Scale Drone Mapping on K8S: Auto-scale Drone Imagery Processing on Kubernetes-orchestrated On-premise Cloud-computing Platform

Hemang Narendra Vithlani[1][a], Marcel Dogotari[1], Olee Hoi Ying Lam[1], Moritz Prüm[1], Bethany Melville[2], Frank Zimmer[1] and Rolf Becker[1]

[1] *Faculty of Communication and Environment, Rhine-Waal University of Applied Sciences, 47475 Kamp-Lintfort, Germany*

[2] *Astron Environmental Services, 129 Royal Street, East Perth Western Australia, 6004, Australia*

Keywords: Drone Imagery, Opendronemap, Photogrammetry, Open-source, Kubernetes, Cloud Computing.

Abstract: Aerial images acquired using drone-based imaging sensors can be processed by photogrammetry toolkits to create geometrically corrected 2D orthophoto and/or 3D models. This is a crucial step for many of the ever-evolving civil applications of drones such as precision agriculture and surveying. Nevertheless, limited computational resources become bottleneck in providing these results quickly. Cloud computing helps in such scenarios because of its value-added features, namely virtualization, elasticity, high performance and distributed computing for the web-based image processing. The containerization approach plays a vital role in cloud computing by providing operational efficiency. Container orchestration engine, Kubernetes, not only provides template-based or GUI-based service deployment but also better monitoring, log querying and auto-scaling. The present work displays a scalable photogrammetry service, deployed on a Kubernetes-orchestrated on-premise cluster. This reference implementation on Kubernetes enables the parallel processing of large datasets in less time than a single computer using the free and open-source toolkit OpenDroneMap.

1 INTRODUCTION

Throughout the last decade, an acceleration in the demand for computing resources and the availability of low-cost, interconnected computers have led to a valuable increase in the usefulness and functionality of distributed systems (Tanenbaum and van Steen, 2016). According to Bejjam and Seshashayee (2018), the data in distributed systems can be easily analyzed (than on a single machine) and interpreted in different ways and can also, be obtained from different sources. Many research groups are using drone technologies with imaging sensors to acquire aerial data where less time and less (human) efforts are desired. Open-source toolkits help researchers collaborating with small and medium-sized enterprises (SMEs) cost-effectively solve problems in various environmental fields (Vithlani et al., 2019).

Cloud computing is a popular concept for handling web-based data processing, that hands over Information Technology (IT) infrastructure and service management to the service providers. The client can utilize the modalities of distributed computing vastly at the desired time. Although, there is no need to set up and administer the physical infrastructure from scratch, the users conventionally lean towards a middle path where the administration of the servers in distributed computing remains under their control (Garrison and Nova, 2017).

This work aims to achieve auto-scaling for the resource-hungry task of generating orthophoto using open-source tools (namely OpenDroneMap) in a cloud-native environment (namely Kubernetes and Rancher) was conducted. Kubernetes' readying automation, scaling, and management of containerized application can serve operational-end (Sarajlic et al., 2018); while the open-source, web processing and rebranding nature of OpenDroneMap (ODM) can serve very well to the end-users. Contributions of this paper include on-premise reference cloud computing architecture, added-value to the ODM project by Kubernetes deployment, auto-scaling workflow, and evaluation using a parallel processing algorithm of ODM.

Section 2 captures the related works with its supporting and opposite arguments for this work. Section 3 conceptualizes virtualization approaches, container orchestration technique, and ODM. Section 4 presents an experimental system and dataset prepara-

[a] https://orcid.org/0000-0001-7721-3467

Vithlani, H., Dogotari, M., Lam, O., Prüm, M., Melville, B., Zimmer, F. and Becker, R.
Scale Drone Mapping on K8S: Auto-scale Drone Imagery Processing on Kubernetes-orchestrated On-premise Cloud-computing Platform.
DOI: 10.5220/0009816003180325
In *Proceedings of the 6th International Conference on Geographical Information Systems Theory, Applications and Management (GISTAM 2020)*, pages 318-325
ISBN: 978-989-758-425-1

tion, workflow, and the dataset and parameters used for the system evaluation. Discussion and visualization of the results are presented in section 5. Closing thoughts and future steps are discussed in section 6.

2 RELATED WORK

Cloud computing is becoming a more convenient choice for geo-scientists because of its effective and efficient resource management capabilities, and also because web-based solutions allow multiple users at the same time. Lan et al. (2018) investigated the potential of processing massive remote-sensing imagery for land-cover change detection using cloud computing. They concluded that the computation time is directly proportional to the size of the dataset. Kang and Lee (2016) studied the capability of auto-scaling based on OpenStack, i.e. to scale the cloud-environment based on the number of users. However, because OpenStack supports a huge software stack, Suryateja (2020) suggested that its architecture was not suitable for virtualized set ups. According to Martinez-rubi et al. (2017), parallel processing was proven to be necessary to reduce computation time for a photogrammetry process, and that virtualization was desirable in the situation where resource constraints applied. To ensure the proposed set up can benefit SMEs, open-source solutions are preferred. Groos et al. (2019) had demonstrated a low-cost system set up and a promising use-case of high-resolution imagery processing based on ODM. These works demonstrated the potential of using open-source cloud computing platforms for photogrammetry workflows. Yet, to the best of our knowledge, no research had been done in evaluating the performance of parallel and on-premise cloud computing using Kubernetes containerization technology in the case of remote sensing image processing. Therefore, in this work, not only an open-source photogrammetry processing chain had been set up based on the previous work in an on-premise cluster (Vithlani et al., 2019), but system evaluation after auto-scaling had also been performed using various datasets.

3 TOOLS AND TECHNOLOGIES

Cloud computing development has been trending and lifted since the software advancement: Virtualization. Conceptually, the virtualization technique creates division into physical resources and separates them from original resources and other virtual environments. Virtualization can be implemented through various approaches. One approach is the hypervisor-based approach, where a hypervisor is running on the top of the host operating system (OS) and distributing the physical hardware resources to every guest OS. Each physical machine is divided into different virtual machines (VMs) and can be used for different purposes. This approach is advantageous due to the above-mentioned reason. Yet the problem with this hypervisor is that each VM has its own OS. Container-based virtualization approach came into the market to overcome the problems with the performance overhead of hosted hypervisors while keeping the benefit of simpler deployment of the same. This technology uses the fundamental concept of using the kernel features to create an isolated virtual environment for processes, makes it light-weight and platform-independent (Pathak et al., 2019; Tae-Hoon Kim and Rajput, 2016).

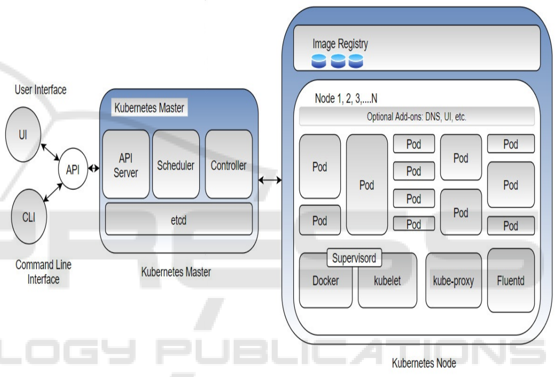


Figure 1: Kubernetes component diagram based on the work by (MSV, 2016).

Kubernetes is an open-source project that is used for managing large containerized workloads, as well as scaling, automatic deployment, and provision as a service in various sort of medium like virtual, physical and cloud infrastructure. Another important advantage of Kubernetes is that it controls and monitors hardware usage like CPU, storage and RAM. From the findings of Shah and Dubaria (2019), features like popularity, community support, auto-scaling, logging, monitoring, and better GUI are among the advantages of Kubernetes over the alternative orchestration platform Docker Swarm. After diligent consideration, Kubernetes was chosen for the task. As illustrated in Figure 1, Kubernetes follows a master-slave architecture. The Kubernetes master is responsible for scheduling the deployments, allowing or disallowing the application programming interfaces (APIs) and monitoring the entire cluster. The API server is used to control or communicate with the cluster components using REST APIs. Kubernetes slaves

- the worker nodes - handle the work-force and expose computation, network or storage. In this work, Docker[1] (v18.09.3) has been selected as a container run-time environment on each node, because of its strong user-base. A pod, the smallest unit on a node, consists of one or more containers inside it. It supports shared storage and network for the containers. For scaling up or down an application, only the addition or deletion of the pods is required. Pod, service, deployment and namespace definitions are stored in the master node in etcd storage (an open-source key-value store). The current status of Kubernetes objects is exposed to the master node. The Kube-scheduler schedules new workloads based on resource availability. Kubectl is a command-line tool that helps to communicate with the API server.

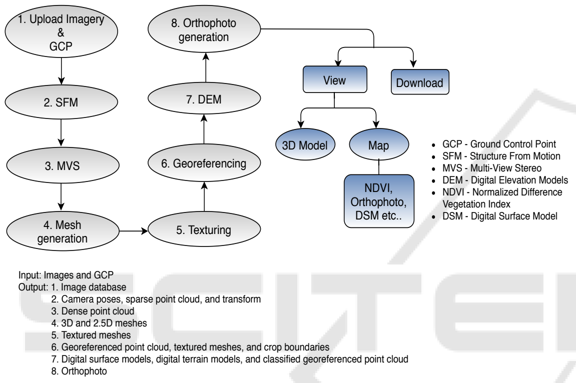

1. Upload Imagery & GCP
2. SFM
3. MVS
4. Mesh generation
5. Texturing
6. Georeferencing
7. DEM
8. Orthophoto generation
View
Download
3D Model
Map
NDVI, Orthophoto, DSM etc..
GCP - Ground Control Point
SFM - Structure From Motion
MVS - Multi-View Stereo
DEM - Digital Elevation Models
NDVI - Normalized Difference Vegetation Index
DSM - Digital Surface Model
Input: Images and GCP
Output: 1. Image database
2. Camera poses, sparse point cloud, and transform
3. Dense point cloud
4. 3D and 2.5D meshes
5. Textured meshes
6. Georeferenced point cloud, textured meshes, and crop boundaries
7. Digital surface models, digital terrain models, and classified georeferenced point cloud
8. Orthophoto

Figure 2: ODM processing chain based on the work by (Toffanin, 2019).

OpenDroneMap (ODM) is an open-source photogrammetry toolkit used for mosaicking, modeling, analyzing and visualizing aerial imaging data (OpenDroneMap, 2020). It is used in this work to support the drone-related business activities of SMEs, while providing them the freedom to modify and present the project to different stakeholders and eliminating the cost for the black-boxed vendor-owned software. ODM can be utilized from the command line as well as with a user-friendly web interface. According to Toffanin (2019), ODM is used across different fields for a multitude of applications, such as monitoring agricultural crop yields, mapping land zones, classifying and tallying trees, improving OpenStreetMap and stitching aerial images. ODM is developed with a microservice-based architecture in mind and supports containerization. It can be seen from Figure 2 that the collection of aerial imagery with Ground Control Points (GCPs) (discretionary) is stacked in the first processing steps from the user input. Structure from motion (SFM) is a photogrammetry approach for generating the 3D structure from overlapping 2D images with the help of camera motion. Utilizing point of view geometry and optics, the position and angle of the camera can be recouped for each image. Meshing is the method of joining the results from previous steps to operate the dataset as a whole. Texturing will assist with adding colors to meshes and create texture meshes. Georeferencing is the method of translating the local coordinate space to global coordinate space. Once georeferenced, boundaries get defined for the orthophotos and digital elevation models (DEMs) (Toffanin, 2019).

4 EXPERIMENTAL WORKFLOW

The novelty of this experiment consists of on-premise cloud computing, ODM deployment in Kubernetes to achieve auto-scaling and its performance evaluation for faster drone image post-processing.

4.1 System Configuration

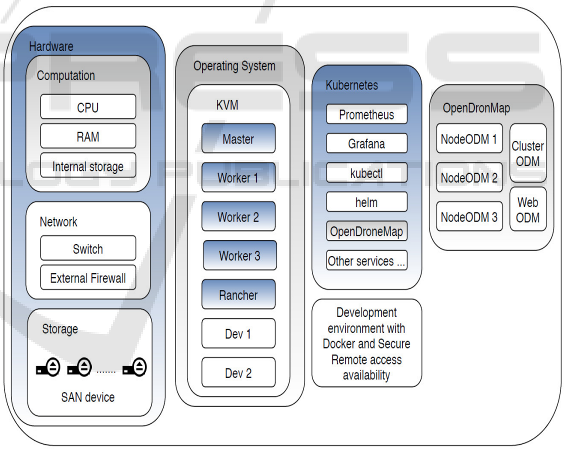


Figure 3: System design overview with software and hardware components.

The system design overview with software and hardware components is shown in Figure 3. For the number cruncher application of remote-sensing imagery processing, a bare-metal set up was prepared. It consisted of a computer cluster with high-performance nodes, a 10 Gbps Cisco manageable switch, and a Storage Area Network (SAN) device, which were set up to serve as computation engines, backbone network, and storage, respectively. To add heterogeneity in computation, processing nodes were chosen with three different configurations (of CPUs, GPUs, RAM, and SSDs) as shown in Table 1. All the servers

[1]https://github.com/docker

were provisioned with Ubuntu server OS (18.04.2 LTS GNU/Linux 4.15.0-58-generic x86-64) for easy Kubernetes deployment.

Table 1: Bare-Metal servers configurations.

	No. of CPU cores	No. of GPUs	RAM (GB)	Internal SSD (TB)
1	10	4	128	2
2	16	-	512	2
3	28	-	512	2

In this paper, auto-scaling were applied on all three machines, a Kubernetes [2] (v1.16.3) master, development engine, and Rancher GUI [3] GUI (v2.3.3) engine. To realize this set up from the available configuration a layer of virtualization was setup using an open-source KVM[4] (Qemu v2.11.1) because of its performance benefits and lower provisioning time. Rancher empowers production-grade Kubernetes with backup, security, one-click installations, and a single command to add new workers or master nodes in the cluster. Previous setup Vithlani et al. (2019) was expanded to handle parallel workload using Kubernetes in the presented work.

4.2 Process Flow

Parallel processing in ODM had been made available since version 0.6.0. However, auto-scaling was only supported since ClusterODM (v1.3.7) when commercial cloud platforms such as DigitalOcean, Hetzner, Scaleway or Amazon Web Services were made available. A new algorithm was introduced to perform split-merge (Toffanin, 2019) for distributed computing, after realizing the Horizontal Pod Autoscaler (HPA) in Kubernetes ODM deployment to cost-effectively manage resources. To access multiple processing instances (NodeODM v1.6.0) for distributed processing over a single network address (ClusterODM), some obligatory steps were needed to be performed by the administrator in advance as portrayed in Figure 4.

Figure 5 shows how the system was made accessible to the web-portal WebODM (v1.2.0) for better usability. This user-centric approach is meant to provide an abstraction layer over the underlying infrastructure, scaling calculations and resource management. The parameters used in the proposed implementation for parallel processing included (Toffanin, 2019):

[2]https://github.com/kubernetes/kubernetes
[3]https://github.com/rancher/rancher
[4]https://github.com/qemu/qemu

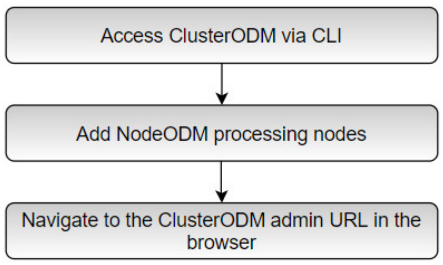

Access ClusterODM via CLI
Add NodeODM processing nodes
Navigate to the ClusterODM admin URL in the browser

Figure 4: Required steps for realizing auto-scaling and parallel processing with the help of ClusterODM.

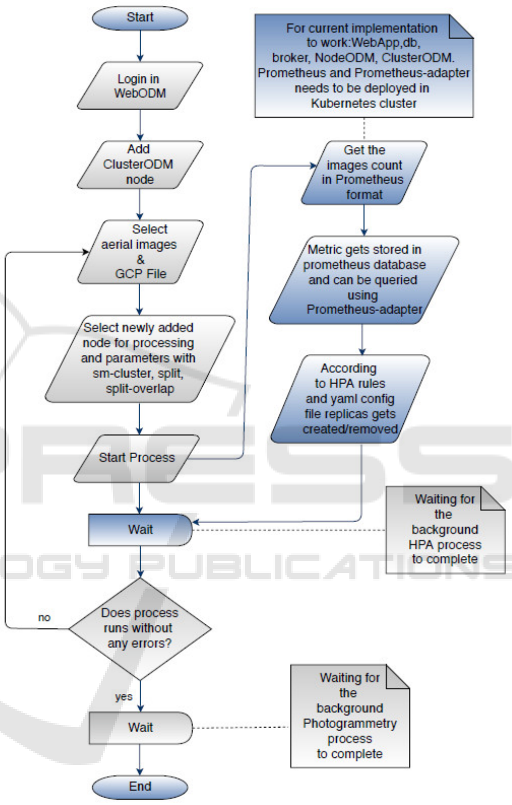

Start
Login in WebODM
Add ClusterODM node
Select aerial images & GCP File
Select newly added node for processing and parameters with sm-cluster, split, split-overlap
Start Process
Wait
Does process runs without any errors?
no
yes
Wait
End
For current implementation to work:WebApp,db, broker, NodeODM, ClusterODM. Prometheus and Prometheus-adapter needs to be deployed in Kubernetes cluster
Get the images count in Prometheus format
Metric gets stored in prometheus database and can be queried using Prometheus-adapter
According to HPA rules and yaml config file replicas gets created/removed
Waiting for the background HPA process to complete
Waiting for the background Photogrammetry process to complete

Figure 5: Experimental workflow. Typical steps to be followed by a user to realize auto-scaling and parallel processing in WebODM.

- sm-cluster: it states a URL to a ClusterODM instance. At the point when joined with the split choice, it empowers the conveyed split-merge pipeline for handling huge datasets in parallel utilizing various processing nodes.
- split: for proper sub-division of the dataset. As per Equation (1), this choice determines the number of images that ought to be processed in each submodel.

$$number\ of\ submodels = \frac{total\ images}{split\ value} \tag{1}$$

- split-overlap: for the adequate merging of the submodels. In arrange to adjust and blend comes about, each submodel must be reproduced with a certain sum of overlap (in meters) and redundancy with other submodels.

Afterwards, the algorithm waited for the auto-scaling status response, in order to proceed with the photogrammetry step in parallel. The registered node status (online/offline) from ClusterODM, served as a validation point for the system's recursive wait period. This auto-scaling part was handled by Kubernetes after getting the required information from the WebODM.

4.3 Auto-scaling in Kubernetes

In the presented work, the HPA was chosen due to the dynamic resource utilization and limited resource availability in SMEs. For HPA in Kubernetes to calculate desired replicas custom metrics current value needed to be exported. A newly introduced monitor in the ODM, exports input files count (custom metric) in Prometheus[5] format:

```
# HELP Cluster_images_in The number of Images
from Cluster input
# TYPE Cluster_images_in gauge
Cluster_images_in{method="IMAGES UPLOADED",
status="OK"} 524
```

These outcomes were then exported to a predicator or Prometheus where it can be stored in and queried from a time-series database setup in Kubernetes. Inside Kubernetes, auto-scaling was performed based on the specifications of the HPA config file:

```
kind: HorizontalPodAutoscaler
...
spec:
  maxReplicas: 3
  metrics:
  - type: Pods
    pods:
      metric:
        name: Cluster_images_in
      target:
        type: AverageValue
        averageValue: "200"
  minReplicas: 1
....
```

This example HPA config file is for custom metrics type "Pod". There are two different types of custom metrics namely pod and object. Unlike pod metrics that determine the replica count based on target value averaged across pods, object metrics describe a different object in the same namespace. Desired replica count can be derived as per Equation (2)[6]:

$$ceil\left[currentReplicas \times \left(\frac{currentMetric}{desiredMetric}\right)\right] \quad (2)$$

4.4 Evaluation Parameters

In order to evaluate the performance of the designed approach, so that the results could be a reference for SMEs working in the drone and environmental research, datasets and system configuration that fulfill the 3Vs (volume, variety, and velocity) were selected in this work. Higher and lower number of images were chosen for the experiment (volume); different datasets consist of RGB, Multispectral openly available datasets and use-case specific self-acquired drone imagery (variety); the speed of the execution is ensured by the comparison of low and high config nodes to distributed execution (velocity).

Processing enormous dataset is usually challenging with limited resources. This is because, with increment in images count, RAM usage also increases. In the split-merge algorithm, the dataset gets divided into sub-parts called submodels. These submodels can be processed parallelly on the local or remote machine to overcome memory shortcomings while achieving the desired output from the mapping process (Toffanin, 2019).

Table 2: Dataset Overview.

Dataset	Resolution (DPI)	No. of Images	Size (MB)	GCP
Grassland 2019	72	186	2030	Yes
Waterbury	350	248	1921	No
Zoo	350	524	3521	No
Grassland 2018	72	662	5700	No
MicaSense	144	210	517.7	No
Garfield	96	112	276.2	No

Six geo-tagged datasets had been selected for system evaluation. These included four open-source datasets that were known as Waterbury (248 RGB images), Zoo (524 RGB images), MicaSense[7] (210 multispectral images) and Garfield (112 multispectral images). As well as two grassland datasets that were acquired using the consumer-grade DJI Phantom 4 Pro, known as Grassland 2019 (186 RGB images) and Grassland 2018 (112 RGB images) (Lam et al., 2019). Table 2 shows the qualitative characteristics of each dataset.

[5]https://github.com/prometheus/prometheus

[6]https://kubernetes.io/docs

[7]https://community.opendronemap.org/c/datasets/10

Table 3: Overview of the test machine configurations.

Machine Configuration	Logical CPUs	RAM (GB)	Split-merge	Auto-scaling
Low	10	$\sim$ 52	No	Yes
High	28	$\sim$ 250	No	Yes
3-worker nodes	10, 28, 18	$\sim$ 250	Yes	Yes

Fine-tuning between resource necessity and processing velocity can be analyzed by executing the algorithm with three different configurations, which are laid out in Table 3.

```
horizontal-pod-autoscaler-downscale-delay: 6s
horizontal-pod-autoscaler-
downscale-stabilization: 15s
horizontal-pod-autoscaler-sync-period: 3s
horizontal-pod-autoscaler-upscale-delay: 5s
prometheus.scrapeInterval: 15s
prometheus.evaluationInterval: 15s
```

For comparative evaluation of HPA implementations, “Pod” and “Resource” metrics were evaluated with Kubernetes metrics default scrape intervals as well as modified intervals. Modified timeouts are listed above while the response times are mentioned in Table 4.

5 RESULTS AND DISCUSSION

The performance of the proposed auto-scaled split-and-merge system set up was evaluated using the six datasets as described in Section 4.4. Exported values (input files count) from ODM as explained in Section 4.3, got successfully parsed and queried via Prometheus end-point. Time-series based representation of value change was scraped and visible in Figure 6. This time-series values can be further exploited (in Kubernetes HPA) to perform auto-scaling. The outcome of this internal HPA process can be visualized with respect to current replica count (plus, resource utilization) in Grafana[8] (v3.6.3) end-point as shown in Figure 7.

The execution time of the photogrammetry process relies on many external and internal parameters, such as the data acquisition platform (drone, camera model, sensors) and environmental conditions. However, results portrayed here only emphasis on the effect of computation power to the dataset processing times as demonstrated in Figure 8. From the illustration, first four bar chart results represent the execution time (in seconds) of the datasets Grassland 2018 and

[8]https://github.com/grafana/grafana

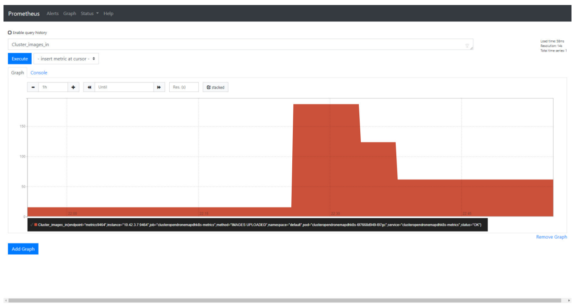


Figure 6: View of Prometheus with time-series graph generated from different executions with the different number of input images.

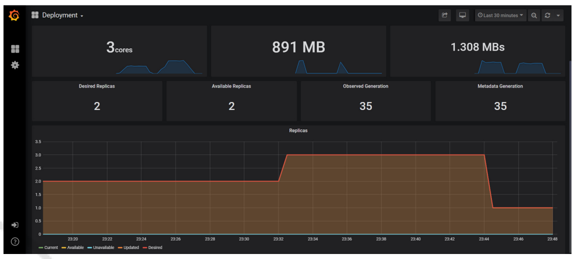
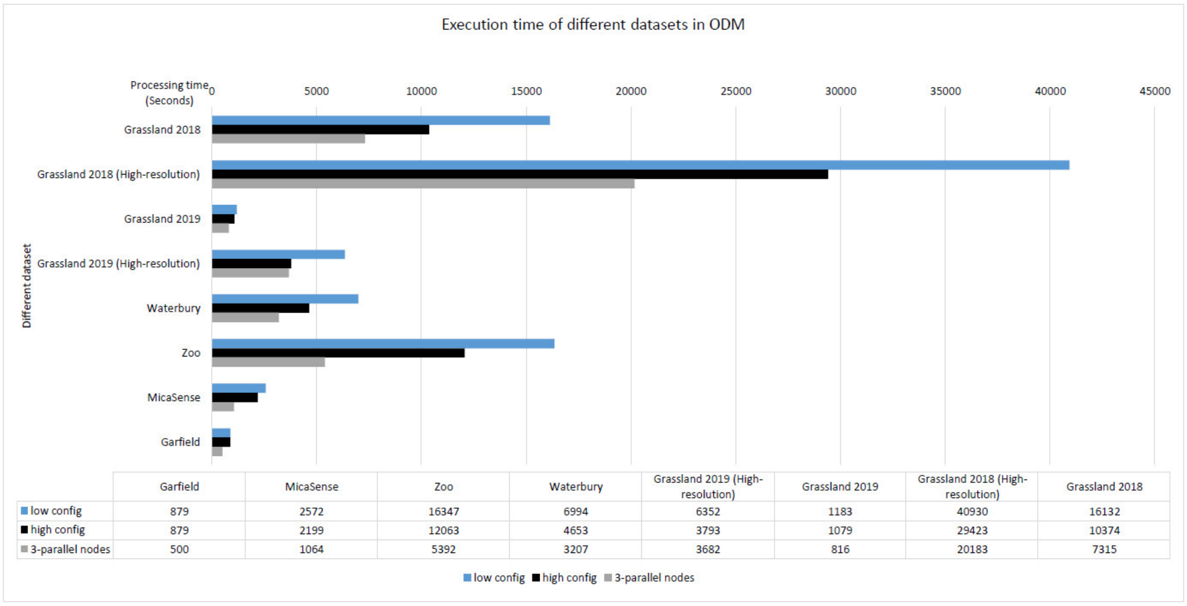


Figure 7: View of Grafana after auto-scaling performed on OpenDroneMap Kubernetes deployment.

2019 respectively using the default and use-case specific parameters with the three configurations stated in Table 3. The use-case specific parameters were set to generate an orthophoto with the sub-centimeter resolution of 0.3 cm/pixel from the acquired imagery. Whereas, the remaining four bar chart results, display the execution time (in seconds) of the open-source RGB and multispectral datasets respectively.

Initial results showed that for larger datasets (Grassland 2018), it took more than 11 hours to generate a high-resolution orthophoto (use-case specific) with the low configuration node; while the execution time was reduced to nearly 8 hours with the high configuration node. This is further proven when the process was run parallel on three workers nodes, the execution time was almost halved in comparison to the low configuration node. The results also demonstrated that modification of the parameters such as increasing the orthophoto and DEM resolutions from the default 5 cm/pixel to 0.3 cm/pixel required more execution time despite the size of the dataset (Grassland 2018 and Grassland 2019). This shows that split-and-merge or parallel processing can provide faster photogrammetry products.

Another observation was that running distributed split-merge on an up-scaled set up (higher computation power) led to shorter execution time, which can be seen especially in large datasets. However, in the case of smaller datasets (Grassland 2019 and



	Garfield	MicaSense	Zoo	Waterbury	Grassland 2019 (High-resolution)	Grassland 2019	Grassland 2018 (High-resolution)	Grassland 2018
low config	879	2572	16347	6994	6352	1183	40930	16132
high config	879	2199	12063	4653	3793	1079	29423	10374
3-parallel nodes	500	1064	5392	3207	3682	816	20183	7315

Figure 8: Computation time (in second) for the system evaluation based on the six datasets, including the use-specific cases for Grassland 2018 and 2019.

Garfield), the computation time difference between single and scaled parallel nodes was not as obvious. It was also observed that with a small dataset, sometimes the distributed execution took more time than expected. The reasons behind the delay were the splitting and merging steps, as well as the network and I/O delays.

Table 4: Comparative evaluation of Kubernetes HPA before and after modifications in timeout values and custom metrics type: object vs. pod.

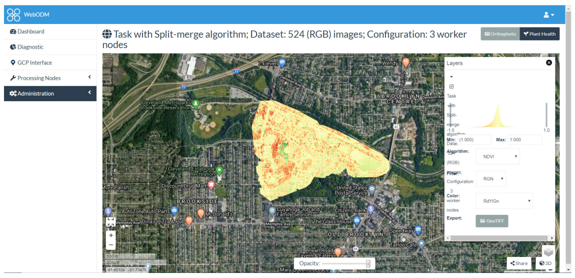
No. of replicas [from-to]	Before modifications response time (seconds) [object type]	After modifications response time (seconds) [pod type]
[2-1]	206	76
[1-2]	-	41
[2-3]	102	85
[3-1]	371	54
[1-3]	54	43
[3-2]	399	-

Critical reflections: The default timeout values in HPA and Prometheus led to more auto-scaling delay as shown in Table 4. Before modifying the delay, times were noted based on the experiments rather than the actual logic of Equation (2). As custom metrics with "object" type fails to calculate replicas count according to it. It was also unable in up-scaling from 1 to 2 replica count. However, these failures can be handled up-to a certain extent by specifying pod custom metric type in HPA because it is consistent with Equation (2). This custom metric comes with the limitation of not being able to handle downscaling of 3 to 2 replicas count and end up running unnecessary ClusterODM instances. Apart from HPA implementation problems, the ODM algorithm limits itself with JPEG and TIFF files as input and no-support for GPUs.

6 CONCLUSION AND OUTLOOK

A novel approach of deploying a photogrammetry toolkit on a Kubernetes managed on-premise cloud computing cluster was proposed and evaluated in this work. Comparison of generating orthophoto with scaled parallel execution to the singular execution displays feasibility of drone-acquired imagery mapping using the Kubernetes HPA method. In all the scenarios, the combination of auto-scaling and parallel execution had proven to be faster than single node executions. HPA turned out to be a promising Kubernetes feature with respect to the orthophoto generation time for various datasets. The evaluation results showed that total orthophoto generation time can be reduced with parallel processing, while auto-scaling was achieved by fusing open-source tools in ODM. This combination allowed the system to run faster and more resource-efficient and provided a better picture to the cloud service providers in preparation of an agreement with clients. After completion of the processing, the online "cloud-optimized-

GeoTIFF" feature of the latest ODM project can be utilized to calculate the normalized difference vegetation index (NDVI) online which is shown in Figure 9. This client-side resource-saving feature is a really good show-case of a low-cost end-to-end system.



Task with Split-merge algorithm; Dataset: 524 (RGB) images; Configuration: 3 worker nodes

Figure 9: Server-Side NDVI calculation from the processed dataset using scaled nodes in WebODM portal.

In the future, further evaluation with larger datasets can be performed to get a better idea about the system saturation point. Security and backup, aspects need to be considered to provide cloud service-level-agreement with the proposed approach. To conclude, the proposed scheme and the initial results represented a show-case in setting up a cloud-based web-portal for faster and resource-efficient drone imagery mapping on-premise.

ACKNOWLEDGEMENTS

This work is part of the SPECTORS project with project number 143081, which is funded by the cooperation program INTERREG Deutschland-Nederland. The authors would like to acknowledge the support of KnowH2O and Wageningen University & Research in consultation, and the teuto.net Netzdienste GmbH in technical support for the system.

REFERENCES

Bejjam, S. and Seshashayee, M. (2018). Big data concepts and techniques in data processing. *International Journal of Computer Sciences and Engineering*, 6:712–714.

Garrison, J. and Nova, K. (2017). *Cloud Native Infrastructure: Patterns for Scalable Infrastructure and Applications in a Dynamic Environment*. O'Reilly Media, Inc., 1st edition.

Groos, A., Bertschinger, T., Kummer, C., Erlwein, S., Munz, L., and Philipp, A. (2019). The potential of low-cost uavs and open-source photogrammetry software for high-resolution monitoring of alpine glaciers: A case study from the kanderfirn (swiss alps). *Geosciences*, 9:356.

Kang, S. and Lee, K. (2016). Auto-scaling of geo-based image processing in an openstack cloud computing environment. *Remote Sensing*, 8:662.

Lam, O. H. Y., Melville, B., Dogotari, M., Prüm, M., Vithlani, H. N., Roers, C., Becker, R., and Zimmer, F. (2019). Mapping invasive rumex obtusifolius in grassland using unmanned aerial vehicle. *Multidisciplinary Digital Publishing Institute Proceedings*, 30(1):34.

Lan, H., Zheng, X., and Torrens, P. (2018). Spark sensing: A cloud computing framework to unfold processing efficiencies for large and multiscale remotely sensed data, with examples on landsat 8 and modis data. *Journal of Sensors*, 2018:1–12.

Martinez-rubi, O., Nex, F., Pierrot-deseilligny, M., and Rupnik, E. (2017). Improving foss photogrammetric workflows for processing large image datasets. *Open Geospatial Data, Software and Standards*, 2.

MSV, J. (2016). Kubernetes: An overview. https://thenewstack.io/kubernetes-an-overview. Accessed: 2020-01-18.

OpenDroneMap (2020). Odm, open source project retrieved from. https://github.com/OpenDroneMap.

Pathak, A., Pandey, M., and Rautaray, S. (2019). Approaches of enhancing interoperations among high performance computing and big data analytics via augmentation. *Cluster Computing*, pages 1–36.

Sarajlic, S., Chastang, J., Marru, S., Fischer, J., and Lowe, M. (2018). Scaling jupyterhub using kubernetes on jetstream cloud: Platform as a service for research and educational initiatives in the atmospheric sciences. In *Proceedings of the Practice and Experience on Advanced Research Computing*, PEARC '18. Association for Computing Machinery.

Shah, J. and Dubaria, D. (2019). Building modern clouds: Using docker, kubernetes google cloud platform. In *2019 IEEE 9th Annual Computing and Communication Workshop and Conference (CCWC)*, pages 0184–0189.

Suryateja, P. S. (2020). Experimental analysis of openstack effect on host resources utilization. In Khanna, A., Gupta, D., Bhattacharyya, S., Snasel, V., Platos, J., and Hassanien, A. E., editors, *International Conference on Innovative Computing and Communications*, pages 11–19. Springer Singapore.

Tae-Hoon Kim, K. J. and Rajput, V. S. (2016). Adoption of container-based virtualization in it education. In *2016 ASEE Annual Conference & Exposition*. ASEE Conferences. https://peer.asee.org/26530.

Tanenbaum, A. and van Steen, M. (2016). *Chapter 1 of Distributed Systems - Principles and Paradigms*. Maarten van Steen.

Toffanin, P. (2019). *OpenDroneMap: The Missing Guide*. UAV4GEO, first edition.

Vithlani, H. N., Marcel, D., Melville, B., Prüm, M., Lam, O. H. Y., Becker, R., and Zimmer, F. (2019). Applicability of remote sensing workflow in kubernetes-managed on-premise cluster environment. *Multidisciplinary Digital Publishing Institute Proceedings*, 30(1):38.