J Inf Process Syst, Vol.20, No.4, pp.442~457, August 2024
https://doi.org/10.3745/JIPS.04.0313
ISSN 2092-805X (Electronic)

JOURNAL OF INFORMATION PROCESSING SYSTEMS JIPS

On the Scaling of Drone Imagery Platform Methodology Based on Container Technology

Phitchawat Lukkanathiti and Chantana Chantrapornchai*

Abstract

The issues were studied of an open-source scaling drone imagery platform, called WebODM. It is known that processing drone images has a high demand for resources because of many preprocessing and post-processing steps involved in image loading, orthophoto, georeferencing, texturing, meshing, and other procedures. By default, WebODM allocates one node for processing. We explored methods to expand the platform's capability to handle many processing requests, which should be beneficial to platform designers. Our primary objective was to enhance WebODM's performance to support concurrent users through the use of container technology. We modified the original process to scale the task vertically and horizontally utilizing the Kubernetes cluster. The effectiveness of the scaling approaches enabled handling more concurrent users. The response time per active thread and the number of responses per second were measured. Compared to the original WebODM, our modified version sometimes had a longer response time by 1.9%. Nonetheless, the processing throughput was improved by up to 101% over the original WebODM's with some differences in the drone image processing results. Finally, we discussed the integration with the infrastructure as code to automate the scaling is discussed.

Keywords

Cloud Computing, Container Technology, Drone Imagery Processing, Infrastructure as Code, WebODM

1. Introduction

Processing drone imagery requires many computing resources due to the large number of images and processing steps involved. These images are obtained by flying a drone according to a specified route. These images often have overlapping areas and metadata that can be used in future processing. At a basic level, construction is required of an orthomosaic—a large image created from smaller images (called orthophotos). In combination with other collected data, such as altitude, and ground control points (GCPs), other image types can be constructed such as a digital elevation model (DEM), contour lines, or the normalized difference vegetation index (NVDI). Creating such images can be time-consuming, depending on factors such as the software algorithm used, the number of images, and image resolutions [1].

These images have many useful applications in many domains such as urban architecture design [2], agriculture [3,4], and geographical survey [5]. The current research focused on improving the drone imagery processing and analysis process to enable the scaling of the platform.

WebODM is an open-source platform for drone imagery processing with a large active community. It

Manuscript received January 31, 2023; first revision May 9, 2023; accepted June 6, 2023.
* **Corresponding Author:** Chantana Chantrapornchai (fengcnc@ku.ac.th)
Dept. of Computer Engineering, Faculty of Engineering, Kasetsart University, Bangkok, Thailand (phitchawat.lu@ku.th, fengcnc@ku.ac.th)

has been used as an open-source processing flow in many applications [6] and can be deployed in commercial cloud providers. Here, we have provided a methodology for experimenting with scaling the WebODM-based container-based technology. The goal was to increase the request handling capacity. The current architecture of WebODM is based on OpenDroneMap (ODM) [7]. Inside the container, WebODM constructs a processing node that runs one ODM node. A scheduling queue is used to serve a user request. Additional processing nodes can be included as other running containers or virtual machines (VMs) that run ODM.

In WebODM, when creating a task, the default executed node is "auto" which is applied to assign the task to the available node. Another method is to assign the task to the specific node in the processing node list. However, with the default installation, there is only one node called "node-odm-1" in the container. The administrative users must manually create the ODM node and add it via the user interface. This node information is stored in the database which can be retrieved for user selection.

Additional requests are queued in sequence on the given node. Adding the new node will speed up the processing. On the other hand, the node can be deleted automatically if it is disconnected or has no further use. Automation is made possible with modern container technology and the infrastructure as code (IAC) technique, automation is made possible. Terraform [8] is core to automating container creation and destruction, while Proxmox [9] manages monitoring containers and storing container images. The CPU and resource utilization can be monitored while running based on using other script.

This paper presents the task of scaling the open-source WebODM using virtualization technology, (specifically a Kubernetes cluster). By using both open-source tools, we restructured the WebODM components to meet user requirements for both flexibility and reliability. This paper presents an on-premise reference architecture for handling WebODM on a Kubernetes cluster, with the ability to scale and evaluate performance with different numbers of requests. In particular, we experimented with the performance increment when WebODM was scaled out using a VM. With concurrent processing, more throughput was achieved. We demonstrate how WebODM can be integrated with the tools, such as Terraform [8], and Proxmox [9], using their APIs. The template container can be constructed and utilized as an IAC.

The paper is structured as follows. Section 2 presents some background on the related technologies and other work related to WebODM. Section 3 explains the original design of WebODM, as well as its elements, are explained. Section 4 describes the methodology for studying the scaling out of performance. The experiments show the effectiveness of the scaling out. Section 5 presents the design for autoscaling with IAC. Section 6 provides the lessons learned and Section 7 concludes the work and discusses future research

2. Background

We present some background related to the cloud platform technologies and tools, such as containers, a Kubernetes cluster, and IAC. Existing drone imagery software is also presented. Next, we introduce WebODM and its current features. In addition, some literature regarding to WebODM is discussed.

2.1 Containerized Technology and Management

Modern cloud applications utilize container technology for virtualization. Originally, one of the virtualization forms was a VM that relied on the resources on a physical machine. Each VM contains its own operating system. One physical machine may contain many VMs to perform different purposes. However, switching between VMs has many overheads. In the next generation, virtualization occurs using the container which relies on the host operation system, making it lighter for deployment. The container has the necessary software and environment to run the application.

Kubernetes [10] is an open-source software that enables container orchestration. In addition, it can provision and manage container workloads, as well as automatic deployments. The metrics can be added to monitor the resource usage such as at the CPU, memory, and network levels. With these metrics, the scaling can be automated with predefined resources.

IAC is an approach that utilizes code to manage infrastructure. It simplifies the process of infrastructure provisioning and management. When combined with container technology, automating infrastructure provisioning becomes more convenient. Terraform [8] is one of the tools used for IAC, and it can be used in conjunction with other elements such as Kubernetes, and Docker containers. The current work used Proxmox Virtual Environment (VE) [9] with Terraform to construct VMs and Linux Container (LXC). Proxmox VE is a virtualization environment that has a web-based user interface for visualizing and updating VMs and container states, as well as providing APIs for programming.

2.2 Drone Imagery Processing Software

There are many software options available for processing drone images, both commercial and non-commercial. Some published work has included comparisons of orthomosaic and photogrammetry software [11,12]. Some popular commercial software options include DroneDeploy [13], Pix4D Mapper [14], AutoDesk Recap [15], 3DF [16], Agisoft PhotoScan [17], and ODM [18].

DroneDeploy is a popular platform with both enterprise and individual licenses available. As of 2022, plans started at $329 per month, allowing for up to 3,000 images per map, including services such as orthophoto, plant health, and GCPs.

Another popular software package is Pix4D Mapper, specifically designed for photogrammetry tasks. It creates 3D maps from 2D maps by constructing surfaces, volumes, and cloud points. The minimum monthly subscription for Pix4D Mapper is $350, with a floating license available for $4,990 [14].

The Agisoft PhotoScan photogrammetry software includes a feature for detecting powerlines. It offers three pricing models: node-lock license, floating license, and educational license [17]. The basic edition of Agisoft PhotoScan offers features such as photogrammetric triangulation, dense point cloud generation and editing, 3D models generation and texturing, and diffuse, occlusion, and normal texture map generation, as well as spherical panorama stitching. While the features are excellent, the pricing model may be unaffordable for beginners. Therefore, an open-source version is a good solution, as it can be deployed at no cost.

In this research, we focus on WebODM, which is an open-source platform with the possibility of expansion among open-source projects [19] and advanced photogrammetry features. Pell et al. [20] conducted a comparison of Structure-From-Motion results obtained from several software packages, and reported that WebODM performed similarly to other commercial software options, although its processing time may be longer for a larger number of images.

2.3 WebODM

The current release of WebODM includes features that support processing backends such as ODM and MicMac [21]. ODM is also an open-source backend that provides a command line interface with a Python program for access. It contains a collection of programs for photogrammetry toolkits, including orthomosaic photos, 3D modeling (point clouds, texture, tiles), contours, and elevation models. WebODM includes measurement UI features, such as volume, height, and area, as well as easy-to-use plant health indices like NDVI, VARI, and GNDVI [22] for analyzing and visualizing aerial imaging data [1,6]. GCPs can be integrated to improve the accuracy of image stitching while GPU support is available to improve processing time. Additionally, various outputs can be exported for further use.

WebODM uses container services, making it easy to deploy. It combines pluggable elements connected with APIs and a command line interface, allowing for modifications to add analysis features and scale computation. There have been attempts to scale out WebODM using ClusterODM [23], which can scale out on commercial clouds such as DigitalOcean and Amazon Web Services. ClusterODM is designed based on NodeODM [24], which is based on Node.js and can support NVIDIA GPU.

In [25], there was a similar attempt to investigate the scaling out of WebODM using Kubernetes (K8s). The authors implemented K8s and conducted experiments on the number of replicas ranging from 1 to 3 by scaling in and out, focusing on the response time metric. The dataset sizes tested varied from 72 to 662 images. The current study focused on the performance and throughput of concurrent WebODM, considering both horizontal and vertical scaling by assuming a different architecture, dividing cluster components into stateless and stateful objects where stateless objects can be replicated to increase concurrency. The throughput, average response time, and execution time were measured along with the discussion on how to integrate IAC to automate the scaling out.

3. Methodology and Tools

An overview of the traditional structure of WebODM is provided and its important software elements that run inside the containers on the standalone machine are presented. We elaborate on the processing flow of WebODM. We also describe the design of the Kubernetes cluster that we have developed for running WebODM. Finally, we discuss our modifications to the components of WebODM.

3.1 WebODM Structure

As the traditional WebODM architecture was not designed for deployment on a Kubernetes cluster, we needed to reorganize its software structure to enable it.

Container technology is a popular choice for modern cloud applications, as it combines necessary software dependencies and automates the construction of the environment as well as running using a Docker engine and utilizing the host operating system kernel. Multiple containers can be integrated into one application using the Docker Compose facility. Containerization offers advantages such as modularity, a light weight, reusability, and instant deployment.

The original WebODM was constructed using Docker Compose containing the elements shown in Fig. 1(a). In Fig. 1(a), Nginx and gunicorn [26] are used for web deployment and run on the default port 8000. Django is used to create REST API and models, while PostgreSQL is used to store user information, project,

and carry out task-related processing. The Celery software package [27] is used to manage a scheduling queue to process a requested task, with the Celery backend broker being Redis [28], which allows for queue monitoring with an API on port 6379. A Celery worker is created to handle the task queue. Volumes named "appmedia" and "dbdata" are mounted at "/app/media/" and "/var/lib/postgresql/data," respectively, to store the task data.

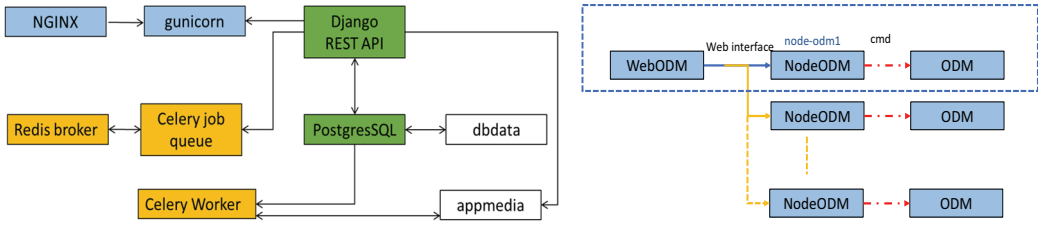


Fig. 1. (a) Architecture of WebODM (adapted from [20]) and (b) components with NodeODM.

NodeODM [24] is a container implementation for the API used by WebODM. By default, one NodeODM is created when starting the WebODM container, named node-odm-1 to provide a web interface running at the default port 3000 for uploading images and submitting task requests.

NodeODM serves as a JavaScript wrapper that executes the ODM library [17] via the command line interface to process images upon request. The default NodeODM is contained within the dashed box in Fig. 1(b). Additional processing nodes can be added via the WebODM web interface.

ODM is an open-source implementation in Python and a core tool for processing drone images. It is a collection of command-line programs that can be invoked individually. Additionally, a Python library called PyODM is available. ODM includes features such as the construction of classified point clouds, 3D textured models, georeferenced orthorectified imagery, georeferenced DEMs, and NDVI raster.

3.2 Processing Flow

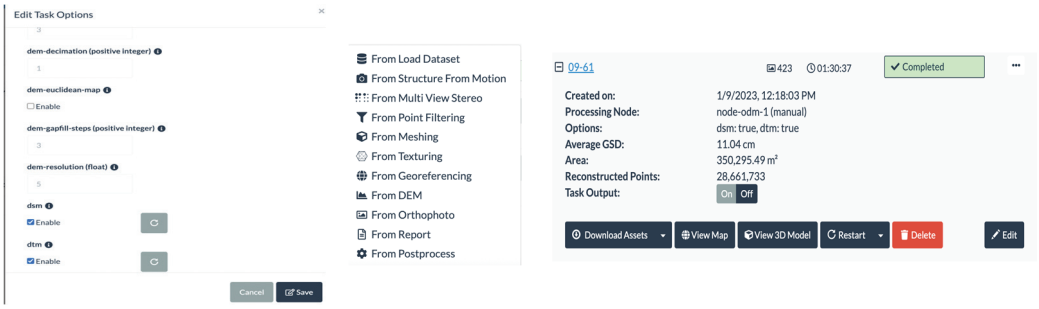
Fig. 2(a) displays the web interface of WebODM. In order to process images, the user creates a project at Step 1, and then creates a task inside the project at Step 2. Multiple tasks can be created within a project. After creating a task, the user can specify the files to process in Step 3.

After creating a task, the user can specify the worker node to process the task at 1) as shown in Fig. 2(b) which displays a list of available nodes. The default option "auto" determines the available node to process the task. New nodes can be added from the menu in Fig. 2(b). In "auto" mode, the system selects the node with the smallest queue size. The worker node takes the task from the list and processes it as a background process.

The users can edit the task setup before starting the processing by selecting the available task processing parameters, such as the number of features, DTM, DSM options, and resolution size as shown in Fig. 3(a). Users can also select the processing step to start from before clicking the start button, as shown in Fig. 3(b). These processing steps are ordered based on the WebODM document [7], and users can also choose to restart the processing at a specific step. Once the processing is complete, the results are displayed in Fig. 3(c), including the number of reconstructed points and area size.

(a) (b)

Fig. 2. (a) WebODM interface and (b) WebODM UI to create task.



(a) (b) (c)

Fig. 3. (a) WebODM parameters, (b) WebODM steps, and (c) WebODM processing results.

3.3 Kubernetes Cluster Environments

The traditional WebODM runs on a standalone system. To support the scaling, the cluster of ODM can be used as in Fig. 2. In this paper, the system was modified to run on a Kubernetes cluster, which provides the virtualization. The clusters contain the amount of VM resources where the containers are deployed. More containers can be allocated when the scaling out is performed.

The performance of scaling was investigated by setting up the environments and applying some modification of WebODM container configurations. The host machine specifications are shown in Table 1.

Table 1. The host machine specifications

CPU	Intel Xeon CPU E5-2670 v2 @ 2.50GHz (2 Sockets)
Memory	DDR-3 8.00GB 1333MHz ECC Memory (12 slots)—96 G
OS	Proxmox 7.0-13 Operating System
Kernel version	Linux 5.11.22-7-pve #1 SMP PVE 5.11.22-12
Storage	2x HGST Ultrastar 7K4000 HUS724030ALS640 3072GB (RAID 1) (Read Policy: Read Ahead)
	2x SEAGATE BARRACUDA 7200RPM SATA3 2TB (ST2000DM006) (RAID 0) (Read Policy: Adaptive Read Ahead)

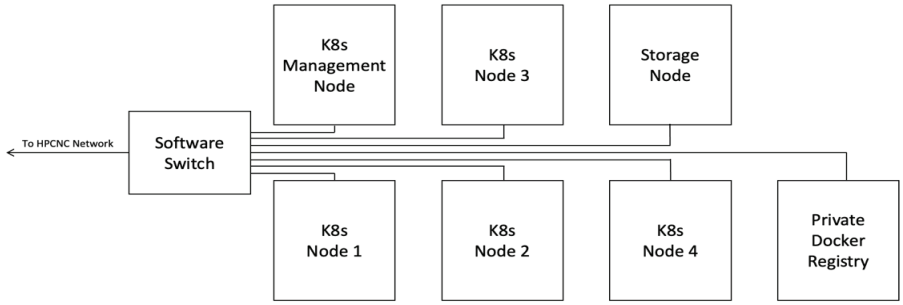


Fig. 4. Cluster for the experiments.

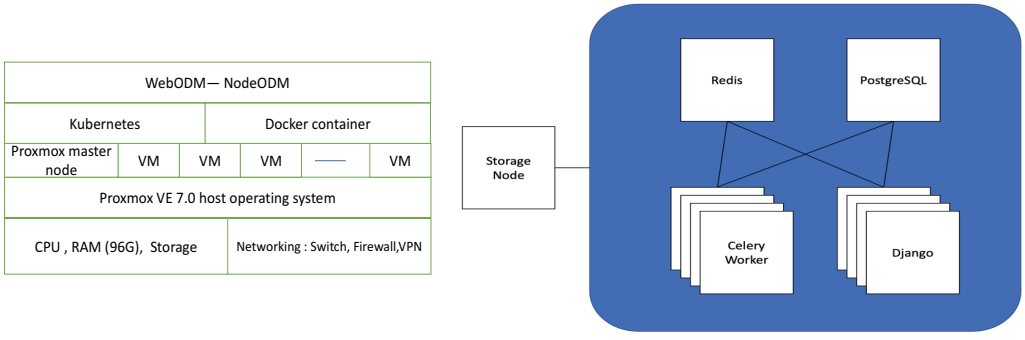
As the Kubernetes cluster is being utilized, the components are orchestrated as the infrastructure. A VM node was constructed for each component on the host, with each node connected to the lab switch as depicted in Fig. 4. The network speed was kept at 1 Gbps to mimic real-world scenarios.

Each node in Fig. 4 has the following specifications.

- Management node: 4 cores with 1 socket CPU, memory 4 GB, hard disk 100 GB.
- Worker node: 8 cores 1 with socket CPU, memory 16 GB, hard disk 100 GB
- Storage node: 4 cores 1 with socket CPU, memory 4 GB, hard disk 400 GB storing images for operating systems, and hard disk 2,000 GB, storing shared data via NFS among the VMs.
- Docker registry: 4 cores 1 with socket CPU, memory 16.00GB, and hard disk 250GB.

Fig. 5(a) shows the software stack of the testing environment is shown. Proxmox master is responsible for managing the creation, modification, and deletion of the VMs through its user interface. The Kubernetes cluster runs on the allocated VMs and has a control pane that manages all nodes. The traditional WebODM architecture shown in Fig. 1(a) launches multiple containers. However, to accommodate the Kubernetes cluster, components that were not related to data storage were replicated.

In Fig. 5(a), the modifications made to deploy WebODM on the Kubernetes cluster are shown in detail. Compared to the traditional architecture, Fig. 5(b) allows for the scaling of both Celery worker and Django components. The components are divided into two types: stateless and stateful.



(a) (b)

Fig. 5. (a) Software stack and (b) Kubernetes deployment and connections.

- Stateless components: Django, Nginx, Gunicorn, Celery worker, and Celery scheduler where every pod has the same function and there is no ordering.

- Stateful components: PostgresSQL and Redis Server. Here, the applications have different functions and cannot be replaced.

The Kubernetes StatefulSet was used to achieve horizontal scaling for Redis and PostgreSQL. Each StatefulSet creates a unique stable network identity for each replica and provides a persistent identity for the pod, allowing for reliable stateful applications. For Redis, we used a single master and multiple slave replicas to distribute the load. For PostgreSQL, we used a primary with multiple standby replicas. The number of replicas for these stateful components was determined during the initial deployment and cannot be changed dynamically.

3.4 Script Modifications

The original WebODM was launched via docker-compose which activated several components as shown in Fig. 1. The original WebODM docker-compose script was transformed into Kubernetes launch script and split into several functions based on the original one:

- database_secret.yaml – for storing the username and password of PostgreSQL
- redis.yaml – used for services and StatefulSet of Redis server.
- postgresql.yaml – used for services and StatefulSet of PostgreSQL.
- webapp.yaml – used for services and deployment of WebODM.
- worker.yaml – used for services and deployment of Celery workers.

As part of the modifications to run WebODM on a Kubernetes cluster, several miscellaneous changes were required to ensure that all pods were running in the same user session and had access to the same secret key. The cache for Redis was enabled so that all pods could use it, a worker process was added, and a mutex was implemented.

4. Performance Measurement

The results were reported based on three parameters: the number of requests per seconds, the response time per active thread, and the execution time.

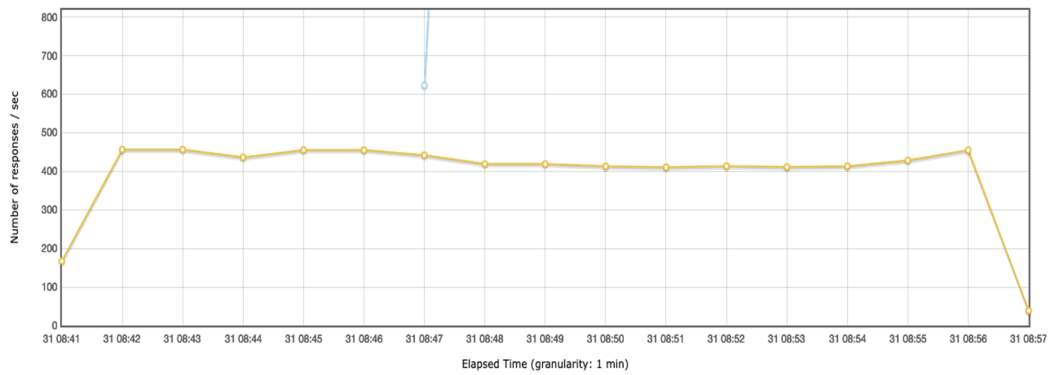


Fig. 6. Original system with 8 cores and 8G memory.

Number of requests per seconds: Apache JMeter is a tool used for injecting requests with thread group size set to 1,000 and the number of threads was incremented by 50 every 30 seconds. From the start, the ramp up occurs every 5 seconds and 5 threads are stopped for every second. Every request is sent to the WebODM master URL.

The baseline system was the VM with 8 cores and 8 G RAM running the original WebODM code. The number of responses per seconds are shown as in Fig. 6 producing around 400–500 number of responses per seconds.

The modified system was based on the Kubernetes cluster containing one replica with 8 cores and 8G RAM. The number of responses per seconds are as shown in Fig. 7(a) which is around 600–800 number of responses per seconds. This is slightly higher than that in Fig. 6 due to the increment in the number of workers in JMeter.

Consider the case when there is a reduction on the number of CPU cores. In Fig. 7(b), 1 replica with 2 CPU cores is utilized. The number of responses per second is reduced by around 1/3 compared to Fig. 7(a). When considering adding replicas in Fig. 7(c), the number of responses is almost double compared to Fig. 7(b). Fig. 7(d) shows the case where 4 replicas are selected. The four replicas have a slightly greater number of responses per seconds than for two replicas since there are two CPU cores in the setup.

Response time per active thread: Fig. 8 shows the average response time per active thread of the original WebODM. It shows that the linear increment is proportionally to the number of threads. Fig. 9(a) shows the average response time for a similar system on the Kubernetes cluster. Though there are spikes in the response time than in Fig. 8, it shows a similar trend. The spikes in the response time may have been due to the dynamic nature of the Kubernetes cluster, where pods can be created, destroyed, or moved around the cluster.

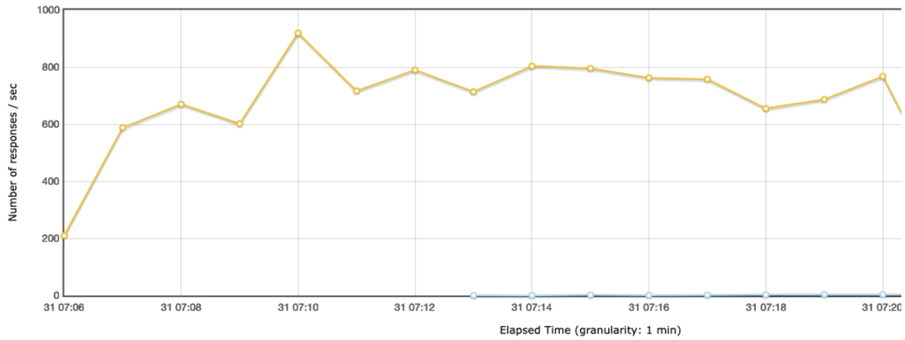
Fig. 9(b)–9(d) show the results of reducing number of CPU cores to 2 with 1, 2, and 4 replicas, respectively. All clearly shows similar linear increments when increasing the number of active threads while a greater number of replicas increases variation more.

Execution time: The execution time of processing our drone dataset using a DJI Maverick was measured. We used the term "original" refers to the original WebODM installation, while "our case" was the Kubernetes cluster testbed system for the same specification.

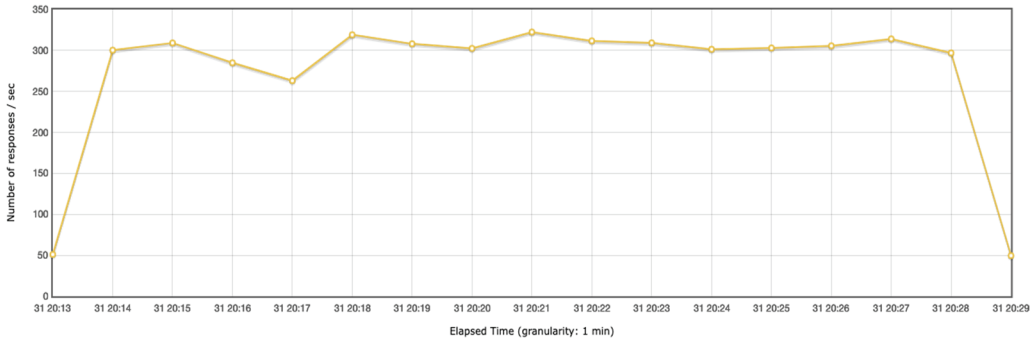
Table 2 shows the execution data for the test drone datasets with each test carried out three times. The settings for WebODM processing were auto-boundary: true, feature-quality: ultra, minimum-features: 18,000; split: 100; split-overlap: 25, and use-3dmesh: true.

The corn dataset contained 310 images and each of which has 5,472×3,648 pixels covering for an area of 22,486.88 m^2, as shown in Fig. 10, with the resulting ortho 2D images and 3D point clouds on the left and right respectively.

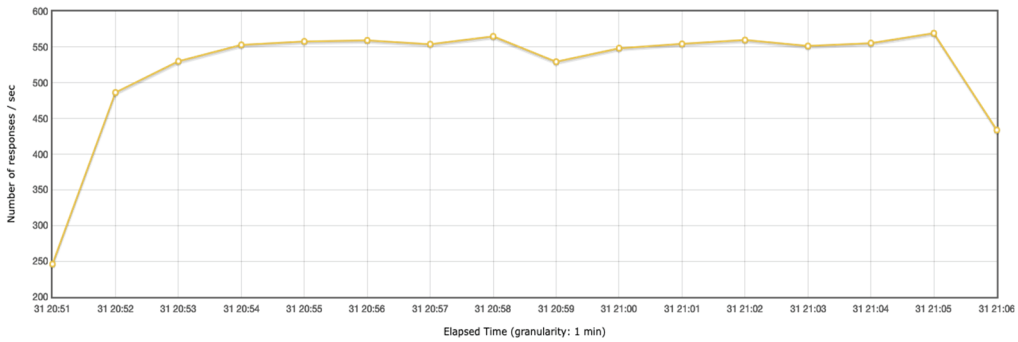
The palm dataset contained 756 images with each being 5,472×3,648 pixels and covering an area of 2,280,000 m^2. The execution time was about the same for the corn dataset for both the original and our case procedure; however, the original case was slightly longer with the palm dataset since there were more images.



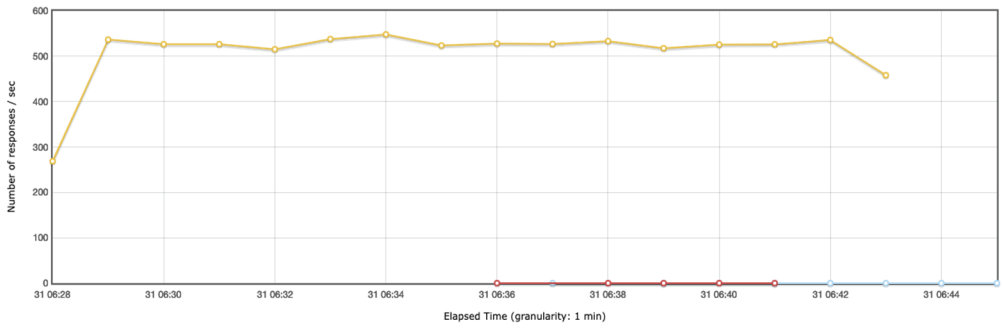
(a)



(b)



(c)



(d)

Fig. 7. (a) 1 replica CPU 8 core memory 8 GB, (b) 1 replicas CPU 2 core memory 8 GB, (c) 2 replicas CPU 2 core memory 2 GB, and (d) 4 replicas CPU 2 core memory 2 GB.

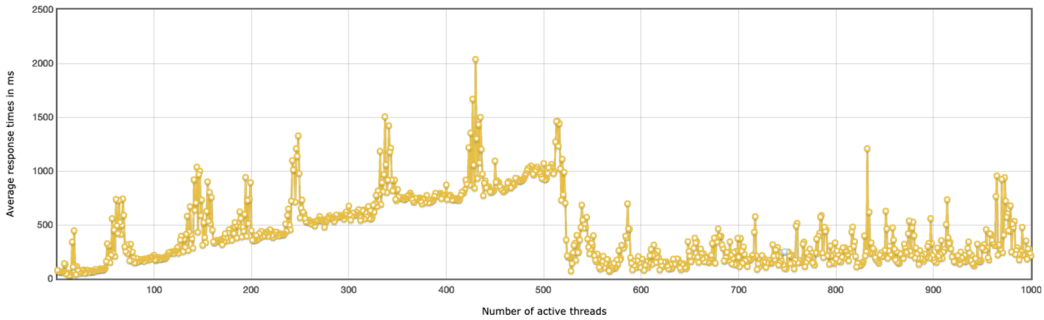
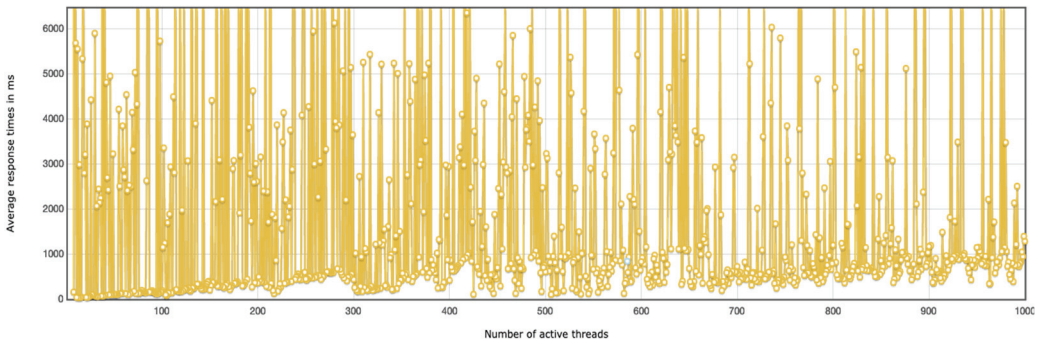
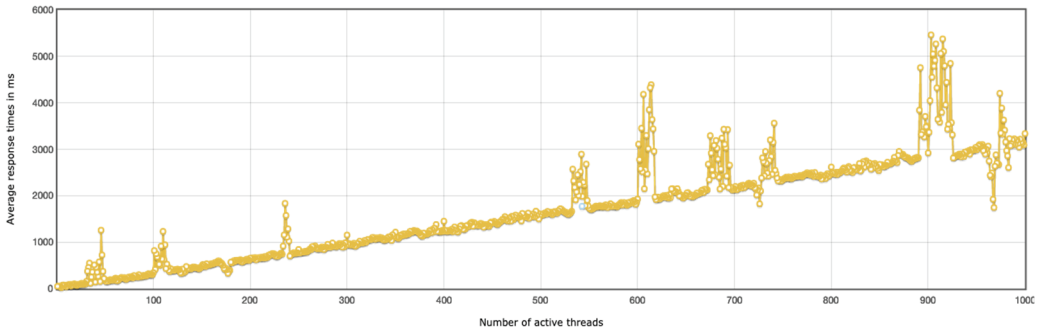


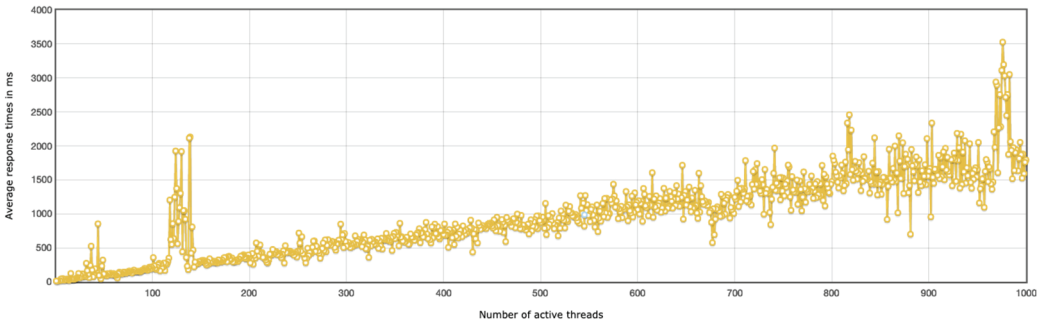
Fig. 8. Original system with 8 cores 8 GB.



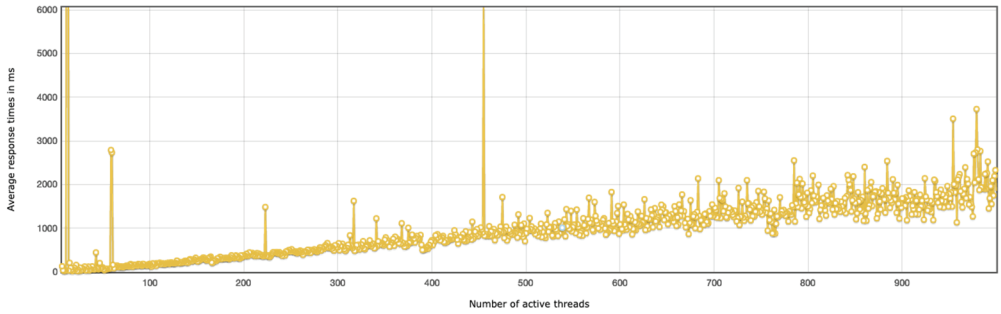
(a)



(b)



(c)

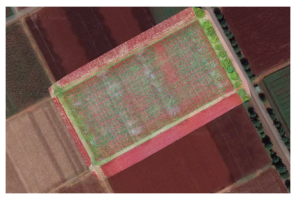


(d)

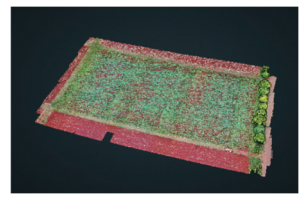
Fig. 9. (a) 1 replica CPU 8 core memory 8 GB, (b) 1 replica CPU 2 core memory 2 GB, (c) 2 replicas CPU 2 core memory 2 GB, and (d) 4 replicas with CPU 2 core memory 2 GB.

Table 2. Execution time for different datasets

Dataset	Method	Time used to process for each trial (s)			
		1st	2nd	3rd	Average
Corn dataset	Original	71:03	70:12	70:46	70:40.33
	Our case	67:50	68:17	66:12	67:26.33
Palm dataset	Original	128:18	170:08	207:17	168:34.33
	Our case	230:06	226:04	233:44	229:58.00



(a)



(b)

Fig. 10. Example from corn field dataset: (a) 2D orthoimages and (b) 3D point clouds.

5. Integration with IAC

Integration using IAC was carried out to improve the automation of horizontal scaling. Our proposed modification for this case contained three steps: 1) create the approach to add the ODM container automatically; 2) add the node information to the existing database; and 3) assign the task to the new node when there is no available free node.

Fig. 11 shows the changes from Fig. 5(a). Rather than using the Kubernetes cluster, Terraform was used for the automation of horizontal scaling. Terraform can also be utilized with the Kubernetes cluster to integrate with vertical scaling. The VM was replaced with *lxc* (Proxmox container), derived from the template created previously. There were three steps in automating the creation.

Step 1. The NodeODM template was first prepared using Proxmox UI. We created the template *lxc* from the docker file from the original github. After creating the *lxc* and installing the nodeodm docker, as well as necessary scripts (for monitoring the CPU, and memory resources), it was converted to a template.

Step 2. The Terraform script was created to perform the following: (i) connect to Proxmox node with API; (ii) use the template in step 1 to create the Proxmox *lxc* with the standard size (4 CPUs and 4 G RAM), which can be changed later when monitoring; and (iii) mount the disk volume for *lxc* and assign the running IP.

Step 3. Since we could only monitor the status of *lxc* only from inside VM, we created a script that enters *lxc* with the given IP using *ssh*. We can check the IP address of our ODM nodes from within the Proxmox node. After using *ssh* to the odm *lxc*, we checked for the CPU resources with the monitoring scripts installed in step 1. This script activated step 2 or we could make changes to the *lxc* based on the monitored outputs.

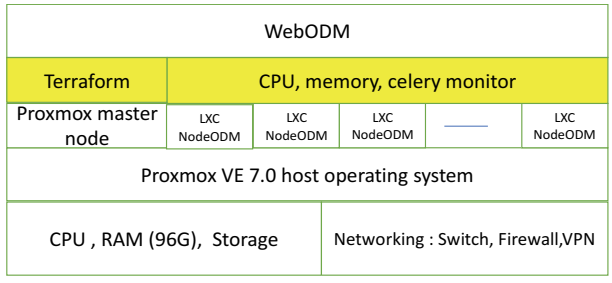


Fig. 11. Modified software stack with LAC.

6. Discussion

From the first experiment, the system (having 1 replica, CPU 8 core, and memory 8 GB) could handle 2.01 more times requests per second than the original version. The system with 4 replicas (each having 2 cores with memory 2 GB) could handle 1.255 times more requests per second than the original version.

In the second experiment, when the number of active threads was increased, the response time to return the web page varied as shown in Table 3 and Figs. 7–8. The increment from the original was about 1.09 times on average, except for the case using 1 replica and 2 CPU cores where the increment was about 1.6 times which may have been due to saving more time on overheads. In the last experiment in Table 2, the test dataset could be processed faster than using the original version by 4.58 times.

Table 3. Average response time per active threads

System type	Average response time per active threads (ms)	Ratio increment
Original	1.9135	1
1 replica CPU 8 core memory 8 GB	-	-
1 replica CPU 2 core memory 2 GB	3.1301	1.636
2 replica CPU 2 core memory 2 GB	2.0024	1.046
4 replica CPU 2 core memory 2 GB	1.9493	1.019

Table 4 compares our work and the other studies, where "original" refers to the original WebODM. The original work provides the cluster version, called ClusterODM where it allows NodeODM to be added. NodeODM is assumed to run on another node. The work in [25] did not consider vertical scaling while our work performed the test on vertical scaling as well (for example, considering the case for 1, 2,

or 4 CPUs). Rather than using execution time, two different metrics were evaluated: throughput (the number of requests per seconds), and average response time for requests.

The resource size must be estimated to take an advantage of IAC in the case of using *lxc*. This estimation may require more experiments. According to the suggestion on the WebODM installation page, the minimum hardware requirement for processing is 4 GB RAM and 20 GB. Notably, that this is the recommended size for 40 images [29]—if more images are needed, the memory and CPU resources should be increased accordingly.

7. Conclusions and Future Work

This paper presented a study of the scaling approach for WebODM on the Kubernetes cluster. WebODM is an open-source software for processing drone imagery. Its design architecture contains pluggable elements, including ODM, NodeODM, and Celery, and it is deployed as a web application. We designed the Kubernetes cluster to run the WebODM where the elements of WebODM were designed based on the cluster. Our design aids horizontal scaling with NodeODM interfacing ODM. The experiments demonstrated the better performance of the modification based on average response time, number of responses per second, and total execution time. Finally, we suggest the use of IAC to automate the scaling with given metrics, the implementation of this being in progress. The modified scripts of this work for testing the scaling and the test datasets are available at https://github.com/cchantra/webodm-kube.

Conflict of Interest

The authors declare that they have no competing interests.

Funding

This work was supported in parted by the Kasetsart University Research and Development Institute, (KURDI), Bangkok, Thailand (Grant no. KURDI FF(KU)33.65).

References

[1] GitHub, “WebODM: GNU Affero General Public License v3.0,” 2022 [Online]. Available: https://github.com/OpenDroneMap/WebODM.

[2] S. Hendriatiningsih, A. Y. Saptari, A. Soedomo, R. Widyastuti, P. Rahmadani, and A. Harpiandi, “Large scale mapping using unmanned aerial vehicle (UAV)-photogrammetry to accelerate complete systematic land registration (PTSL)(Case Study: Ciwidey Village, Bandung Regency, Indonesia),” *IOP Conference Series: Earth and Environmental Science*, vol. 313, no. 1, article no. 012042, 2019. https://doi.org/10.1088/1755-1315/313/1/012042

[3] K. Kawamura, H. Asai, T. Yasuda, P. Khanthavong, P. Soisouvanh, and S. Phongchanmixay, "Field phenotyping of plant height in an upland rice field in Laos using low-cost small unmanned aerial vehicles (UAVs)," *Plant Production Science*, vol. 23, no. 4, pp. 452-465, 2020. https://doi.org/10.1080/1343943X.2020.1766362

[4] L. Volpato, F. Pinto, L. Gonzalez-Perez, I. G. Thompson, A. Borem, M. Reynolds, B. Gerard, G. Molero, and F. A. Rodrigues Jr., "High throughput field phenotyping for plant height using UAV-based RGB imagery in wheat breeding lines: feasibility and validation," *Frontiers in Plant Science*, vol. 12, article no. 591587, 2021. https://doi.org/10.3389/fpls.2021.591587

[5] S. H. Chio and C. C. Chiang, "Feasibility study using UAV aerial photogrammetry for a boundary verification survey of a digitalized cadastral area in an urban city of Taiwan," *Remote Sensing*, vol. 12, no. 10, article no. 1682, 2020. https://doi.org/10.3390/rs12101682

[6] O. H. Y. Lam, M. Dogotari, M. Prum, H. N. Vithlani, C. Roers, B. Melville, F. Zimmer, and R. Becker, "An open source workflow for weed mapping in native grassland using unmanned aerial vehicle: using Rumex obtusifolius as a case study," *European Journal of Remote Sensing*, vol. 54(sup1), pp. 71-88, 2021. https://doi.org/10.1080/22797254.2020.1793687

[7] P. Toffanin, *OpenDroneMap: The Missing Guide*, 2nd ed. St. Petersburg, FL: UAV4GEO, 2023.

[8] Terraform, "Automate Infrastructure on any cloud with Terraform," c2023 [Online]. Available: https://www.terraform.io/.

[9] Proxmox Virtual Environment [Online]. Available: https://www.proxmox.com/en/proxmox-virtual-environment/overview.

[10] Kubernetes [Online]. Available: https://kubernetes.io/.

[11] J. Gross, "A comparison of orthomosaic software for use with ultra high resolution imagery of a wetland environment," 2015 [Online]. Available: https://www.semanticscholar.org/paper/A-Comparison-of-Orthomosaic-Software-for-Use-with-a-Gross/d330cc157e6a9c2d23bdba8b87695c12dc0430cf.

[12] F. Corrigan, "12 Best photogrammetry software for 3d mapping using drones," 2020 [Online]. Available: https://www.dronezon.com/learn-about-drones-quadcopters/drone-3d-mapping-photogrammetry-software-for-survey-gis-models/.

[13] DroneDeploy: Drone Mapping Software [Online]. Available: https://www.dronedeploy.com.

[14] PIX4Dmapper [Online]. Available: https://www.pix4d.com/pricing/pix4dmapper.

[15] AutoDesk ReCap [Online]. Available: https://asean.autodesk.com/solutions/photogrammetry-software.

[16] 3DFlow, "3DF ZEPHYR," 2022 [Online]. Available: https://www.3dflow.net/3df-zephyr-photogrammetry-software/.

[17] Agisoft, "Discover intelligent photogrammetry with metashape," 2023 [Online]. Available: https://www.agisoft.com/.

[18] Github, "OpenDroneMap" 2022 [Online]. Available: https://github.com/OpenDroneMap/.

[19] J. Baker, "8 open-source drone projects," 2018 [Online]. Available: https://opensource.com/article/18/2/drone-projects.

[20] T. Pell, J. Y. Li, and K. E. Joyce, "Demystifying the differences between structure-from-MotionSoftware packages for pre-processing drone data," *Drones*, vol. 6, no. 1, article no. 24, 2022. https://doi.org/10.3390/drones6010024

[21] GitHub, "NodeMICMAC," 2022 [Online]. Available: https://github.com/OpenDroneMap/NodeMICMAC.

[22] Phthon Software Foundation, "vegindex 0.10.2," 2022 [Online]. Available: https://pypi.org/project/vegindex/.

[23] GitHub, "ClusterODM," 2022 [Online]. Available: https://github.com/OpenDroneMap/ClusterODM.

[24] Github, "NodeODM," 2022 [Online]. Available: https://github.com/OpenDroneMap/NodeODM.

[25] H. N. Vithlani, M. Dogotari, O. H. Y. Lam, M. Prum, B. Melville, F. Zimmer, and R. Becker, "Scale drone mapping on K8S: auto-scale drone imagery processing on Kubernetes-orchestrated on-premise cloud-computing platform," in *Proceedings of the 6th International Conference on Geographical Information*

Systems Theory, Applications and Management (GISTAM), Prague, Czech Republic, 2020, pp. 318-325. https://doi.org/10.5220/0009816003180325

[26] gunicorn 22.0.0 [Online]. Available: https://gunicorn.org/.

[27] Celery 5.4.0 documentation: workers guide [Online]. Available: https://docs.celeryq.dev/en/stable/userguide/workers.html.

[28] Redis [Online]. Available: https://redis.io/.

[29] OpenDroneMap: installation and getting started [Online]. Available: https://docs.opendronemap.org/installation/.

Phitchawat Lukkanathiti **https://orcid.org/0000-0002-5273-3244**

He graduated from Kasetsart University of Thailand with a Bachelor of Engineering (Computer Engineering) with Second Class Honors in 2022. His fields of interest include software engineering, security, high-performance computing, and distributed systems.

Chantana Chantrapornchai **https://orcid.org/0000-0002-8699-5736**

She obtained here Ph.D. from Department of Computer Science and Engineering, University of Notre Dame, in 1999. Currently, she is a professor of Department of Computer Engineering, Faculty of Engineering, Kasetsart University, Thailand. Her current research interests include GPUs processing, parallel and distributed computing, deep learning model optimization, and cloud computing.