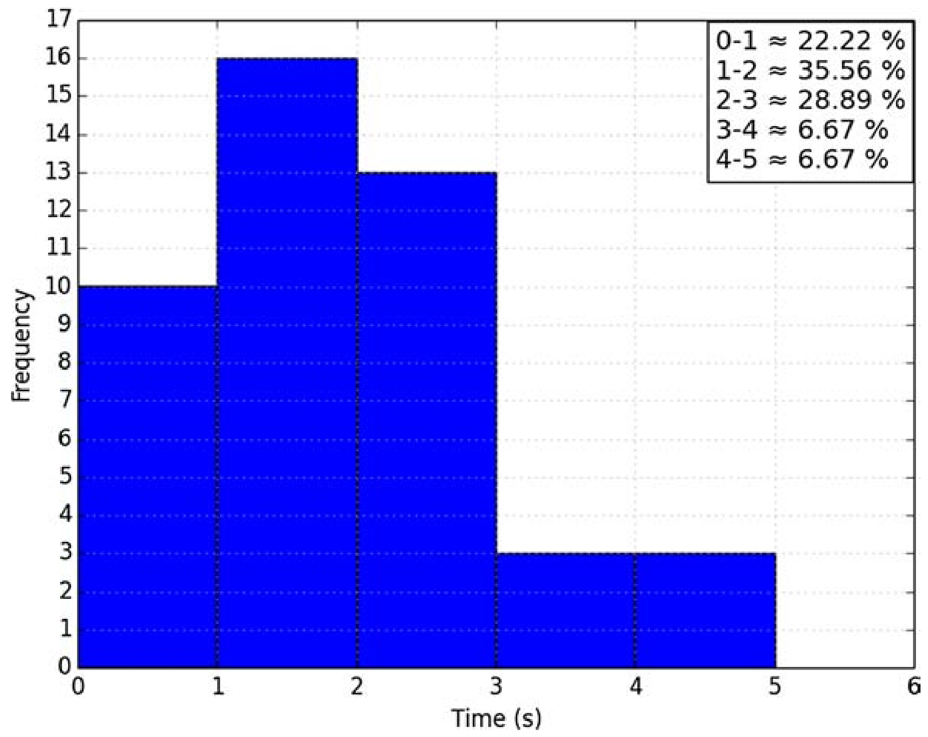


Fig. 12 User request command (URC) processing time for stopping image processors

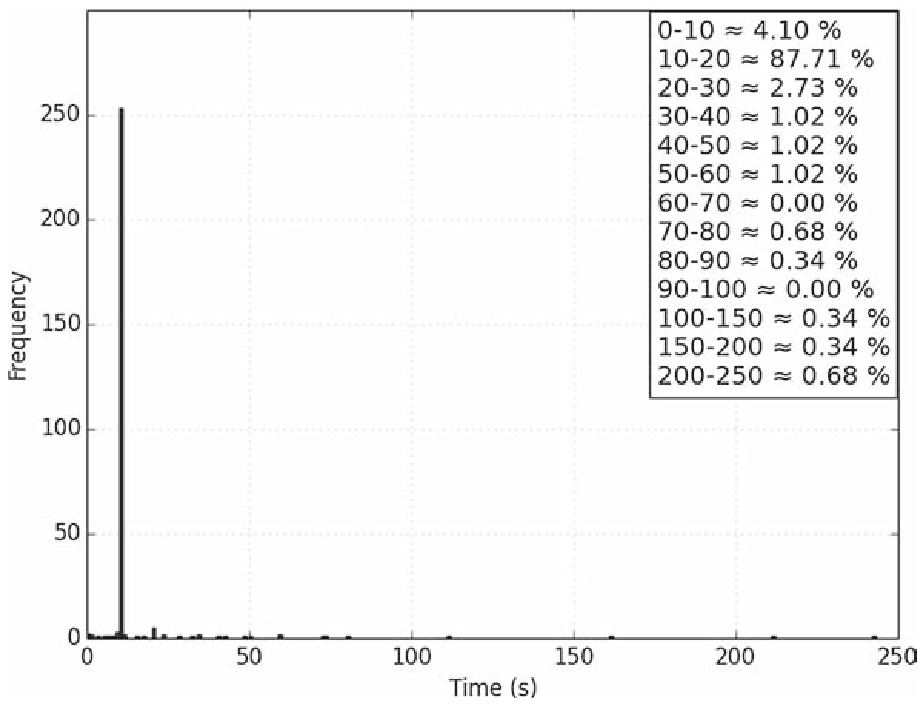


Fig. 13 System request command (SRC) waiting time for starting image processors

Figures 9 and 10 show the waiting time histograms for URCs to start and stop image processors. Figures 11 and 12 show the processing time duration histograms for URCs to start and stop image processors. The command waiting time histograms contain a range of time

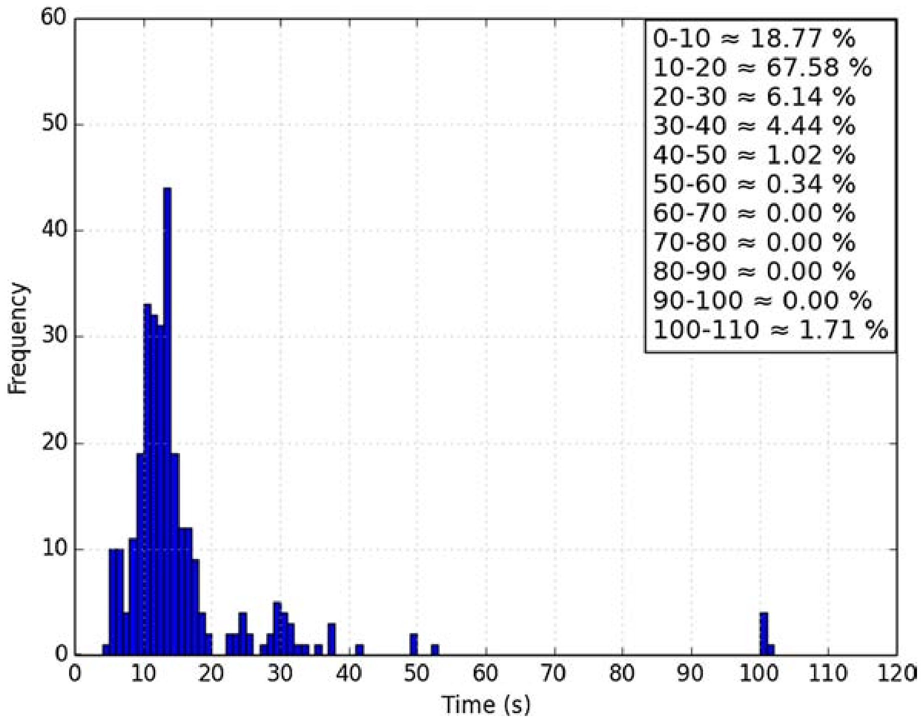


Fig. 14 System request command (SRC) processing time for starting image processors

distributions due to several causes: the task scheduler is a single thread performing sequential processing, the user submits requests for several sequential actions in too short a time, and the user command action must wait for an available compute node. The waiting time for a URC to start an image processor begins when the system starts loading the image processor binary, and finishes when the image processor gets its first image from the video connection. The image acquisition time varies depending on the IP camera type, camera model, and network topology. Most of the URC processing times for starting an image processor are in the range 10–15 seconds (Fig. 11). Measuring the processing time for a URC to stop an image processor does not involve the impeding causes when measuring the command waiting time. Most the URC waiting times for starting an image processor are in the range 0–10 seconds (Fig. 9) because long waiting times for acquiring a new compute node worker do not often occur. Most waiting and processing times for URCs that stop an image processor in the range 0–10 and 1–2 seconds respectively (Figs. 10 and 12).

Figures 13 and 14 show the waiting and processing times of SRCs for image processor recovery. The waiting and processing times of SRCs are similar to the waiting and processing times of URCs. But URC events occur spontaneously while SRC events occur automatically. For example, when an image processor exits with an error because it cannot acquire an image from the video connection, then the compute node worker will detect the suspicious consumption of computing resources. Most waiting and processing times for SRCs starting an image processor have similar characteristics, falling within the range 10–20 seconds. The response times show that the system works well. However, results of this kind are difficult to compare between different systems since some focus on resource allocation rather than system architecture. Also, some research systems use simulated data rather than actual results.

Table 4 Compairison of Cloud-based Video Surveillance Systems

System	Architecture	AAA service	Cloud technology	Real Time image Analysis	Bandwidth usage
Nokkhum	Distributed components	Account and Billing, Authentication, Authorization	Nova/EC2, Swift/S3	Yes	According to configuration
P2PCloud	Distributed physical nodes	N/A	HDFS	L/I	Low (only when event detectors are used, otherwise as high as others)
VAQACI	Distributed components	Billing	MapReduce, HDFS	Only after videos are recorded and pushed into the HDFS	High (centralized video controller) by Hadoop

Table 5 System scalability compairison

System	Resource allocation	Deployment configuration	Minimum number of deployed machines
Nokkhum	Automatic	Supporting many scenarios according to Fig. 6	1
P2PCloud	N/A	Local node and directory node	Minimum required 2 (central and local)
VAQACI	Automatic	One scenario	2

5.4 Discussion

In this section, we compare Nokkhum with available video surveillance systems as shown in Tables 4 and 5. However, direct comparisons are difficult because some of these systems are conceptual designs without complete implementations. Some focus on resource allocation for system scalability rather than software architecture.

As a consequence, some descriptions are unavailable (N/A) or lacking in information (L/I). For example, Nokkhum provides Account/Billing, Authentication, and Authorization (AAA), while the situation is unclear for other systems. Nokkhum can more flexibly configure its components for different deployment scenarios. For instance, it can utilize resources in a minimal way, then later scale them up. In this way, Nokkhum is not just a specific type of video surveillance system applying cloud computing technology, but also supports various users in the public cloud, flexibility configurations for different system sizes, and scalability, which are topics that are not addressed in other systems.

Section 5.3 included histograms of VSaaS response times for service delivery to users. Other cloud-based surveillance systems do not provide such response time information. Instead, they focus on image processing performance or network bandwidth usage. Our system response times are similar to those for the bundled software that come with the IP cameras. However, there are many factors that can affect the URC processing times for starting an image processor. These include the camera response times for starting video streaming, resource allocation waiting time, and the URC scheduling waiting time. Nokkhum supports real time image analysis while other systems do not fully support it. Nokkhum can vary its network bandwidth usage according to its users' configurations while other systems are not as flexible.

6 Conclusion

Our Nokkhum Video Surveillance as a Service (VSaaS) software architecture can automatically scale the number of virtual machines needed to support user requirements. The Amazon EC2 API handles automatic virtual machine acquisition, while the Amazon S3 API is used by the storage engine. Our VSaaS allows its compute units to be composed from both virtual machines and physical servers, and all the components are designed for system scalability and flexibility. Many server topologies can be configured according to the provider requirements. Our architecture can be deployed using a minimal number of servers at the beginning and later be scaled up easily for a growing number of cameras. Nokkhum supports small to large VSaaS, in both private and public clouds.

A REST interface allows the systen to support multiple types of client. Experiments show that the system performance is quite decent, and is flexible enough to tolerate difficult scenarios when some components are disconnected. Typical causes are electricity black-out, lost of network communication, or suspension due to software errors which require a restart.

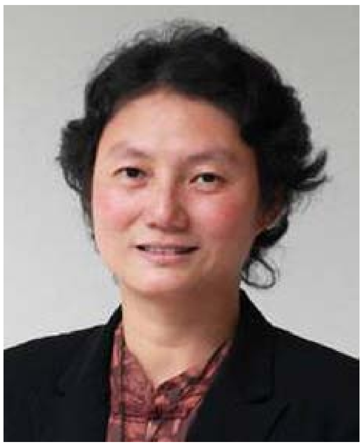
Acknowledgments The authors are grateful for financial supports from the Thailand Research Fund and Prince of Songkla University through the Royal Golden Jubilee Ph.D. Program (Grant No. PHD/0047/2552).

References

1. Amazon (2014) AWS documentation. http://aws.amazon.com/documentation/
2. Apache Software Foundation (2014) Apache cloudstack. http://cloudstack.apache.org/
3. Baggio DL, Emami S, Escrivá DM, Ievgen K, Mahmood N, Saragih J, Shilkrot R (2012) Mastering OpenCV with practical computer vision projects. Packt Publishing
4. Chodorow K, Dirolf M (2010) MongoDB: the definitive guide. O'Reilly Media, Inc
5. Crockford D (2006) Json: The fat-free alternative to xml. In: Proceeding of XML, vol 2006
6. Fielding RT, Taylor RN (2002) Principled design of the modern web architecture. ACM Trans Internet Technol 2(2):115–150. doi:10.1145/514183.514185
7. Georis B, Desurmont X, Demaret D, Redureau S, Delaigle J, Macq B (2003) IP-distributed computer-aided video-surveillance system. In: Proceedings of the IEE symposium on intelligence distributed surveillance systems, pp 18/1–18/5
8. Gualdi G, Prati A, Cucchiara R (2008) Video streaming for mobile video surveillance. IEEE Trans Multimed 10(6):1142–1154. doi:10.1109/TMM.2008.2001378
9. Han J, Choi N, Chung T, Kwon T, Choi Y (2012) A target-centric surveillance system based on localization and social networking. Multimed Tools Appl:1–25. doi:10.1007/s11042-012-1285-8
10. Hossain M, Hassan M, Qurishi M, Alghamdi A (2012) Resource allocation for service composition in cloud-based video surveillance platform. In: Proceedings of the 2012 IEEE international conference on multimedia and expo workshops (ICMEW), pp 408–412
11. Huang Y (2010) The design and implementation on a new generation of remote network video surveillance system. In: Proceedings of the 2010 3rd international conference on advanced computer theory and engineering (ICACTE), vol 2, pp V2–294–V2–297
12. Jackson K (2012) OpenStack cloud computing cookbook. Packt Publishing
13. Jaynes C, Webb S, Steele RM, Xiong Q (2002) An open development environment for evaluation of video surveillance systems. Proceeding of the third international workshop on performance evaluation of tracking and surveillance. PETS 2002(1):32–39. http://citeseerx.ist.psu.edu/viewdoc/summary
14. Karimaa A (2011) Video surveillance in the cloud: dependability analysis. In: Proceedings of the fourth international conference on dependability, DEPEND 2011., pp 92–95
15. Kivity A, Kamay Y, Laor D, Lublin U, Liguori A (2007) kvm: the linux virtual machine monitor. In: Proceedings of the Linux Symposium, vol 1, pp 225–230
16. Lee J, Feng T, Shi W, Bedagkar-Gala A, Shah S, Yoshida H (2012) Towards quality aware collaborative video analytic cloud. In: 2012 IEEE 5th international conference on cloud computing (CLOUD), pp 147–154
17. Limna T, Tandayya P (2012) Design for a flexible video surveillance as a service. In: Proceedings of the 2012 5th international congress on image and signal processing (CISP), pp 197–201
18. Lin CF, Yuan SM, Leu MC, Tsai CT (2012) A framework for scalable cloud video recorder system in surveillance environment. In: Proceedings of the 2012 9th international conference on ubiquitous intelligence computing and 9th international conference on autonomic trusted computing (UIC/ATC), pp 655–660
19. Mahmoud Q (2005) Middleware for communications. Wiley
20. Masinter L, Berners-Lee T, Fielding RT (2013) Uniform resource identifier (URI): generic syntax. http://tools.ietf.org/html/rfc3986
21. McDonough C (2011) The pyramid web application development framework. http://docs.pylonsproject.org/projects/pyramid/en/1.4-branch/

22. Milojičić D, Llorente IM, Montero RS (2011) Opennebula: a cloud management tool. IEEE Internet Comput 15(2):11–14
23. Neo IT Solutions Inc (2013) OVS online video surveillance as a service | VSaaS | MVaaS | RVMaS | VAS. http://www.neovsp.com/solutions
24. Nurmi D, Wolski R, Grzegorczyk C, Obertelli G, Soman S, Youseff L, Zagorodnov D (2009) The eucalyptus open-source cloud-computing system. In: 9th IEEE/ACM international symposium on cluster computing and the grid, 2009. CCGRID'09., IEEE, pp 124–131
25. NW Systems Group Limited (2013) SecurityStation - how VSaaS works and the benefits of VSaaS. URL http://www.securitystation.com/how-vsaas-works.php
26. Pivotal Software Inc (2014) RabbitMQ – messaging that just works. URL http://www.rabbitmq.com/
27. Python Software Foundation (2014) Subprocess management – python documentation. http://docs.python.org/3.4/library/subprocess.html
28. Rescorla E (2013) HTTP over TLS. https://tools.ietf.org/html/rfc2818
29. Rodriguez-Silva D, Adkinson-Orellana L, Gonz'lez-Castano F, Armino-Franco I, Gonz'lez-Martinez D (2012) Video surveillance based on cloud storage. In: Proceedings of the 2012 IEEE 5th international conference on cloud computing (CLOUD), pp 991–992
30. San Miguel J, Bescos J, Martinez J, Garcia A (2008) DiVA: a distributed video analysis framework applied to video-surveillance systems. In: Proceedings of the ninth international workshop on image analysis for multimedia interactive services, 2008. WIAMIS '08., pp 207–210
31. Secure-i (2013) VCR DVR NVR... and now HVR.. http://www.secure-i.com/learn/technologies
32. Suvonvorn N (2008) A video analysis framework for surveillance system. In: Proceedings of the 2008 IEEE 10th workshop on multimedia signal processing, pp 867–871
33. Triornis Ltd (2013) ZoneMinder – main documentation. URL http://www.zoneminder.com/wiki/index.php/Documentation
34. Valera M, Velastin S (2005) Intelligent distributed surveillance systems: a review. In: Proceedings of the IEE vision, image and signal processing, vol 152, pp 192–204. doi:10.1049/ip-vis:20041147
35. Vezzani R, Cucchiara R (2010) Video surveillance online repository (ViSOR): an integrated framework. Multimed Tools Appl 50(2):359–380. doi:10.1007/s11042-009-0402-9
36. Videla A, Williams JJW (2012) RabbitMQ in action: distributed messaging for everyone. Manning Publications
37. Vinoski S (2006) Advanced message queuing protocol. IEEE Internet Comput 10(6):87–89. doi:10.1109/MIC.2006.116
38. Wan E, Van der Merwe R (2000) The unscented kalman filter for nonlinear estimation. In: The IEEE 2000 adaptive systems for signal processing, communications, and control symposium 2000. AS-SPCC., pp 153–158
39. White T (2009) Hadoop: the definitive guide, 1st edn. O'Reilly Media, Inc
40. Wijnhoven RGJ, Jaspers EGT, de With PHN (2006) Flexible surveillance system architecture for prototyping video content analysis algorithms. In: Proceedings of the SPIE, vol 6073, pp 60,730R–60,730R–9
41. Wu YS, Chang YS, Juang TY, Yen JS (2012) An architecture for video surveillance service based on P2P and cloud computing. In: Proceedings of the 2012 9th international conference on ubiquitous intelligence computing and 9th international conference on autonomic trusted computing (UIC/ATC), pp 661–666
42. Yuan X, Sun Z, Varol Y, Bebis G (2003) A distributed visual surveillance system. In: Proceedings of the IEEE conference on advanced video and signal based surveillance, 2003., pp 199–204

Thanathip Limna received his B.Eng. and M.Eng. degree in Computer Engineering from Prince of Songkla University, Thailand, in 2007 and 2010, respectively. Currently, he is pursuing a Ph.D. in Computer Engineering at the same university. He is interested in the research fields of Parallel and Distributed Computing and Systems, and Cloud Technology.



Pichaya Tandayya graduated in Electrical Engineering (Communications) from Prince of Songkla University (PSU) in Thailand in 1990. She obtained her Ph.D. in Computer Science in 2001 from the University of Manchester in the area of distributed interactive simulation. Currently, she is an Assistant Professor in the Department of Computer Engineering, PSU. Her current research works concern Parallel and Distributed Computing and Assistive Technology.

Multimed Tools Appl
DOI 10.1007/s11042-016-4225-1

Workload scheduling for Nokkhum video surveillance as a service

Thanathip Limna[1] · **Pichaya Tandayya**[1]

Received: 2 June 2016 / Revised: 14 November 2016 / Accepted: 29 November 2016

Abstract Video Surveillance Systems (VSS) on the Internet as known as Video Surveillance as a Service (VSaaS) or Cloud based Video Surveillance (CVS) systems. Video processing workload analysis has usually employed only one category of static video processing attribute, such as a frame rate with a single frame size on the same computing node specification, but VSaaS must handle a variety of video processing attributes. Also, in a static workload, it is difficult to identify the resource consumption of video processing attributes, especially involving a combination of frame rates and sizes on different computing nodes on virtual or physical machines. Consequently, it is difficult to place a task on a computing node if the resource usage information is unknown to the scheduler. In this paper, the video processing workload characteristics utilize various parameters, such as the type of video processing task, frame rate, frame size, and compute node specification. The analysis results have helped us to design a scheduler that supports different computing node specifications. We explore video processing workload for testing resource usage capacity in several computing nodes, and collect information for the scheduler's estimation. This paper also proposes a resource estimation module for predicting the video processing resource usage for a new video processing task when there is no matching or close estimation. Furthermore, we suggest scheduler criteria for optimizing system resource usage.

Keywords Video surveillance system · Video surveillance as a service · Scheduling · Workload analysis

✉ Pichaya Tandayya
pichaya@coe.psu.ac.th

Thanathip Limna
thanathip.limna@gmail.com

[1] Department of Computer Engineering, Faculty of Engineering, Prince of Songkla University, Hat Yai, Songkhla, Thailand

1 Introduction

Online Video Surveillance Systems (VSS) are known as Video Surveillances as a Service (VSaaS) [7, 15] or Cloud-based Video Surveillance (CVS) systems. VSaaS provides several service components including real-time video monitoring, online video playback, video recording, security alerts, and storage space. Customers rent a service package from a VSaaS provider, and attach Internet Protocol (IP) cameras to the system. The customer records the IP cameras' video using the VSaaS service, and chooses suitable video analysis.

There are several software stacks for building VSaaS systems on a cloud infrastructure, such as P2PCloud [17], VAQACI [9], the CVR system [11], VISERAS [15], and Nokkhum [10]. Most of the architectures are based on a cloud infrastructure and big data technology, such as Amazon EC2, Amazon S3, and Hadoop. Some systems include resource allocation techniques and algorithms for minimizing the number of virtual machines (VMs) on Cloud infrastructures, including Nan [13], Miao et al. [12], and Hossain et al. [4, 5], and suppose that the video analysis workloads consume static computational resources. Therefore, the VM providers offer the same VM capacity and capabilities. In reality, a VSaaS system must support different video analysis workloads which means that resource allocation must consider factors involving video analysis, video frame size, and frame rate.

Video processing tasks consume different computing resources according to their kind of video analysis, video frame size, and frame rate. Task processing on different machines involve different computing resource consumption according to CPU vendor, architecture, and frequency, which makes it difficult to predict computing resource consumption for video processing task scheduling. One way to estimate the consumption of different computing resources is to run a video processing testing suite to collect the results, utilizes results as heuristics. The heuristics can be automatically adjusted by rerunning the testing suite on new machines.

This paper presents video analysis workloads and task scheduling for VSaaS running on a cloud infrastructure. Our scheduling method can handle different video analysis configurations from VSaaS users on heterogeneous computational units. We present results for different VSaaS workloads running on our Nokkhum system.

2 Video surveillance as a service

Video Surveillance as a Service (VSaaS) is a cloud environment for supporting users located at different places. The users register their IP cameras with the VSaaS system, and pay a service charge over a billing period. VSaaS provides several facilities for managing the cameras, video processing, storage, and billing, and can scale to support dynamic numbers of video content analysis. Several VSaaSes have been deployed using cloud services, especially elastic computing units and object storage. VSaaS systems can request dynamic computing resources and easily scale according to system requirements.

Previous works [4, 5, 10, 12], and [13] as focused on CVS system design, mainly utilized Amazon EC2 and the S3 API. Hossain et al. [5] presented a VM allocation scheme for supporting video streaming for emergency officials. Hossain et al. [4] described a VSS framework for dynamic workloads such as face detection and storage tasks. Alamri et al. [1] and Hossain [6] were interested in the quality of service for distributed video surveillance services, especially video transcoding. They proposed a VSS and a service configuration algorithm based on video transcoding workloads. Hossain [6] described video workloads involving 320 × 240 pixels at 30 FPS. Alamri et al. [1] presented computing resource consumption for surveillance video streaming and video repurposing/transcoding, but without any information about video workloads. Unfortunately, there has been little work on video processing factors and workload characteristics for identifying computing resource consumption.

Deploying VSaaS in a cloud computing environment involves VSaaS system design and computing resource management, two issues which affect the quality of service. Video task scheduling involves symmetric video analysis workloads, including the frame size, frame rate, and video analysis type, running on a homogeneous computing units. Such VSaaSes display high resource efficiency when everything is static but these settings restrict the flexibility of the VSaaS's capacity, and do not alway support user requirements. Furthermore, when some computing hardware is replaced, the infrastructure will be changed from a homogeneous to a heterogeneous computing system. Therefore, flexible resource management is needed in VSaaS systems, especially for task scheduling handling dynamic user requirements and heterogeneous computing units.

We address these issues by analyzing CPU and memory consumption of several types of home-use video processing, such as motion detection, video recording, and motion recording. The collected video processing consumption data involves several types of video processing, various video frame rates and frame sizes, and several computing unit specifications based on physical servers and virtual machines. The analyzed results are employed in a scheduling process for placing the video processing tasks on a suitable computing node. We have designed video processing task scheduling that supports several situations described in the workload analysis. The aim is that the scheduling process can handle dynamic video processing workloads, and so enhance the VSaaS's efficiency and video processing resource utilization.

3 Analysis of video workload characteristics

Few VSSes perform video workloads analysis or emphasize scalable distributed video processing on a pool of computing resources. Many VSSes only utilize a single video size and variable frame rate in their case studies. Recently, VSSes had been transformed into VSaaS for supporting various user requirements, and such workload analysis is likely to provide wrong results for these systems. They do not take into account significant parameters involving changing video size and frame rate, or video processing based on several machine specifications. This section describes video workload characteristics affecting computing resources based on the Nokkhum VSaaS architecture.

3.1 Nokkhum VSaaS

Nokkhum [11] is a flexible, scalable component-based architecture for VSaaS. It can deploy components on both physical and virtual machines running Infrastructure as a Service (IaaS) using the Amazon AWS API. It has been designed and developed to support video applications for various organizations and business sizes, where the deployment and configuration involve a flexible range of computing units. It has an API interface server providing a REST over HTTP [3] which supports any client platform, and is able to control, view, or manage video analysis, video recording, and camera configuration. The Nokkhum architecture is shown in Fig. 1.

Figure 1 shows five components including the Nokkhum client, APIs, controller, compute node worker, and video processor. The components are described as follows:

- The controller component is a daemon process which consists of many modules including, task, scheduler, compute node, VM, notifier, billing, and storage controller. The compute node, VM, and storage modules play different roles providing elastic computational resources according to user requirements. The task and scheduler are concerned with monitoring, and managing video analysis tasks. They ensure that an analysis works until the user cancels it. The notifier provides a notification to the user when an exception occurs during an analysis. The billing module calculates the user service charge.
- A compute node worker is a daemon process which runs on each computational unit. It reports the CPU and memory utilization the node, and sends all video analysis tasks to

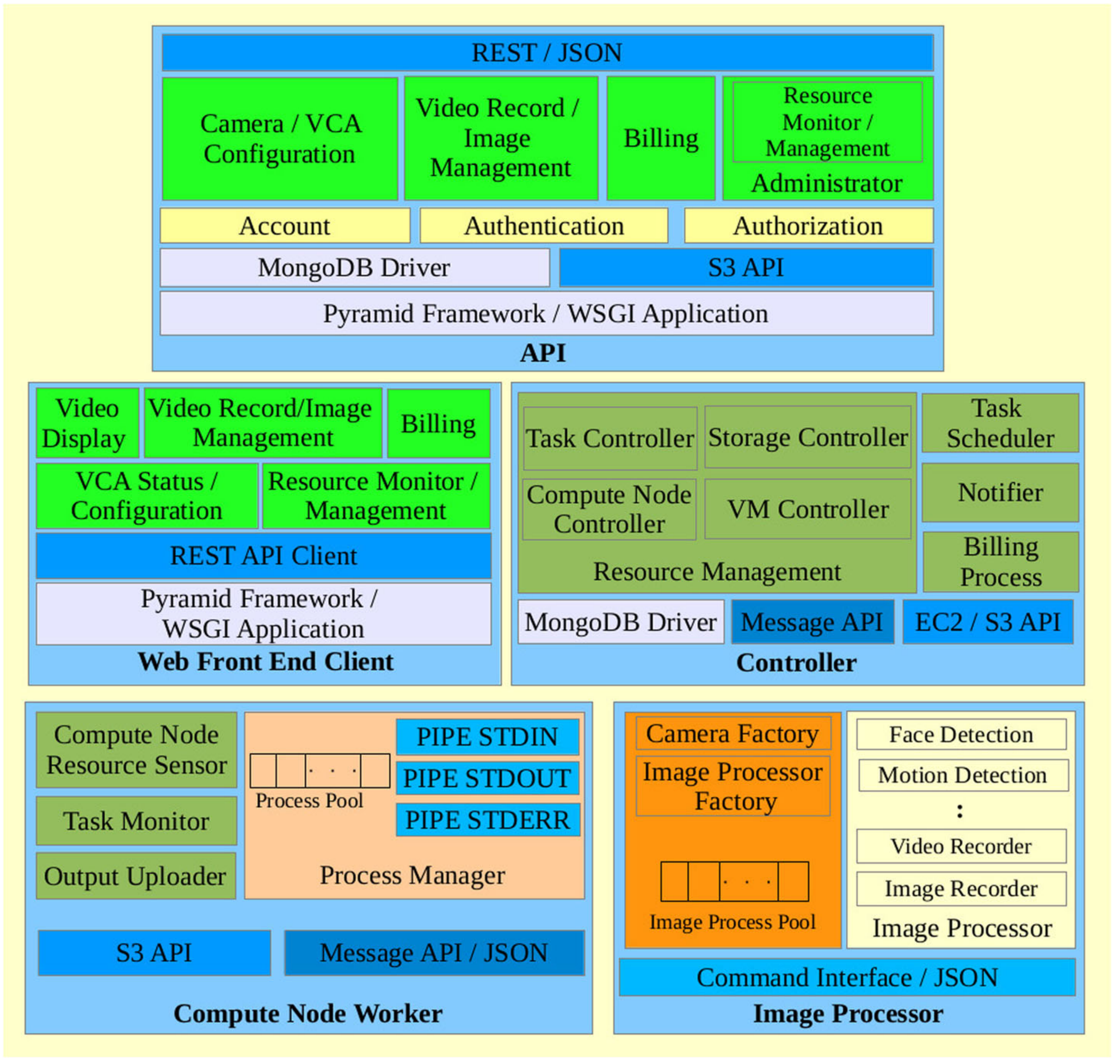


Fig. 1 Nokkhum VSaaS components

the controller via message passing [16]. It also provides an interface for managing the video processor module.

- The video processor component provides a set of tools including Video Content Analysis (VCA) and a video recorder utilizing the OpenCV library. The image component is started by a compute node worker, and its behavior controlled by the user. The user can compose a set of video analyses suitable for their requirements.
- The API server provides a REST style architecture via HTTP to manage a video processing task. The API server supports multiple types of clients and employs JavaScript Object Notation (JSON) [2] for data description formatting. The API server encloses several modules, such as accounting, authentication, authorization, camera configuration, video processing task management, and billing.
- A client can be implemented in several ways, but the default is a web front-end, which provides easy management of the modules located in the API server.

Our system overview of the Nokkhum components begins when the user composes a video processing configuration using the web front-end, and submits it to the Nokkhum API. After the API has validated the configuration, it stores a video processing command in the database, which the controller later reads. It finds a suitable Nokkhum compute node worker for the task using message-oriented middleware. The worker starts a Nokkhum processor with the configuration from the user which processes the video stream following the user's configuration. All output from the processor is immediately pushed to cloud storage. In addition, the user can manage the status of the video processing and playback video records via the web front-end.

Nokkhum's scalability benefits from its message-oriented middleware which enables it to distribute components to many servers. Earlier versions of the Nokkhum VSaaS only had a simple task scheduler based on examining the current resource workloads, without any specific or application-tailored resource prediction. As a result, the scheduler's resource decision process sometimes failed. The current Nokkhum versions enhances task scheduling with real video processing workloads, as explained later in this paper. It utilizes CPU and memory usage prediction, and decides whether to start a video processing task on the target computational unit.

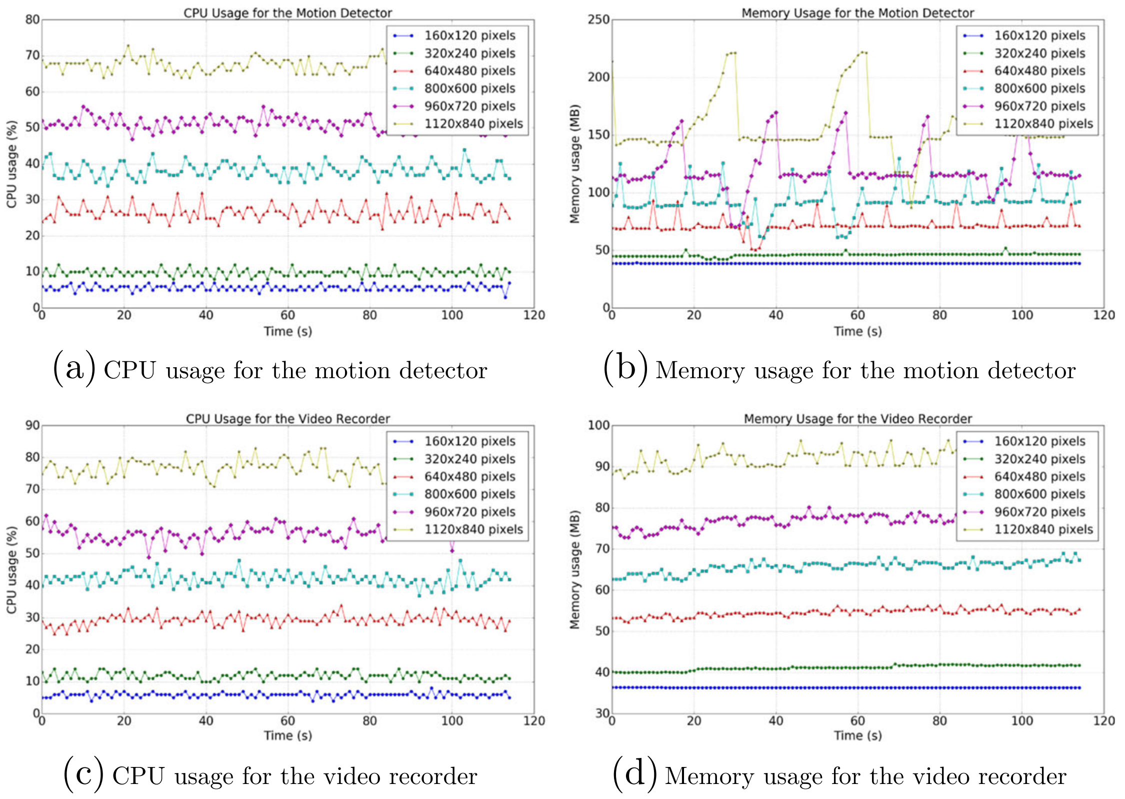
3.2 Video processing task exploration

Most VSS employ similar compute nodes specification, because it is easy to manage many video processing tasks with the same parameters. This simple scenario is insufficient for a VSaaS system with dynamic requirements involving many variables, such as frame rate, frame size, and video processor type. In addition, the cloud environment provides VM specifications according to its capacity and limitations. This means that the video processing task will consume different computational resources on different VM specifications, in a difficult to predict manner. Therefore, we utilize video processing task exploration to determine the computational resources consumed by the desired video processing task, depending on its parameters. The results are used in workload analysis described in Section 3.3, which become a heuristic for video task scheduling. There are two different ways to run video processing exploration: the first is manually by the administrator, and the second is automatically by the Nokkhum controller, which is described as follows:

- Video processing task exploration during system installation is manually run by the administrator, because he usually knows the number of compute nodes and their specifications. With manual execution, the administrator can collect all the resource usage information about the video processing tasks soon after the initialization. This means that the Nokkhum system does not need to collect any further resource consumption information for the task scheduling. Nokkhum can use the existing information for scheduling without repeating task exploration.

– The Nokkhum controller automatically runs video processing task exploration under two circumstances: the first is when there are no experimental results related to the compute node in the database. The second occurs when the node is idle for at least 100 % when the total percentage (more than 100 %) is calculated by 100 % multiplied by the number of CPU cores. In this case, the administrator does not have to decide to execute task exploration. Also, the scheduler has to focus on workload estimation in order to respond to the tasks in the queue. If it has to wait for experimental results from video exploration, tasks will probably have a long wait in the queue.

The exploration employs two types of video processing tasks: motion detection and video recording. A surveillance video from the VIRAT video dataset [14] was used in our experiments, with eight frame rates (1, 5, 7, 10, 15, 20, 25 and 30 FPS) and six frame sizes (160 × 120, 320 × 240, 640 × 480, 800 × 600, 960 × 720, and 1120 × 840 pixels). A total of 96 (2 × 8 × 6) test cases were executed, with each test running for two minutes, for a total of 192 minutes. This means that scheduling required another approach to deal with missing experimental results from the task exploration caused by the skipping of some tests to make the scheduling faster. Our approach will be presented in Section 4.2.



(a) CPU usage for the motion detector (b) Memory usage for the motion detector

(c) CPU usage for the video recorder (d) Memory usage for the video recorder

Fig. 2 Motion detector and video recorder resource consumption with various frame sizes

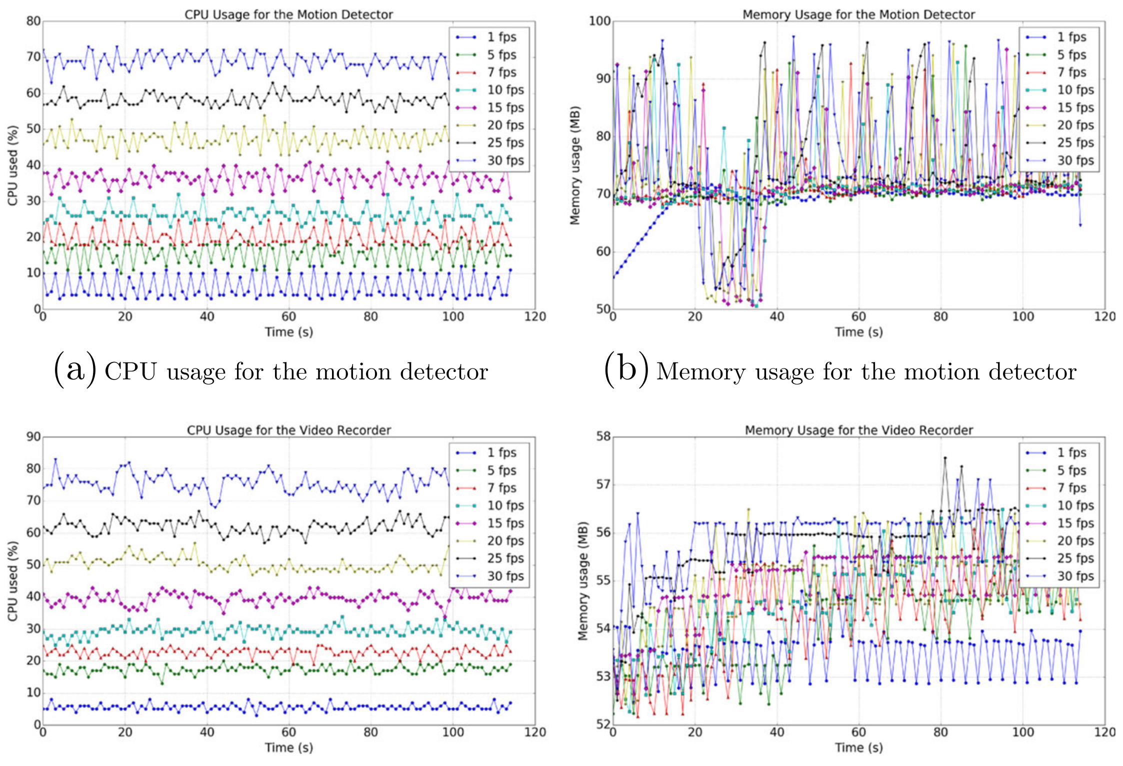
3.3 VCA and video recorder workloads analysis

VCA and video recorder workloads play an important role in driving VSS. Normally, the consumption of computing resources by the VCA and video recording tasks is a function of the frame rate and frame size. The CPU and memory utilizations of the VCA and video recording tasks vary when running on different machines. There are many multimedia systems, both VSSes and VSaaSes, which focus only on static workloads and homogeneous computing machines. However, the situation is rather different in real deployment environments, especially in Cloud infrastructures.

In general, the IaaS provider promises to provide all customers with the same VM template for the same charge. However, when the IaaS provider adds new hardware of a different specification, it is possible that it will provide a different certified VM template. Then, when the VSaaS provider deploys a VSaaS system on a Cloud IaaS, the efficiency of the VCA and video recording tasks scheduler will be affected. Therefore, we have studied the behaviors of the VCA and video recording tasks in terms of computing resource consumption to improve scheduling performance.

3.3.1 The VCA and video recorder

The two main factors affecting resource consumption are the video frame rate and frame size. It is difficult to study VCAs used in VSaaS, because of the many available types. The most popular is motion detection for filtering motion sequences, which are subsequently



(a) CPU usage for the motion detector

(b) Memory usage for the motion detector

(c) CPU usage for the video recorder

(d) Memory usage for the video recorder

Fig. 3 Motion detector and video recorder resource consumption at various frame rates

passed to the video recorder or used to notify the user. We focus on motion detection and video recording tasks, and initially handle resource consumption by fixing the frame rate and varying the frame size as in Fig. 2. The CPU and memory consumption for different frame rates are shown in Fig. 3. All the experiments were performed on a physical machine, an AMD FX(tm)-8320 eight-core processor with a 3.5 GHz CPU and 16 GB RAM.

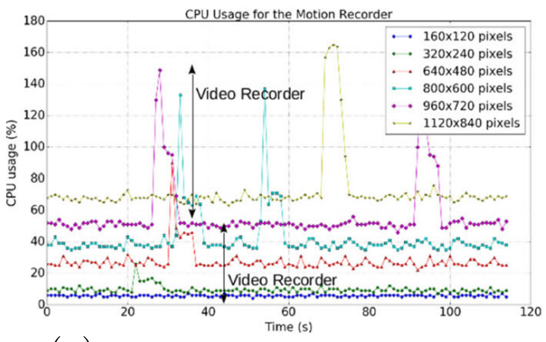
The video's frame rate is fixed at 10 frames per second (FPS) in Fig. 2, and the results show how an increases in the frame size, increase the consumed computational resources, namely CPU and memory usage, for both motion detection and video recording. Also, the frame size is fixed at 640 × 480 pixels and the frame rate of the motion detector and video recorder increased more computational resources are consumed as shown in Fig. 3. Further experiments that varied the frame rate caused the motion detector task to consume memory up to the maximum buffer allowed, which is expected for the Nokkhum processor.

Figures 2 and 3 show individual resource consumption, but the VSaaS video processing task can combine multiple VCAs for complex analysis. An example of VCA combination is the cooperation between the motion detector and video recorder for recording when motion sequences occur. Figure 4 shows the resource consumption of the motion recorder which employs a continuously running subtask to filter motion events before passing them to the video recorder. Therefore, the bottom line of each CPU usage graph in Fig. 5a is the resource consumption of the motion detector. The overshooting line represents the CPU resource consumption of the video recorder. Also, memory usage has similar characteristics to the motion detector on the bottom line.

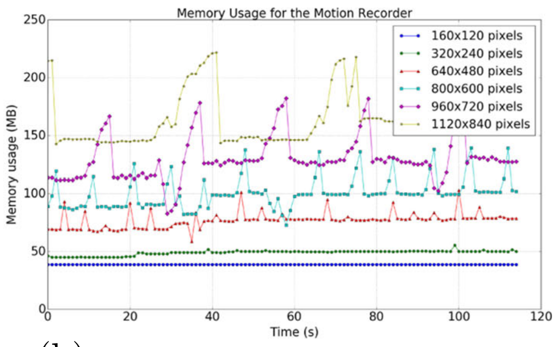
3.3.2 The VCA and video recorder for various computing specifications

This section presents resource consumption for the motion detector, video recorder, and motion recorder running on different physical machines shown in Fig. 5. All experiments utilize a fixed frame rate at 10 FPS and a frame size of 640x480 pixels. The testing machines used a x64 CPU with more than 4 GB of RAM.

In Fig. 5a, it is difficult to find any outstanding related factors to differentiate between the CPU usage characteristics in the motion detector when using different and complicated CPU architectural designs. When we divide the experimental results according to vendor (AMD and Intel), it seems that the CPU frequency is the only discriminator. The memory consumption of the machines in Fig. 5b imply similar utilizations.

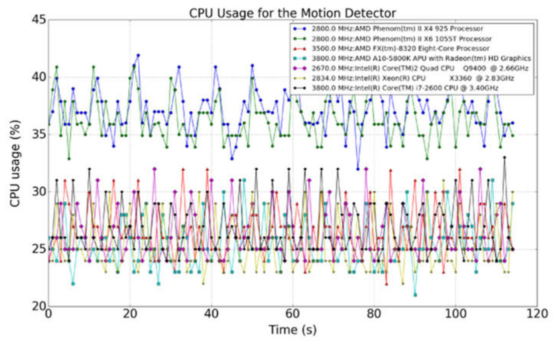


(a) CPU usage for the motion recorder

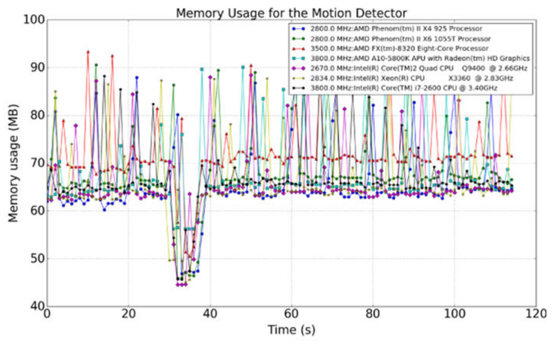


(b) Memory usage for the motion recorder

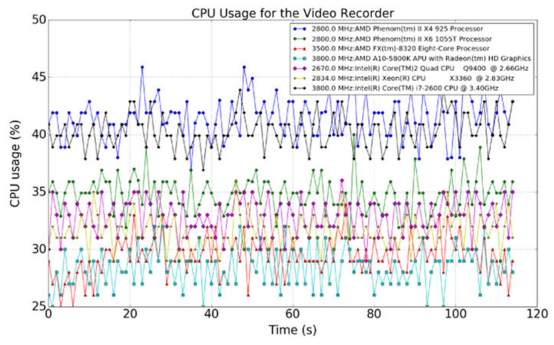
Fig. 4 Motion recorder resource consumption at various frame sizes



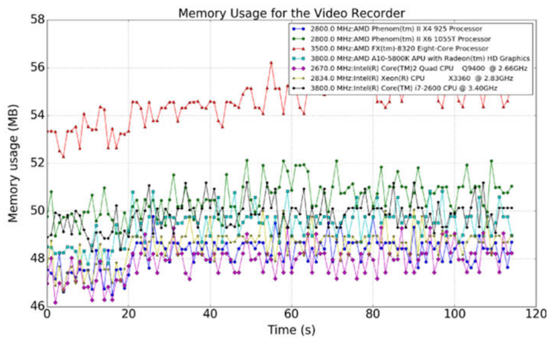
(a) CPU usage for the motion detector



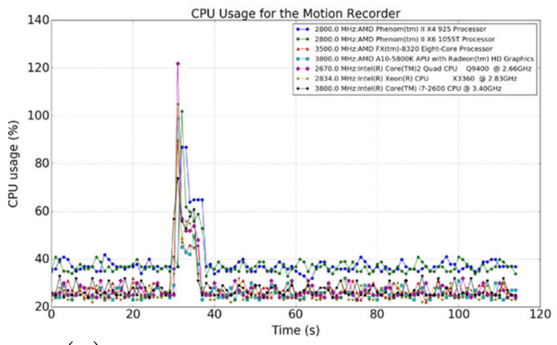
(b) Memory usage for the motion detector



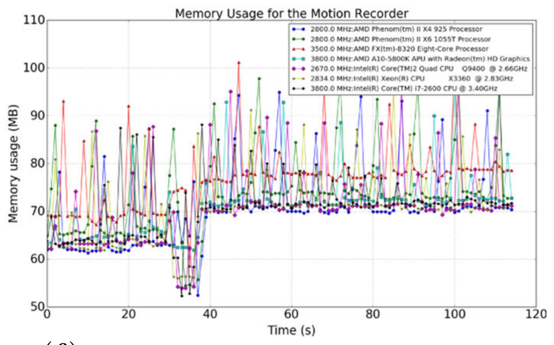
(c) CPU usage for the video recorder



(d) Memory usage for the video recorder



(e) CPU usage for the motion recorder



(f) Memory usage for the motion recorder

Fig. 5 Resources consumption for the motion detector, video recorder, and motion recorder running on different physical machines

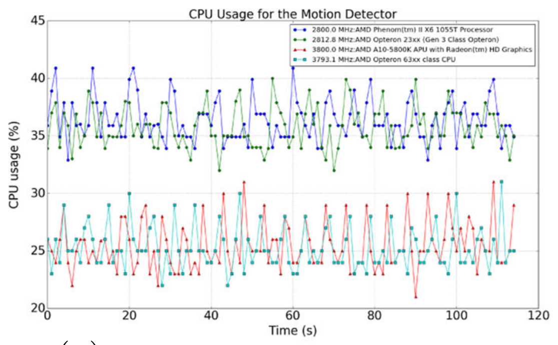
Video recorder resource consumption is shown in Fig. 5c and d. The video recorder resource consumption is different from the motion detector due to different CPU/memory architectural designs. For example, some desktop CPUs employ a special video codec chip set, to lower CPU utilization, but return the same frame size and rate.

The motion recorder is a combination of the motion detector and video recorder, and so inherits CPU and memory utilization characteristics from both. Its result are shown in Fig. 5e and f; the bottom line in Fig. 5e comes from the motion detector, and the overshooting line from the video recorder. However, it is difficult to identify which motion recorder characteristics are affected by the video recorder's CPU consumption, which depends on the CPU model and vendor.

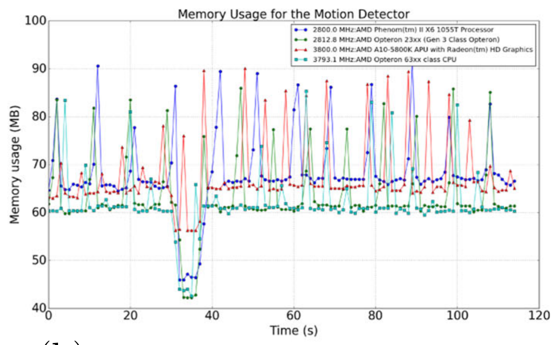
3.3.3 *Virtual and physical machines*

The Nokkhum VSaaS supports hybrid virtual and physical machines for the scalability of the cloud environment. Comparing results between VMs and physical machines helps the Nokkhum system to improve task scheduling performance for both machine types. Our results involving motion detection, video recording, and motion recording are shown in Fig. 6. All the experiments used a frame rate of 10 FPS and frame size of 640x480 pixels. The physical machines support virtualization technology with the KVM [8].

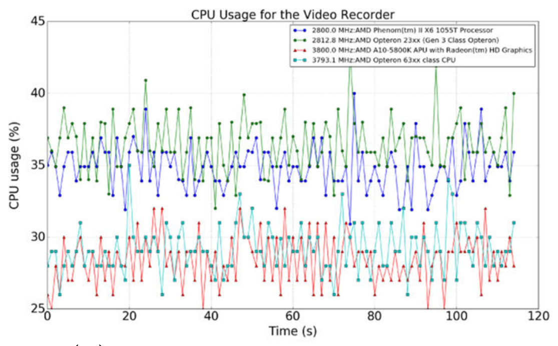
Figure 6a, c, and e present the CPU utilization for VMs and physical machines, showing that the pattern of results are similar for both, but that the VMs' CPU utilization is a little bit higher. Figure 6b, d, and f show that memory consumption for VM tasks is a little bit lower than that for physical machines.



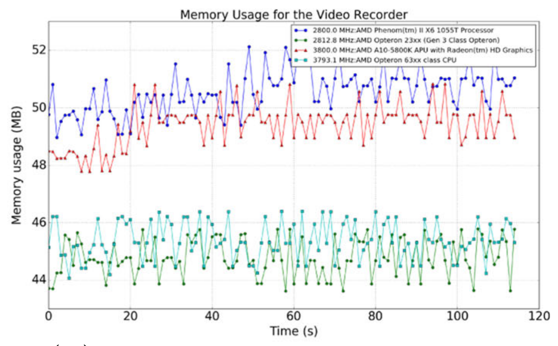
(a) CPU usage for the motion detector



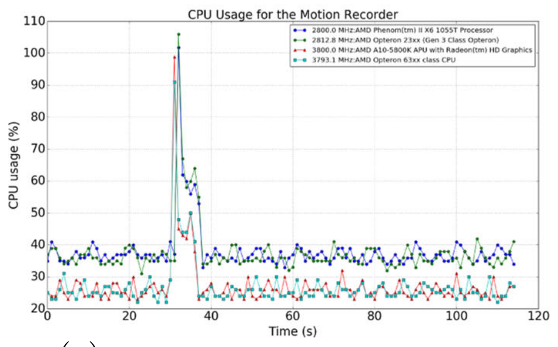
(b) Memory usage for the motion detector



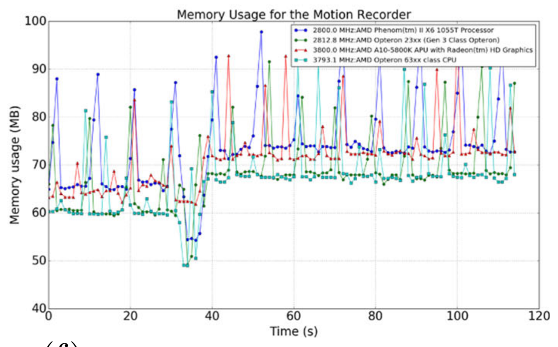
(c) CPU usage for the video recorder



(d) Memory usage for the video detector



(e) CPU usage for the motion recorder



(f) Memory usage for the motion recorder

Fig. 6 Resource consumption for the motion detector, video recorder, and motion recorder tasks running on different physical and virtual machines

3.3.4 Summary

Figures 2–6 show results from experiments using the frame rates 1, 5, 7, 10, 15, 20, 25, and 30 FPS, and frame sizes 160 × 120, 320 × 240, 640 × 480, 800 × 600, 960 × 720, and 1120x840 pixels. The resource consumption results become unclear when they combine frame rate and frame size, and it is difficult to discriminate which are from the VCA or the recording tasks. Moreover, the computing resource utilizations of the VCA and video recorder tasks are different when different machine specifications and hypervisors are employed.

The video processing workload characteristics show that different machine specifications, especially those affecting the CPU model, influence resource usage consumption. In order to support multiple machine specifications, the workload scheduler has to consider CPU and memory usage from the task exploration when assigning a task to a suitable compute node. The scheduler utilizes the exploration results by applying three resource usage criteria, as described in Section 3.4.

3.4 Resource usage criteria

Results from the same compute node specification and similar video processing tasks point in the same direction, but it is difficult to apply them to task scheduling. The scheduler needs to use resource usage criteria for computing resources estimation, based on criteria that involve the resource utilization of all the video processors which generally consume resources linearly. Our factors for approximating resource usage include the average data set, average minimum data set, and average maximum data set, and the proposed criteria are shown in Fig. 7.

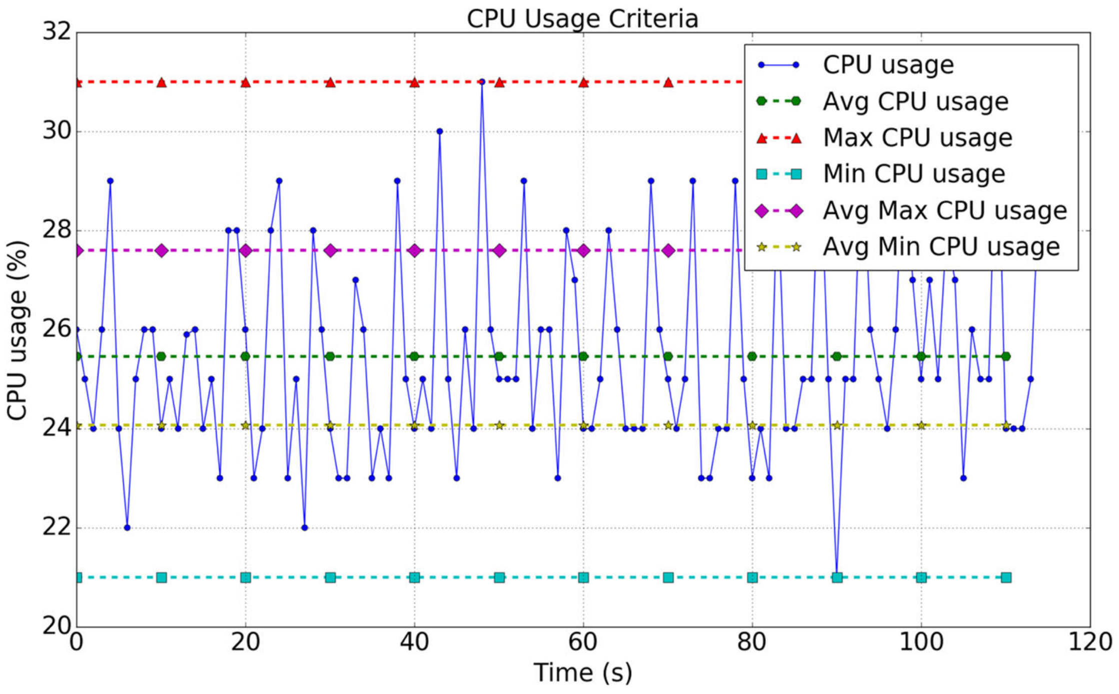
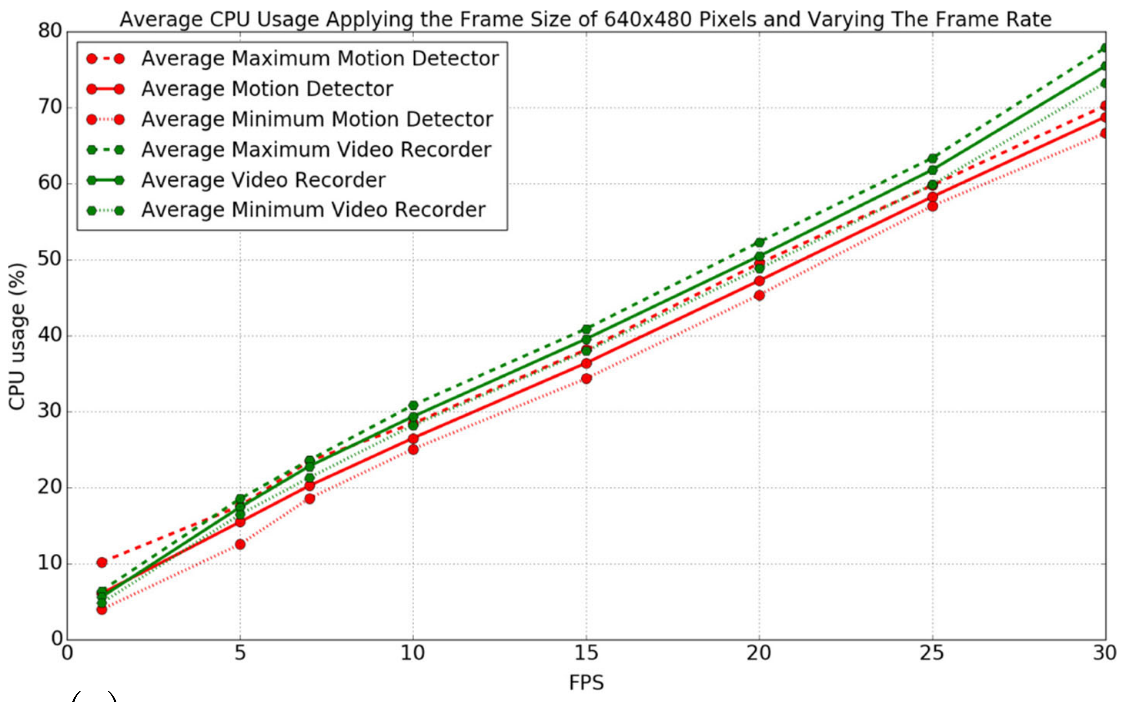
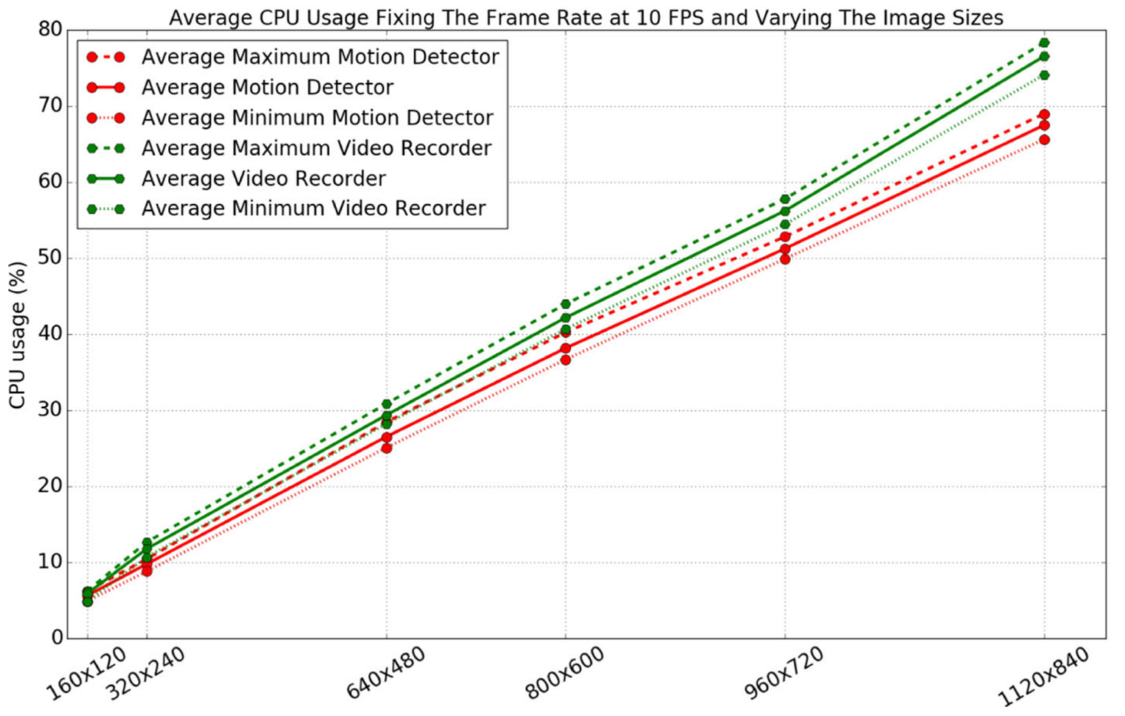


Fig. 7 Nokkhum resource criteria: the average data set, average minimum data set, and average maximum data set

The first criterion is the average data set, representing the data mean suitable for general CPU utilization events. The second criterion is the average minimum data set, which is the data average lower than the mean, and best for heavy CPU utilization although it may affect memory utilization. For example, when the CPU is busy or can not process the image



(a) CPU usage for the three resource usage criteria with varying frame rate



(b) CPU usage for the three resource usage criteria with varying frame size

Fig. 8 CPU usage for motion detection and video recording using the three resource usage criteria

on time, the memory utilization will increase. The last criterion is the average maximum data set, which is the data average higher than the mean. This ensures that the computing resources will be sufficient for the required video processing.

In short, VSaaS administrators must identify their system specification and choose a suitable criterion for highly efficient computing resource management. Section 5 presents experimental results applying our three different criteria and discuss suitable exploitations in real situations.

The three computing resource usage criteria can also be used to estimate the CPU and memory usage in video processing task scheduling. In this section, we only present the CPU usage as shown in Fig. 8, because it has the same characteristics as memory usage. Figure 8 shows CPU usage at several video frame rates and frame sizes, including motion detection and video recording in Fig. 8a and b. In Fig. 8, the average resources usage increases with increasing frame rates or frame sizes, and they can be plotted as straight lines with different slopes.

The video processing task exploration can take quite a long time to collect data from the compute nodes, but the execution time can be reduced by exploiting the CPU usage values in the criteria of Fig. 8. It appears that the average usage in is directly related to the frame size and frame rate. This can then be exploited by determining the linear slope and the sampling points of the video processing tasks from the lowest and highest frame rate or frame size. For example in Fig. 9, a frame rate of 1 and 30 FPS are used to approximate CPU usage, and the resulting motion detection and video recording equation are presented as (1) and (2). Table 1 shows the approximated CPU usages from the two equations compared to the real average CPU usages and their absolute errors.

$$\%CPU = \begin{array}{l} 3.90695652x + 2.1626087 \\ \text{for motion detection on the CPU AMD FX, 8 cores, 3.5 GHz} \end{array} \tag{1}$$

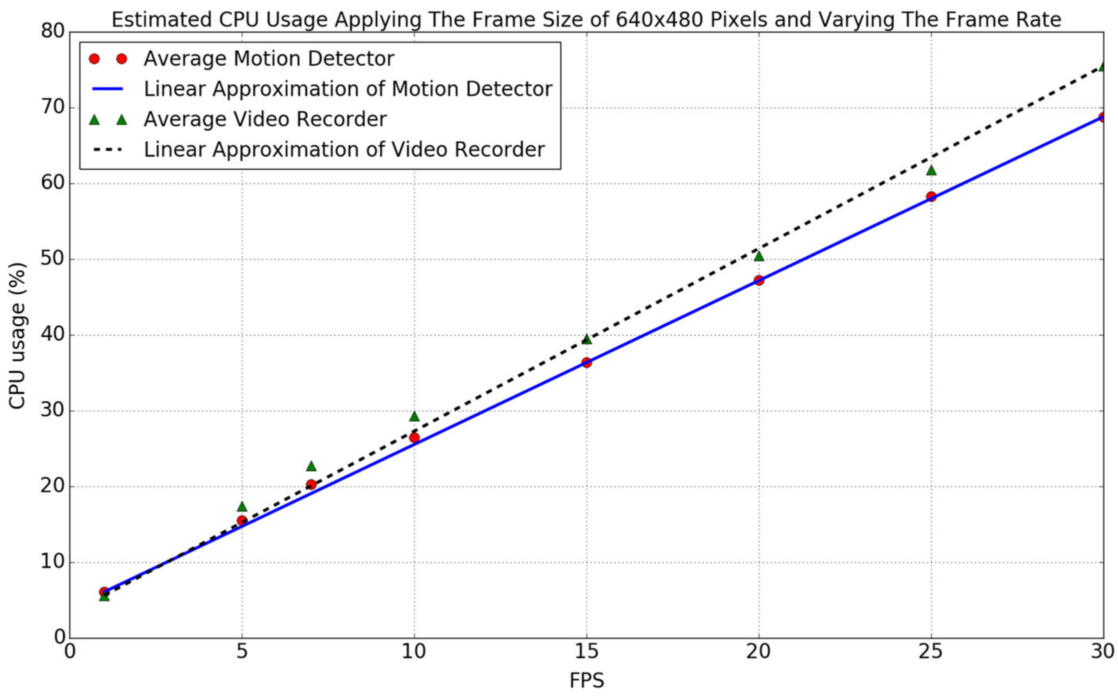


Fig. 9 Approximate CPU usage with a linear equation

Table 1 Comparison of the average CPU usage and its approximation for motion detection and video recording at different frame rates for the CPU AMD FX, 8 cores, 3.5 GHz

Frame rate	Motion detection			Video recorder		
	Average CPU usage	Approximation	Absolute error	Average CPU usage	Approximation	Absolute error
1	6.070	6.070	0.000	5.600	5.600	0.000
5	15.478	14.720	0.758	17.417	15.239	2.178
7	20.243	19.045	1.198	22.809	20.059	2.750
10	26.512	25.533	0.979	29.355	27.289	2.066
15	36.396	36.346	0.050	39.568	39.338	0.230
20	47.237	47.159	0.077	50.462	51.387	0.925
25	58.270	57.972	0.297	61.809	63.436	1.627
30	68.785	68.785	0.000	75.485	75.485	0.000

$$\begin{aligned} \%CPU = \; & 3.19016492x + 2.40983508 \\ & \text{for video recording on the CPU AMD FX, 8 cores, 3.5 GHz} \\ \text{where} \quad & x \text{ is the frame rate} \end{aligned} \tag{2}$$

Table 1 shows that the errors when applying (1) and (2) are small compared to the average CPU usages, with the maximum absolute error about 2.75 % of the usage. By applying these equations, we can reduce the execution time of the video processing exploration task from 96 test cases to eight ($2 \times 2 \times 2$). These come from two video processing types (motion detection and video recording), two frame rates (1 and 30 FPS) and two frame sizes (160×120 and 1120×840 pixels), taking a total 16 minutes for each machine specification. The linear equation model can only be applied to the same machine. Different specifications may result in different slopes due to different CPU architectures.

4 Nokkhum scheduling

Resource management and the video processing task scheduler are the key Nokkhum VSaaS modules for the availability and resource provision of tasks. This section describes a suitable task scheduler based on the real workloads presented in Section 3.

4.1 Scheduling overview

The task scheduler picks a video processing task from a queue, and asks the resource predictor to estimate the computing resource requirements for the task. After the predictor has returned an available computing node, the scheduler calls the task controller to spawn the video processing task. The Nokkhum scheduling process is shown in Fig. 10.

The Nokkhum scheduler evaluates which compute node worker is suitable to run a video analysis task across several steps. It gets a video processing task from the task queue, finds an available compute node worker, and passes it to the resource predictor. The predictor looks up the most suitable experimental workloads from the database and returns the best fit compute node worker to the scheduler. The scheduler then sends the video processing configuration and compute node worker information to the task controller. The

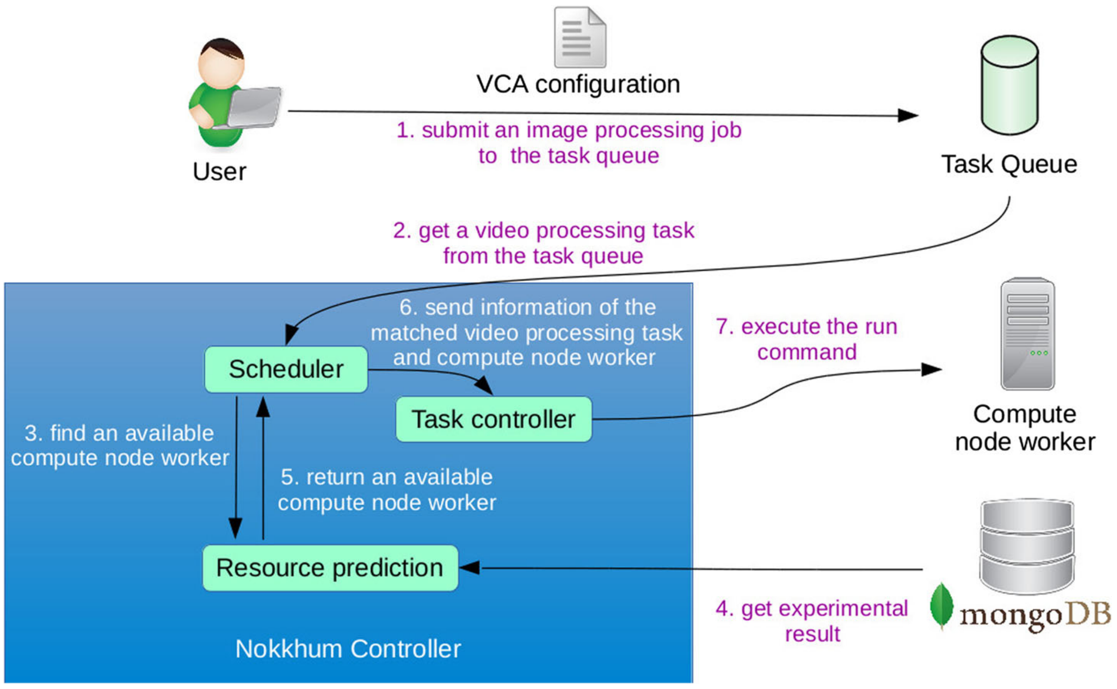

VCA configuration
User
1. submit an image processing job to the task queue
Task Queue
2. get a video processing task from the task queue
6. send information of the matched video processing task and compute node worker
7. execute the run command
Scheduler
Task controller
Compute node worker
3. find an available compute node worker
5. return an available compute node worker
Resource prediction
4. get experimental result
mongoDB
Nokkhum Controller

Fig. 10 Nokkhum scheduling

controller communicates with the compute node worker to run the video processing task. The Nokkhum resource decision process will be described in detail in Section 4.2.

Resource prediction is part of the compute node controller, and responds to the task scheduler by providing an available compute node worker suitable for the video processing task. It performs resource estimation based on data from the previously mentioned experimental results, especially the CPU and memory usage information. This estimation is used to determine which compute node is suitable for running the required video processing task. The experimental results can be collected from the task exploration, which can be run on any machine specification, and are stored in a database for the next round of scheduling. The resource prediction process is shown in Fig. 11.

4.2 Resource decision description

The Nokkhum resource decision module for placing a video processing task on a suitable computing unit is shown in Fig. 11. The module requires both the desired computation unit and the video processing configuration. The resource decision steps are described as follows.

Step 1) *Find matching experimental video analysis*

The resource decision module finds a video processing task with a matching video processing type, frame rate, and frame size from the previously allocated computation unit (of the same CPU vendor, model, and frequency). If it finds a result, it skips Steps 2 and 3 and goes to Step 5.

Step 2) *Find the closest adjacent experimental video analysis based on CPU frequency*

If the resource decision module cannot find an exactly matching experimental result in the database, it requests a new experimental result with an adjacent CPU frequency and ignores the CPU model of the desired compute node, but the

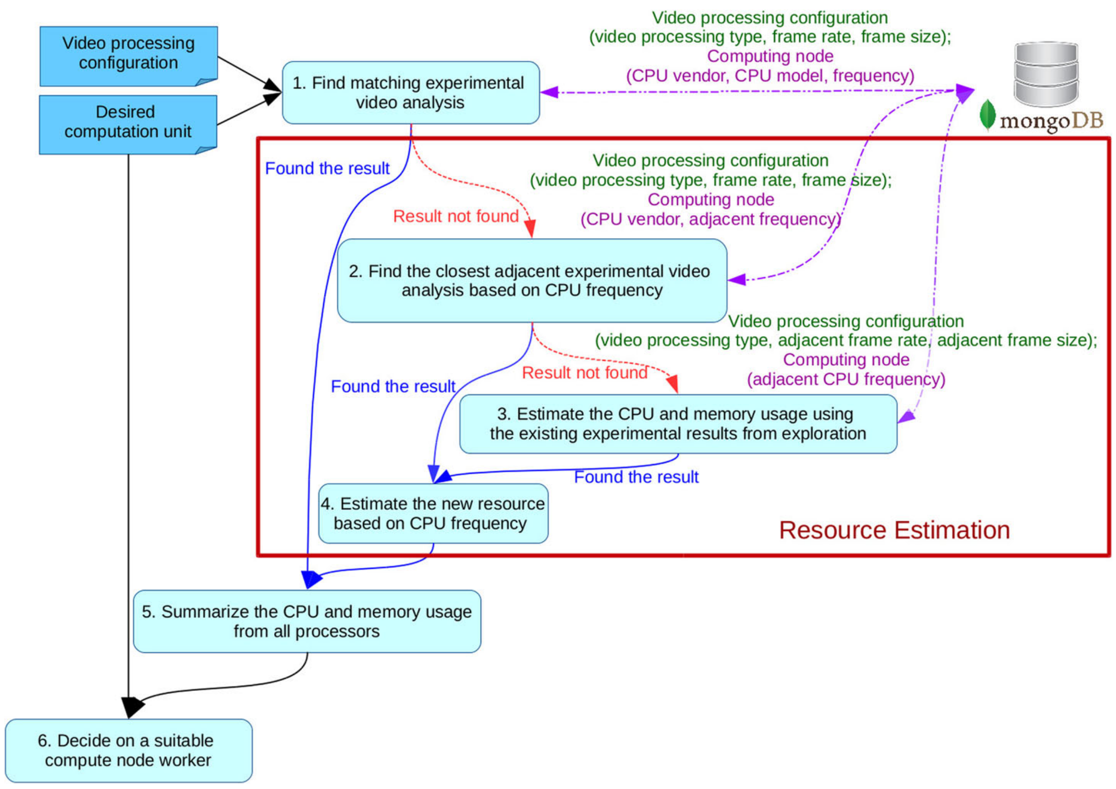

Video processing configuration
Desired computation unit
1. Find matching experimental video analysis
Video processing configuration (video processing type, frame rate, frame size); Computing node (CPU vendor, CPU model, frequency)
mongoDB
Found the result
Result not found
Video processing configuration (video processing type, frame rate, frame size); Computing node (CPU vendor, adjacent frequency)
2. Find the closest adjacent experimental video analysis based on CPU frequency
Video processing configuration (video processing type, adjacent frame rate, adjacent frame size); Computing node (adjacent CPU frequency)
Found the result
Result not found
3. Estimate the CPU and memory usage using the existing experimental results from exploration
Found the result
4. Estimate the new resource based on CPU frequency
Resource Estimation
5. Summarize the CPU and memory usage from all processors
6. Decide on a suitable compute node worker

Fig. 11 The Nokkhum resource management decision process.

video processing type, frame rate, and frame size must still match. The example in Fig. 12 shows that it always chooses the next available CPU with a higher frequency than the desired one. If a result is found, it will skip Step 3.

Step 3) *Estimate a CPU and memory usage using the experiment results*

This step looks for an experimental result in the database which has the same video processing type, and passes the result to Step 4. The result may not be accurate, and may not fit the task well, but it is better than scheduling with no estimation at all.

Step 4) *Estimate the new resource estimation based on CPU frequency*

Step 4 analyzes the experimental results shown in Fig. 5. The task's CPU usage depends on the CPU frequency. Higher clock frequency means a lower CPU usage by the task, depending on the CPU model and specification, although memory usage is rather consistent. The resource decision module estimates the CPU and memory usages from the experimental result equivalent to the desired computation

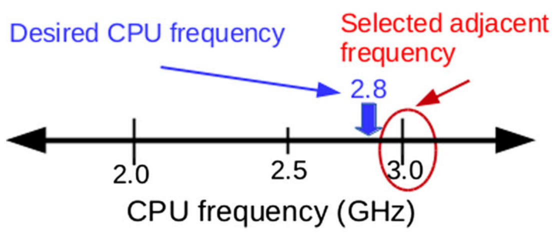


Fig. 12 Selecting a CPU frequency adjacent to the desired frequency

unit's CPU frequency. It applies a scaling factor (SF) as described by (3) to the CPU usage, unit as shown in (4).

$$SF = \begin{cases} \dfrac{F_{desired}}{F_{experiment}} & \text{if } F_{experiment} >= F_{desired} \\ \\ \dfrac{F_{experiment}}{F_{desired}} & \text{otherwise} \end{cases} \tag{3}$$

where SF is the CPU scaling factor;
$F_{desired}$ is the CPU frequency of the desired computation unit;
$F_{experiment}$ is the CPU frequency of the computation unit running the experiment.

Equation (3) calculates a scaling factor using the ratio of the compute node CPU frequency and the adjacent frequency determined from the experimental results in the database. It returns a different ratio according to whether the adjacent frequency is higher or lower than the CPU frequency of the desired compute node. The scaling factor will be used to estimate adjacent CPU usages, according to the resource usage criterion shown in (4) and (5). The estimated memory usage is set to the memory usage of the experiment, as in (5). The CPU and memory usages are derived from the resource usage criteria as presented in Section 3.4. The estimated CPU and memory usages ($CPU_{estimated}$ and $memory_{estimated}$) are used in the next step.

$$CPU_{estimated} = SF \times CPU_{experiment}[criterion] \tag{4}$$

where $CPU_{experiment}[criterion]$ is the CPU usage, collected from the experiment, selected by the $criterion$;
$criterion$ is a resource usage policy identified by the administrator.

$$memory_{estimated} = memory_{experiment}[criterion] \tag{5}$$

where $memory_{experiment}[criterion]$ is the memory usage from the experiment selected by applying the $criterion$.

Step 5) *Summarize the CPU and memory usages from all processing tasks*

The Nokkhum video processor components are sequentially connected to each other. The resource decision module summarizes the CPU and memory usages of all the video tasks from the previous steps to aid decision making in the final step. For example, a motion recorder consists of a motion detector and a video recorder; the resource decision module sums the CPU and memory usages from both processors for consideration in the next step.

Step 6) *Decide on a suitable compute node worker*

The final step decides which computing node is suitable. The resource decision module acquires the real workload of the compute node and approximates the current capacity of the running task. The approximated current resource usage of the target compute node and the task information from Step 5 are used to decide if it is suitable. The module will return an appropriate compute node to the task scheduler which can start the task. The processes are described by (6), (7) and (8).

$$
\begin{aligned}
&T_{CPU} = \textstyle\sum_{n=1}^{N} CPU_{e_{(n)}} + CPU_{e_p} \\
&\text{where} \quad T_{CPU} \text{ is the summation of } CPU_{estimated} \text{ of all} \\
&\qquad \text{the video processor tasks running on the compute node } C; \\
&\qquad CPU_{e_{(n)}} \text{ is the estimated CPU usage of} \\
&\qquad \text{the video processing task } n; \\
&\qquad CPU_{e_p} \text{ is } CPU_{estimated} \text{ of the requested video processing task} \\
&\qquad p \text{ that will be executed by the compute node } C; \\
&\qquad N \text{ is the number of video processing tasks} \\
&\qquad \text{contained in the compute node } C.
\end{aligned}
\tag{6}
$$

$$
\begin{aligned}
&T_{memory} = \textstyle\sum_{n=1}^{N} memory_{e_{(n)}} + memory_{e_p} \\
&\text{where} \quad T_{memory} \text{ is the summation of } memory_{estimated} \text{ of all} \\
&\qquad \text{the video processor tasks running on the compute node } C; \\
&\qquad memory_{e_{(n)}} \text{ is the estimated memory usage of} \\
&\qquad \text{the video processing task } n; \\
&\qquad memory_{e_p} \text{ is } memory_{estimated} \text{ of the requested} \\
&\qquad \text{video processing task } p \text{ that will be executed by} \\
&\qquad \text{the compute node } C.
\end{aligned}
\tag{7}
$$

$$
\begin{aligned}
&D(C) = \begin{cases} True & \text{if } T_{CPU} < C.cpu\_core * 100, \\ & \text{and } T_{memory} < C.total\_memory \\ False & \text{otherwise} \end{cases} \\
&\text{where} \quad D(C) \text{ is the decision function for} \\
&\qquad \text{the compute node qualification;} \\
&\qquad C \text{ is the desired compute node.}
\end{aligned}
\tag{8}
$$

Equations (6), (7) and (8) determine a compute node suitable for executing the video processing task. T_{CPU} and T_{memory} are summations of the estimated CPU and memory usages ($CPU_{estimated}$ and $memory_{estimated}$) of all the video processing tasks (N) containing the desired compute nodes and selected video processing tasks (CPU_{e_p} and $memory_{e_p}$). The decision function (D) of compute node C employs T_{CPU} and T_{memory} to determine whether the compute node C can run the video processing task p. T_{CPU} must be lower than the number of cores multiply by 100 % and T_{memory} must be lower than the maximum memory of the desired compute node. If there is no compute node with the desired specification, the decision function will return $False$. The resource decision algorithm for placing tasks is also presented as pseudocode in Algorithm 1.

Algorithm 1 presents three functions for task scheduling: GetSuitableComputeNode, EstimateComputingResources, and GetVPTExperiment.GetSuitableComputeNode validates whether a compute node is suitable for starting a task using a compute node information and a video processing configuration. It checks an available compute node using EstimateComputingResources which gathers approximated computing resources from the current task running on the candidate compute node. The approximated computing resource is compared with the estimated video processing resource taken from the video processing

Algorithm 1 Nokkhum task scheduling

```
input  : A set of available compute nodes
input  : A VCA configuration
output: A suitable compute node
Function GetSuitableComputeNode (compute-nodes, p)
    foreach c in compute-nodes do
        cEstimateCPU ← 0; cEstimateMemory ← 0;
        foreach cp in c.processors do
            subcpu, submemory= EstimateComputingResources (c, cp);
            cEstimateCPU = cEstimateCPU + subcpu;
            cEstimateMemory = cEstimateMemory + submemory;
        end
        processorCPU, processorMemory = EstimateComputingResources (c, p);
        if processorCPU + cEstimateCPU < 100 × c.cpucore and processorMemory +
        cEstimateMemory < c.maxmeory then return c ;
    end
    return None
End
Function EstimateComputingResources (c, p)
    totalCPU ← 0; totalMemory ← 0;
    foreach p in p.processors do
        experiment = GetVPTExperiment (c, p);
        totalCPU ← totalCPU + experiment.cpu[criterion];
        totalMemory ← totalMemory + experiment.memory[criterion];
        if "processors" in p then
            subcpu, submem = EstimateComputingResources (c, p.processors);
            totalCPU ← totalCPU + subcpu; totalMemory ← totalMemory + submem;
        end
    end
    return totalCPU, totalMemory
End
Function GetVPTExperiment (c, p)
    cpu ← c.CPU;
    experiment ← GetVPTExFromDatabase (cpu.vender, cpu.model, cpu.frequency,
    p.vcatype, p.framerate, p.imagesize);
    if experiment == None then
        experiment ← GetVPTExFromDatabase (cpu.vender, ~cpu.frequency, p.vcatype,
        p.framerate, p.imagesize);
        // ~cpu.frequency is the selected CPU frequency that is an adjacent target CPU
        frequency
    end
    if experiment == None then
        experiment ← GetVPTExFromDatabase (~cpu.frequency, p.vcatype, p.framerate,
        ~p.imagesize);
        // ~p.imagesizes is the selected image size that is an adjacent target image size
    end
    return experiment
End
```

task configuration. EstimateComputingResources summarizes the CPU and memory usage sums from all the video processing configurations using GetVPTExperiment. It queries the experimental VCA from the database following the steps described in Fig. 11 and Section 4.2.

The scheduler orders the tasks based on real resource usage, but it is difficult to evaluate the computing resources for such data. Therefore, the decision algorithm is driven by resource utilization, so the system administrator can guide the scheduler with the resource usage criteria described in Section 3.4.

5 Experimental results and discussion

This section presents experimental results when applying the Nokkhum scheduler to different resource configurations and machine specifications.

5.1 Experimental results

Table 2 shows physical machine specifications including their codes used in Tables 3 and 4, while Table 3 shows the VM specifications based on the physical machines from Table 2. Table 4 presents the resource usage for the machines described in Tables 2 and 3, and includes three resource usages criteria: the average (Avg), average maximum (AMax), and average minimum (AMin) of CPU and memory usage. Each run involves two video processing tasks, a video recorder (VR) and motion detector (MD). These results are used as example models for determining task scheduling.

Figures 13 and 14 present task scheduling results based on a physical machine (AMD-3.5-PM) and a VM (AMD-3.5-VM), and includes the number of video tasks contained in both types. AMD-3.5-VM is a VM running on the AMD-3.5-PM. Both are very similar, but their CPU specifications are different as shown in Tables 2 and 3. The important specification parameters are the number of cores and the maximum memory. The video processing tasks used for scheduling are a video recorder (VR) and motion recorder (MR), with MR combining motion detector and video recorder tasks. This means that the MR computing resource usage is a summation of both tasks.

Scheduling results are shown in Figs. 13 and 14. One of the experiments in Fig. 14 involves a video recorder, using a 640x480 pixel size, and a frame rate of 10 FPS. The scheduler can assign six video recorder tasks to this VM based on applying the average CPU usage criterion. The video recorder task has a CPU usage of about 187.59 % and consumes 272.10 MB of 200 % (2×100), where the maximum memory usage is 2000MB and the maximum CPU usage is 200 %. Therefore, there is some space left for placing another task on the VM. For instance, the scheduler cloud add a video recorder VM, with a video size of 320x240 pixels and frame rate of 10 FPS which would consume 199.39 % CPU usage and 313.28 MB memory. All these workloads are within the machine specification, and so can all run smoothly together.

Figures 13 and 14 show that the number of tasks placed on a compute node worker depends on the frame rate and size, for both the video and motion recorder. The number of tasks decreases when the frame rate and size are increased. This characteristic appears in both the physical and virtual machines, and also directly affects memory consumption. Therefore, we can consider the number of tasks in the analysis as reflecting memory usage. However, the CPU consumption by the video and motion recorder affects the CPU usage

Table 2 Physical machines used in the experiments

PM Code	CPU			Total Memory (GB)	Total Disk (GB)
	Model	Frequency (Hz)	Cores		
AMD-2.8-PM	AMD Phenom(tm) II X6 1055T Processor	2800	6	8.11	98.29
AMD-3.5-PM	AMD FX(tm)-8320 Eight-Core Processor	3500	8	16.55	98.29
AMD-3.8-PM	AMD A10-5800K APU with Radeon(tm) HD Graphics	3800	4	16.27	1850.37
Intel-2.6-PM	Intel(R) Core(TM)2 Quad CPU Q9400 @ 2.66GHz	2670	4	4.15	285.31
Intel-2.8-PM	Intel(R) Xeon(R) CPU X3360 @ 2.83GHz	2834	4	4.15	226.78
Intel-3.4-PM	Intel(R) Core(TM) i7-2600 CPU @ 3.40GHz	3800	4	4.13	473.86

Table 3 Virtual machines used in the experiments

VM Code	PM Code	CPU			Total Memory (GB)	Total Disk (GB)
		Model	Frequency (Hz)	Cores		
AMD-2.8-VM	AMD-2.8-PM	AMD Opteron 23xx (Gen 3 Class Opteron)	2812.792	2	2.10	20.09
AMD-3.5-VM	AMD-3.5-PM	AMD Opteron 63xx class CPU	3515.784	2	2.10	20.09
AMD-3.8-VM	AMD-3.8-PM	AMD Opteron 63xx class CPU	3793.102	2	2.10	20.09
Intel-2.6-VM	Intel-2.6-PM	Intel Core 2 Duo P9xxx (Penryn Class Core 2)	2666.362	2	2.10	20.09
Intel-2.8-VM	Intel-2.8-PM	Intel Core 2 Duo P9xxx (Penryn Class Core 2)	2833.53	2	2.10	20.09
Intel-3.4-VM	Intel-3.4-PM	Intel Xeon E312xx (Sandy Bridge)	3391.448	2	2.10	20.09

Table 4 Video processing workload resource usage results

Code	Processor	Video		CPU Usage (%)			Memory Usage (MB)		
		Size	FPS	Avg	AMax	AMin	Avg	AMax	AMin
AMD-2.8-PM	Motion detector	640 × 480	10	36.38	38.13	35.08	68.02	78.22	65.04
	Video recorder	640 × 480	10	34.85	35.77	33.30	50.57	51.17	49.96
AMD-3.5-PM	Motion detector	320 × 240	10	9.80	10.47	8.83	46.08	46.82	45.01
		320 × 240	15	12.44	13.47	11.36	46.16	46.98	44.10
		640 × 480	10	26.51	28.45	25.08	71.70	79.00	69.56
		640 × 480	15	36.40	38.15	34.35	71.90	80.58	68.70
	Video recorder	320 × 240	10	11.80	12.67	10.71	41.18	41.69	40.70
		320 × 240	15	16.50	17.46	15.55	41.51	41.84	40.86
		640 × 480	10	29.35	30.86	28.20	54.61	55.36	53.99
		640 × 480	15	39.57	40.90	37.95	54.98	55.50	54.27
AMD-3.8-PM	Motion detector	640 × 480	10	25.45	27.60	24.07	67.26	79.94	64.43
	Video recorder	640 × 480	10	28.42	29.86	26.95	49.40	49.85	48.63
Intel-2.6-PM	Motion detector	640 × 480	10	26.26	29.19	24.92	65.49	77.76	62.91
	Video recorder	640 × 480	10	32.89	33.93	31.23	47.89	48.33	47.17
Intel-2.8-PM	Motion detector	640 × 480	10	25.22	28.40	23.83	65.65	77.56	62.84
	Video recorder	640 × 480	10	31.55	32.75	30.45	48.57	48.97	47.79
Intel-3.8-PM	Motion detector	640 × 480	10	26.93	29.32	25.28	66.81	78.23	64.40
	Video recorder	640 × 480	10	40.28	41.61	39.19	49.84	50.24	49.20
AMD-2.8-VM	Motion detector	640 × 480	10	35.75	37.22	34.31	63.25	75.05	60.13
	Video recorder	640 × 480	10	36.52	37.96	34.91	44.79	45.51	44.37
AMD-3.5-VM	Motion detector	320 × 240	10	9.97	10.68	8.49	40.14	40.42	37.18
		320 × 240	15	13.13	14.30	12.58	39.94	40.45	35.96
		640 × 480	10	25.24	27.60	23.99	61.79	70.98	59.86
		640 × 480	15	33.85	35.04	31.80	62.72	73.89	59.46
	Video recorder	320 × 240	10	13.37	14.36	12.62	35.92	36.11	35.76
		320 × 240	15	17.72	18.55	16.56	35.70	35.82	35.57
		640 × 480	10	31.27	32.72	30.11	45.35	46.11	44.93
		640 × 480	15	41.20	42.79	39.93	45.64	46.36	45.26
AMD-3.8-VM	Motion detector	640 × 480	10	25.29	27.40	24.16	61.81	71.87	59.82
	Video recorder	640 × 480	10	29.19	30.88	28.18	45.37	46.02	44.94
Intel-2.6-VM	Motion detector	640 × 480	10	40.46	42.36	39.05	64.51	73.43	60.92
	Video recorder	640 × 480	10	34.74	35.95	33.46	45.28	45.93	44.83
Intel-2.8-VM	Motion detector	640 × 480	10	36.98	39.43	35.09	63.10	73.28	59.35
	Video recorder	640 × 480	10	32.92	34.27	31.59	45.81	46.56	45.36
Intel-3.8-VM	Motion detector	640 × 480	10	26.38	29.18	25.00	63.12	75.64	60.48
	Video recorder	640 × 480	10	39.94	41.58	38.88	45.52	46.25	45.09

Avg: Average

AMax: Average Maximum

Amin: Average Minimum

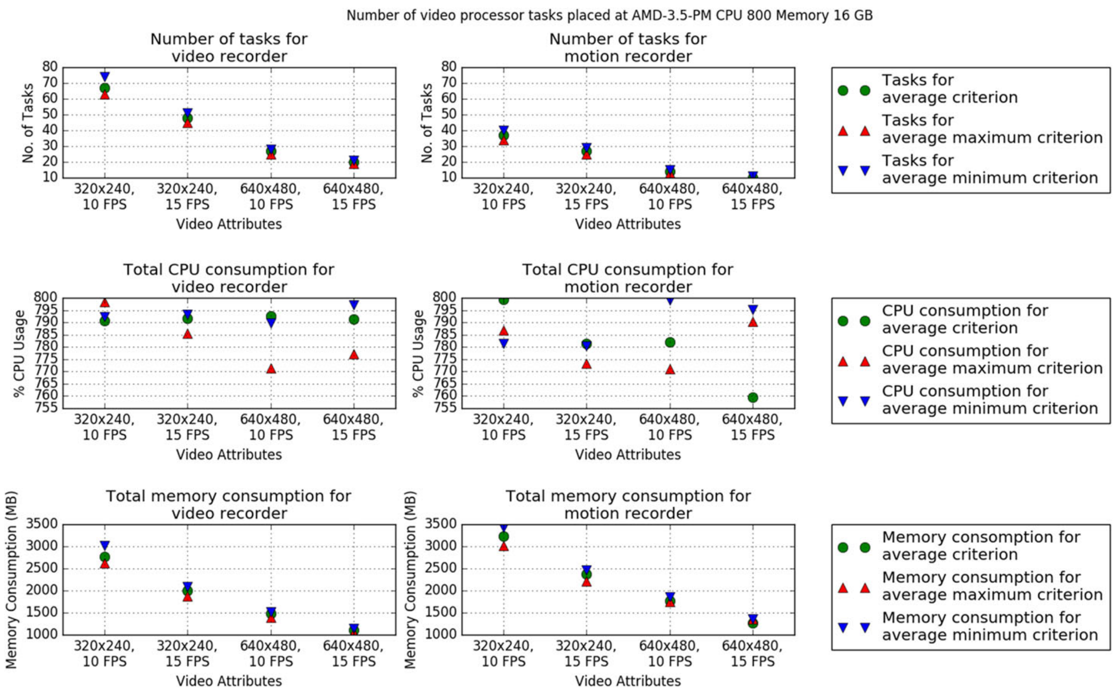


Fig. 13 Number of tasks, the CPU and memory consumption in the physical machine (AMD-3.5-PM)

percentage, depending on the number of tasks and the compute node capacity. This indicates that CPU consumption is the most significant factor for workload scheduling.

5.2 Discussion

The Nokkhum scheduler assigns a video processing task to a compute node by examining adjacent experiment results from the exploration test in order to estimate resource usage.

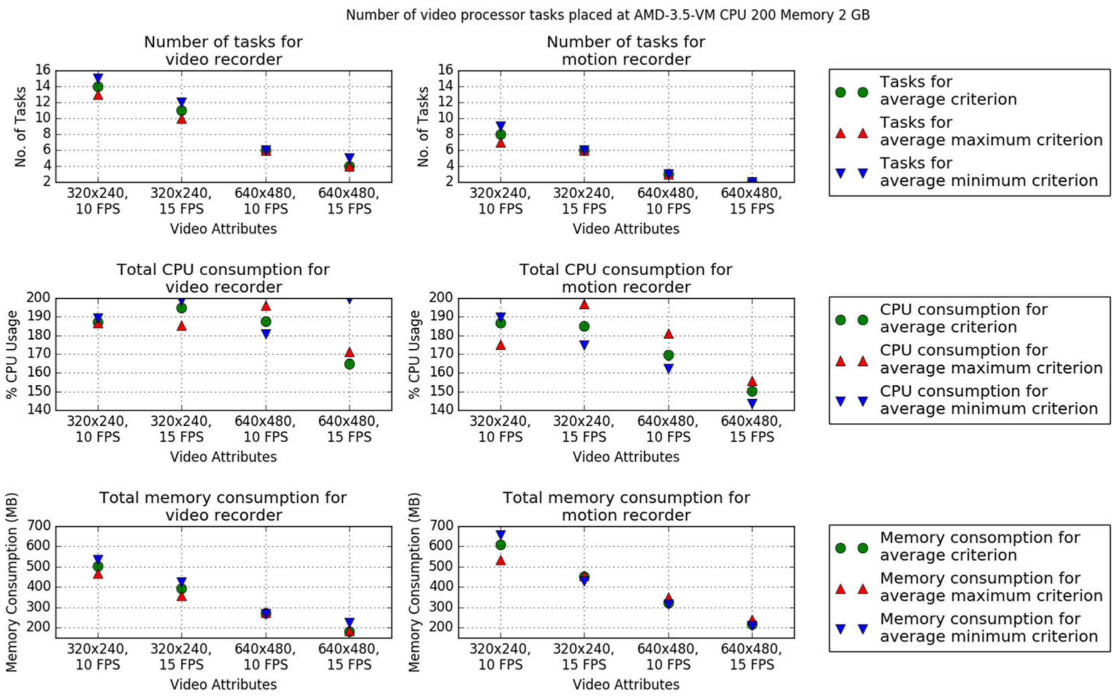


Fig. 14 Number of tasks, the CPU and memory consumption in the virtual machine (AMD-3.5-VM)

However, if the scheduler does not have enough resource usage information, it may assign tasks that exceed the compute node's capacity. To avoid this the Nokkhum scheduler compares real resource usage and the estimated resource usage. This is used to prevent system failure until better exploration results are received.

$$\begin{aligned}
N_{CPU} &= \left\lfloor \frac{C.cpu_{core} \times 100}{CPU_{estimated}} \right\rfloor \\
N_{memory} &= \left\lfloor \frac{C.total\_memory}{memory_{estimated}} \right\rfloor \\
N_{total} &= \begin{cases} N_{CPU} & \text{if } N_{CPU} < N_{memory}, \\ N_{memory} & \text{otherwise} \end{cases}
\end{aligned}$$

where N_{CPU} is the number of video processing tasks divided by CPU usage;
N_{memory} is the number of video processing tasks divided by memory usage;
N_{total} is the number of video processing tasks based on a comparison of N_{CPU} and N_{memory}.

(9)

We can estimate the number of video processing tasks per compute node by transforming (6), (7) and (8) into (9). It shows that the number of video processing tasks depends on the video frame rate and frame size per compute node. Also, the CPU information is given more weight than the memory. The VSaaS administrator can predict the number of compute nodes for their system by applying (9), and so plan the system deployment budget.

6 Conclusions

This paper has described video processing workload analysis and resource estimation for workload scheduling on the Nokkhum VSaaS, which handles the computing resource consumption of video processing tasks by interlacing the percentages of CPU and memory usage with frame rates and sizes. Based on experiments run on both physical and virtual machines, we developed video processing that stably consumes computing resources. A typical combination of video processors incurs different characteristics which makes it difficult to model CPU and memory usage. Therefore, our approach to video processing task scheduling utilizes task exploration which collects and records CPU and memory usage in a database. Later the scheduler uses this data to estimate required computing resources for a new task, to determine which compute node is most suitable for it. Task exploration is initiated when there is no video processing information for a computing unit in the database, and the unit is idle.

We have also described a method for determining video resource usage when there is no exactly matching information in the database. While waiting for exploration results, the scheduler estimates the new video processing task's resource usage from existing experimental results available in the database. The estimation process employs a scaling factor to adjust the CPU usage as a heuristic in the scheduling process. This resource estimation process enables the scheduler to immediately place a task on an appropriate compute node without waiting for exploration results which may take a long time to be produced.

In addition, we have offered criteria (the average, average maximum, and average minimum resource usage) as guidelines for identifying resource usage policy. These three criteria allow the administrator to adjust the requested task to the desired compute node. Average resource usage fits all task consumption relative to the computing unit. Average maximum can force the compute node towards an excessive workload and average minimum resource usage can reduce consumption to allow space to left over.

This paper describes a technique to decide on how to allocate video processing tasks to compute nodes which is useful for estimating the number of servers and budget planning. This will give VSaaS system administrators more insight into their system's behavior and enable them to optimize resource usage.

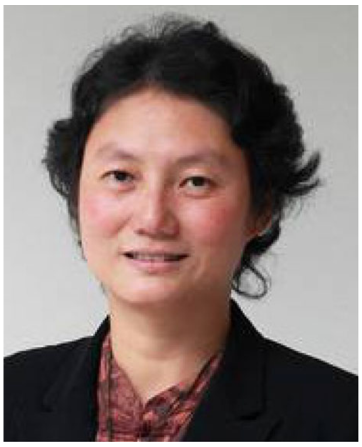
Acknowledgments The authors are grateful for financial support from the Thailand Research Fund and Prince of Songkla University through the Royal Golden Jubilee Ph.D. Program (Grant No. PHD/0047/2552).

References

1. Alamri A, Hossain MS, Almogren A, Hassan MM, Alnafjan K, Zakariah M, Seyam L, Alghamdi A (2015) QoS-adaptive service configuration framework for cloud-assisted video surveillance systems. Multimed Tools Appl:1–16. doi:10.1007/s11042-015-3074-7
2. Crockford D (2006) Json: the fat-free alternative to xml. In: Proceeding of XML, vol 2006
3. Fielding RT, Taylor RN (2002) Principled design of the modern web architecture. ACM Trans Int Technol 2(2):115–150. doi:10.1145/514183.514185
4. Hossain MA, Hossain MA (2014) Framework for a cloud-based multimedia surveillance system, framework for a cloud-based multimedia surveillance system. Int J Distrib Sensor Netw e135(257):2014. doi:10.1155/2014/135257
5. Hossain M, Hassan M, Qurishi M, Alghamdi A (2012) Resource allocation for service composition in cloud-based video surveillance platform. In: 2012 IEEE International conference on multimedia and expo workshops (ICMEW), pp 408 –412. doi:10.1109/ICMEW.2012.77
6. Hossain MS (2013) QoS-aware service composition for distributed video surveillance. Multimed Tools Appl 73(1):169–188. doi:10.1007/s11042-012-1312-9
7. Karimaa A (2011) Video surveillance in the cloud: dependability analysis. In: The Fourth international conference on dependability. IARIA XPS Press, pp 92–95. https://www.thinkmind.org/download.php?articleid=depend_2011_4_20_40042
8. Kivity A, Kamay Y, Laor D, Lublin U, Liguori A (2007) kvm: the linux virtual machine monitor. In: Proceedings of the linux symposium, vol 1, pp 225–230
9. Lee J, Feng T, Shi W, Bedagkar-Gala A, Shah S, Yoshida H (2012) Towards quality aware collaborative video analytic cloud. In: 2012 IEEE 5th international conference on cloud computing (CLOUD), pp 147–154. doi:10.1109/CLOUD.2012.141
10. Limna T, Tandayya P (2014) A flexible and scalable component-based system architecture for video surveillance as a service, running on infrastructure as a service. Multimed Tools Appl:1–27. doi:10.1007/s11042-014-2373-8
11. Lin CF, Yuan SM, Leu MC, Tsai CT (2012) A framework for scalable cloud video recorder system in surveillance environment. In: Proceedings of the 2012 9th international conference on ubiquitous intelligence computing and 9th international conference on autonomic trusted computing (UIC/ATC), pp 655–660. doi:10.1109/UIC-ATC.2012.72
12. Miao D, Zhu W, Luo C, Chen CW (2011) Resource allocation for cloud-based free viewpoint video rendering for mobile phones. In: Proceedings of the 19th ACM international conference on multimedia, MM '11. ACM, pp 1237–1240. doi:10.1145/2072298.2071983
13. Nan X, He Y, Guan L (2011) Optimal resource allocation for multimedia cloud based on queuing model. In: 2011 IEEE 13th international workshop on multimedia signal processing (MMSP), pp 1–6. doi:10.1109/MMSP.2011.6093813
14. Oh S, Hoogs A, Perera A, Cuntoor N, Chen CC, Lee JT, Mukherjee S, Aggarwal JK, Lee H, Davis L, Swears E, Wang X, Ji Q, Reddy K, Shah M, Vondrick C, Pirsiavash H, Ramanan D, Yuen J, Torralba A, Song B, Fong A, Roy-Chowdhury A, Desai M (2011) A large-scale benchmark dataset for event recognition in surveillance video. In: Proceedings of the 2011 IEEE conference on computer vision and pattern recognition, CVPR '11. IEEE Computer Society, pp 3153–3160. doi:10.1109/CVPR.2011.5995586
15. Prati A, Vezzani R, Fornaciari M, Cucchiara R (2013) Intelligent video surveillance as a service. In: Atrey PK, Kankanhalli MS, Cavallaro A (eds) Intelligent multimedia surveillance, Springer Berlin Heidelberg, pp 1–16. http://link.springer.com/chapter/10.1007/978-3-642-41512-8_1

16. Vinoski S (2006) Advanced message queuing protocol. IEEE Int Comput 10(6):87–89. doi:10.1109/MIC.2006.116
17. Wu YS, Chang YS, Juang TY, Yen JS (2012) An architecture for video surveillance service based on P2P and cloud computing. In: Proceedings of the 2012 9th international conference on ubiquitous intelligence computing and 9th international conference on autonomic trusted computing (UIC/ATC), pp 661–666. doi:10.1109/UIC-ATC.2012.43

Thanathip Limna received his B.Eng. and M.Eng. degree in Computer Engineering from Prince of Songkla University, Thailand, in 2007 and 2010, respectively. Currently, he is pursuing a Ph.D. in Computer Engineering at the same university. He is interested in the research fields of Parallel and Distributed Computing and Systems, and Cloud Technology.



Pichaya Tandayya graduated in Electrical Engineering (Communications) from Prince of Songkla University (PSU) in Thailand in 1990. She obtained her Ph.D. in Computer Science in 2001 from the University of Manchester in the area of distributed interactive simulation. Currently, she is an Assistant Professor in the Department of Computer Engineering, PSU. Her current research works concern Parallel and Distributed Computing and Assistive Technology.

VITAE

Name	Mr. Thanathip Limna
Student ID	5910130041

Education Attainment

Degree	Name of Institution	Year of Graduation
Bachelor of Engineering (Computer Engineering)	Prince of Songkla University	2007
Master of Engineering (Computer Engineering)	Prince of Songkla University	2010

Scholarship Awards during Enrolment

The Royal Golden Jubilee Ph.D. Program (Grant No. PHD/0047/2552)

List of Publication and Proceeding

- T. Limna and P. Tandayya, "Design for a flexible video surveillance as a service," in *proceedings of 2012 5th International Congress on Image and Signal Processing (CISP)*, 2012, pp. 197–201. Chongqing, Sichuan, China. doi:10.1109/CISP.2012.6469742
- T. Limna and P. Tandayya, "Video surveillance as a service cost estimation and pricing model," in *proceedings of 2015 12th International Joint Conference on Computer Science and Software Engineering (JCSSE)*, 2015, pp. 174–179. Hat Yai, Songkla, Thailand. doi:10.1109/JCSSE.2015.7219791
- T. Limna and P. Tandayya, "A flexible and scalable component-based system architecture for video surveillance as a service, running on infrastructure as a service," *Multimedia Tools and Application*, vol. 75, no. 4, pp. 1765–1791, Feb. 2016. doi:10.1007/s11042-014-2373-8
- T. Limna and P. Tandayya, "Workload Scheduling for Nokkhum Video Surveillance as a Service," *Multimedia Tools and Application*, pp. 1–27, Jan. 2017. doi:10.1007/s11042-016-4225-1