REMOTE SENSING DATA ANALYTICS WITH THE UDOCKER CONTAINER TOOL USING MULTI-GPU DEEP LEARNING SYSTEMS

Gabriele Cavallaro[1], *Valentin Kozlov*[2], *Markus Götz*[2], *Morris Riedel*[1]

[1]Jülich Supercomputing Centre, Forschungszentrum Jülich, Jülich, Germany
[2]Steinbuch Centre for Computing, Karlsruhe Institute of Technology, Karlsruhe, Germany

ABSTRACT

Multi-GPU systems are in continuous development to deal with the challenges of intensive computational big data problems. On the one hand, parallel architectures provide a tremendous computation capacity and outstanding scalability. On the other hand, the production path in multi-user environments faces several roadblocks since they do not grant root privileges to the users. Containers provide flexible strategies for packing, deploying and running isolated application processes within multi-user systems and enable scientific reproducibility. This paper describes the usage and advantages that the *uDocker* container tool offers for the development of deep learning models in the described context. The experimental results show that *uDocker* is more transparent to deploy for less tech-savvy researchers and allows the application to achieve processing time with negligible overhead compared to an uncontainerized environment.

Index Terms— Containers, uDocker, multi-GPU, deep learning, classification, remote sensing.

1. INTRODUCTION

In this era of a growing number of earth observation satellite and aerial platforms the volume, variety and acquisition rate of remote sensed images have been dramatically increased. This introduced remarkable challenges that lie within the entire acquisition and processing data pipeline—i.e., the Vs of big data [1, 2]. The interpretation of remote sensing images is not straightforward and requires complex algorithms since their content depends upon various factors, e.g., the sensor resolution, the equipment unreliability, the type and amount of noise, etc. Furthermore, the increased data volume and demands of real-time applications require the use of high scalable and parallel processing approaches. While modern desktop computers and laptops having unprecedented performance, e.g., multi-core architectures and built-in accelerators, they are still limited in terms of computable problems due to their memory constraints and raw floating-point operations per second.

Having massive numbers of processors and memory available, multi-GPU systems can overcome these limitations and provide processing capacity that well exceed traditional laptops and work stations. Moreover, the utilized dedicated high-speed networks, such as InfiniBand, enable strong vertical and horizontal scaling of applications. Despite the impact of these new architecture on traditional simulation sciences, parallel computing is currently experiencing focus and advancements due to the current deep learning trend. Both of the latter domains influence each other in numerous ways [3] such as, among others, refreshed attention to hardware and performance engineering around tensor operations, the explorations of scalability boundaries as well as the envisioning simplified, parallel programming models. At the same time, deep learning has made revolutionary achievements for the analysis of remote sensing images [4] possible.

Nevertheless, there are major factors that prevent multi-GPU and -user systems from being the platform of choice for researchers developing new deep learning models. It starts with getting access and computing time on these machines, but goes well beyond that. Users who develop deep learning workflows want to focus first and foremost on the purpose and the realization of their analysis pipeline. This in turn requires them to be in full control of their programming library stack and underlying system. However, in multi-user systems administrators are usually in charge of the maintenance and supervision of the systems; users do not have privileges to install or modify software and can therefore not easily catch up with up-to-date libraries. Instead, a user is usually faced with either a long-pending installation request or a user-land compilation of their custom software, which needs to be repeated for every actively working scientist of the research collaboration. At the same time, users seek reproducible science through computational mobility [5], i.e., the possibility to restore a software environment as closely as possible to verify and continue past research. Containers have drawn a lot of attention in recent years in the parallel computing domain since they allow the simplification and acceleration of the application build and deployment process. Furthermore, for users working in the parallel computing and deep learning domains, containerization offers the benefits of scalability without performance penalties compared to traditional virtual machines. Gomes *et al.* [9] have proposed *uDocker* , a novel container tool that allows the execution of Docker containers without

doi:10.2760/848593

Table 1. Comparison of state-of-the art container technologies suitable for execution on multi-user systems. The displayed table is a heavily modified variant of the previous work from Priedhorsky et al. [6] and Kurtzer et al. [5].

	Docker [7]	Singularity [5]	Shifter [8]	Charlie Cloud [6]	*uDocker* [9]
Privilege model	Root daemon	SUID/UserNS	SUID	UserNS	chroot-like
Current production Linux distros support	✗	✓	✓	✗	✓
No privileged or trusted daemon	✗	✓	✓	✓	✓
Access to the host filesystem	✓	✓	✓	✓	✓
Support for GPU	✗ [a)]	✓ [b)]	✗	✗	✓ [b)]
Support for MPI	✓	✓	✓	✓	✓ [c)]
Pulling from Docker Hub	✓	✓	✓	✓	✓
No system admin intervention required	✗	✗	✗	✗	✓
No escalation of permissions	✗	✓ [d)]	✓	✓	✓
Works with all HPC schedulers	✗	✓	✗	✓	✓

[a)] Can be realized by installing nvidia-docker runtime
[b)] Experimental feature
[c)] Container MPI version has to match the HPC one
[d)] There was a number of high severity security issues in Singularity

the necessity for administrative privileges, i.e., no need to install additional system software. This paper describes the usage of *uDocker* [9] container tool for the development of an exemplary deep learning model for remote sensing images pixel-wise classification. The experimental results show that *uDocker* is comparable to a bare-metal installation, only entailing around a 1% computation time overhead, while simplifying the setup drastically.

2. BACKGROUND

The development of applications on shared multi-GPU systems is a difficult operation which requires that the system administrators build ad-hoc environments, i.e., software modules. For instance, a simple application upgrade can demand updating several environment modules. Furthermore, multi-GPU applications usually require running across multiple platforms and environments and utilize site-specific resources while resolving complicated software-stack dependencies. These are time-consuming tasks which add more work to the administrators, who have to maintain the multi-user systems and assure that the users have the tools and support to make the most efficient use of the computing resources.

Inspired by the shipping containers in inter-modal global transport, i.e., standardized containers that can be directly transferred with different shipping methods without any additional preparation, software containers utilize the same strategy. They are in many ways the next logical progression from virtual machines [10]. However, containers are a type of lightweight virtualization technology, which encapsulates system environments into standard units of software that are: portable, easy to build and deploy, have a small footprint, and low runtime overhead. As researchers started embracing containers for science, their usage within parallel computing environments grew as well. Despite all the issues of using containers in multi-user systems, they have been developed to meet their needs including security, MPI compatibility and GPU access. Since the introduction of Docker [7], the development of technologies associated with containers raised. Table 1 shows the most leading container technologies with their main features.

3. UDOCKER CONTAINER

uDocker is a software technology that allows the reuse and execution of Docker containers in user mode [9]. A container in turn is an isolated environment mimicking an operating system and its installed software. It is created by making use of layering file system, where every change made to the image, e.g., the installation of a software, adds a new layer to the image. These layers can then be shared and reused or further extended to a customized versions. In contrast to traditional virtualization technology, like virtual machines, containers are often referred to as light-weight, as they do not "pull-in" the entire operating system, but reuse the host operating system kernel when executed. This does not only reduce the memory footprint of such a container, but also reduces the computational performance impact.

While there is a plethora of containerization technologies currently being developed and researched on, first and foremost Docker, they often have a particular usage scenario in mind, requiring intervention of a priviliged user, e.g., administrator, for at least one step of the creation or execution of an isolated environment. In multi-user systems, especially with multiple GPUs used for deep-learning, the assumption about privileges does not hold in hindsight of security issues and direct use of containerization technology is not viable. At the same time, the operations requiring containers to have administrator privileges, are in most cases not needed for (scientific) deep learning application like in remote sensing. There-

doi:10.2760/848593

fore, *uDocker* attempts to offer a compromise between both worlds.

Through a second layer of virtualization technologies, like PTRACE, UserNamespaces or `libfakechroot` [9], it emulates as many container technology functions as possible in an unpriviliged userland environment. While actual privileged operation, like access to high-ports or password management, will obviously fail, enabling security by design, access to deep-learning essentials like GPUs is possible. *uDocker* offers multiple execution modes, where each is referring to a particular realization of the secondary virtualization technology—*P* uses PTRACE, *F* `libfakechroot`, *R* UserNamespaces and *S* Singularity as engine. This alongside numeric levels for the execution modes, e.g., *P1* or *P3*, allow the fine tuning of the *uDocker* for the particular execution scenario.

uDocker syntax is designed to be very similar to Docker's interface in order to allow users to reuse documentation material and container technology manager to transfer their knowledge. At the same time, *uDocker* is able to reuse openly published Docker containers, e.g., on DockerHub, enabling a rapid development cycle, custom extension and exchange with large community. A remote sensing scientist, who developed a new classification algorithm for example, may want to establish it either as a generally usable service or open-source it alongside a publication. In this scenario, the respective container can be created directly using *uDocker* on his experimentation device and then later shared with other scientists to verify or build on the results.

4. EVALUATION

4.1. Experimental Setup

The experiments have been performed on the LSDF setup, which is a single computer with all hardware available locally. Its configuration parameters are listed in Table 2. The operating system is a RedHat Enterprise Linux 7.5, CUDA Toolkit 9.0.176 and cudnn 7.0.5 library are installed system-wide. We first created virtual environment and run baremetal tests by means of Keras 2.2.2, TensorFlow 1.8.0 (GPU), and the neural network code[1] described in the next section. Versions of all relevant Python packages were fixed with `pip freeze`, so that exactly same versions are used in all the tests, including created docker image[2]. Note, that the utilized Python versions are slightly different in case of baremetal and the docker image: 2.7.5 and 2.7.12 respectively. *uDocker* is executed in 'F3' mode (Fakechroot) with '`--nvidia`' flag specified, devel branch of *uDocker* from GitHub is used.

[1]Source code: https://github.com/vykozlov/semseg-bids19
[2]https://hub.docker.com/r/vykozlov/semseg/, tag 'bids19-gpu'

Table 2. LSDF setup used in the experiments.

CPU	RAM	Nvidia GPU (driver version)
2 × Intel Xeon E5-2630 v3	128 GB	4 × K80, 12 GB (396.26)

4.2. Dataset and Deep Learning Model

The Vaihingen dataset [11] includes 33 orthorectified image tiles acquired by an aerial camera (i.e., infrared, green and red bands) over the town of Vaihingen (Germany) [3]. Since this dataset was released as a benchmark for a 2D semantic labeling contest, only 16 out of the 33 tiles are annotated (i.e., at pixel level with a spatial resolution of 9 cm). For the experiments, the annotated tiles that are used for the training and validation have ID= 1,3,5,7,11,13,17,21,26,28,34,37 and ID= 30,32, respectively. The semantic segmentation task involves the discrimination of 6 land-cover classes: *impervious surfaces* (i.e., roads, concrete surfaces), *buildings*, *low vegetation*, *trees*, *cars* and a class of *clutter* representing uncategorizable land covers (i.e., excluded in the prediction). The training data was randomly augmented using 90 degree rotations and horizontal and vertical flips.

The deep learning model is a 50-layer Residual Network (ResNet) [12] that was adapted into a Fully Convolutional Network (FCN) with connections from the last 3232, 1616, and 88 layers of the ResNet: ResNet50 FCN [4].

The model was trained using a random initialization for 20 epochs with 4166 samples per epoch (2083 original images and 2083 augmented) with a batch size of 16 using the Adam optimizer with Keras default settings (e.g., a learning rate of 0.001). The network takes 256×256 windows of data as input. To generate predictions for larger images, we made predictions over a sliding window (with 50% overlapping of windows) and stitched the resulting predictions together.

4.3. Results

The code allows to use more than one GPU for training by means of Keras' `multi_gpu_model` function, we therefore perform training on one, two, and four GPUs for every case. Each experiment is run three times under the same conditions in *baremetal* installation and via *uDocker* . In order to compare our results we used mean value and estimated standard error for the sample calculated based on the three runs. Every run consists of 20 epochs of training. Results for the total training time are shown in Table 3. As one can see, in

[3]http://www2.isprs.org/commissions/comm3/wg4/semantic-labeling.html
[4]https://www.azavea.com/blog/2017/05/30/deep-learning-on-aerial-imagery/

doi:10.2760/848593

either case we see no statistically significant difference between *baremetal* and *uDocker* modes of running. There is also a clear performance improve in processing time when using four over one GPU (the higher variance is due to the data parallelization). The scaling with number of GPUs is however imperfect. This can be attributed to the communication overhead of the way Keras synchronizes weight gradients between multiple GPUs in the training's backpropagation step.

Table 3. Total training time of the neural network. Each result is an average of three runs with its standard error. Every run takes 20 epochs of training on either one, two, or four GPUs.

Number of GPUs	Training time, s	
	baremetal	*uDocker*
1	3710 ± 10	3730 ± 10
2	2390 ± 30	2360 ± 16
4	1860 ± 40	1880 ± 10

We note here, that *uDocker* also allows to pass environment settings at container instantiation phase, therefore one can e.g., specify which GPU card to use by setting CUDA_VISIBLE_DEVICES environment.

5. CONCLUSIONS

This paper describes the usage of the *uDocker* container tool within a multi-GPU system for the development of a deep learning classification task. *uDocker* allow to run the classifier in a Docker container without using Docker, root privileges and additional system software. It is run as a normal user without the intervention of the system administrators. The paper shows that researchers can adopt *uDocker* to facilitate the deployment of new analytical models and workflows on multi-user systems and enable scientific reproducibility. Furthermore, the experimental results demonstrated that the overhead introduced by the container is negligible when compared to an uncontainerized environment.

6. ACKNOWLEDGMENTS

uDocker is being developed within DEEP HybridDataCloud project, which receives funding from the European Union's Horizon 2020 research and innovation programme under agreement RIA 777435.
This project has received funding from the European Union's Horizon 2020 research and innovation program under the Grant Agreement No. 754304 DEEP-EST.
The Vaihingen data set was provided by the German Society for Photogrammetry, Remote Sensing and Geoinformation (DGPF) [11] http://www.ifp.uni-stuttgart.de/dgpf/DKEP-Allg.html.

REFERENCES

[1] M. Chi, A. Plaza, J. A. Benediktsson, Z. Sun, J. Shen, Y. Zhu, SunZhongyi, J. Shen, and Y. Zhu, "Big Data for Remote Sensing: Challenges and Opportunities," *Proc. IEEE*, 2015.

[2] Y. Ma, H. Wu, L. Wang, B. Huang, R. Ranjan, A. Zomaya, and W. Jie, "Remote Sensing Big Data Computing: Challenges and Opportunities," *Future Generation Computer Systems*, vol. 51, pp. 47–60, 2015.

[3] T. Ben-Nun and T. Hoefler, "Demystifying Parallel and Distributed Deep Learning: An In-Depth Concurrency Analysis," 2018.

[4] X. X. Zhu, D. Tuia, L. Mou, G. S. Xia, L. Zhang, F. Xu, and F. Fraundorfer, "Deep Learning in Remote Sensing: A Comprehensive Review and List of Resources," 2017.

[5] G. M. Kurtzer, V. Sochat, and M. W. Bauer, "Singularity: Scientific Containers for Mobility of Compute," *PLoS ONE*, 2017.

[6] R. Priedhorsky, T. C. Randles, and T. Randles, "Charliecloud: Unprivileged Containers for User-Defined Software Stacks in HPC," *SC17: International Conference for High Performance Computing, Networking, Storage and Analysis*, 2017.

[7] D. Merkel, "Docker: Lightweight Linux Containers for Consistent Development and Deployment," 2014.

[8] D. M. Jacobsen and R. S. Canon, "Contain This, Unleashing Docker for HPC," *Cray User Group 2015*, 2015.

[9] J. Gomes, E. Bagnaschi, I. Campos, M. David, L. Alves, J. Martins, J. Pina, A. López-García, and P. Orviz, "Enabling Rootless Linux Containers in Multi-User Environments: The Udocker Tool," 2018.

[10] J. Smith and R. Nair, *Virtual Machines: Versatile Platforms for Systems and Processes*. 2005.

[11] M. Cramer, "The DGPF-Test on Digital Airborne Camera Evaluation Overview and Test Design," *PFG Photogrammetrie, Fernerkundung, Geoinformation*, vol. 2010, no. 2, pp. 73–82, 2010.

[12] K. He, X. Zhang, S. Ren, and J. Sun, "Deep Residual Learning for Image Recognition," in *2016 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, 2016.

doi:10.2760/848593