CHAPTER 13

CYBELE: A Hybrid Architecture of HPC and Big Data for AI Applications in Agriculture

Naweiluo Zhou 0000-0001-9329-4500 , Li Zhong 0000-0002-6473-1633 ,
Dennis Hoppe 0000-0001-8976-8603, Branislav Pejak 0000-0003-1094-8770,
Oskar Marko 0000-0001-6683-7178, Javier Cardona 0000-0002-9284-1899
Mikolaj Czerkawski 0000-0002-0927-0416 , Ivan Andonovic 0000-0001-9093-5245
Craig Michie 0000-0001-5132-4572, Christos Tachtatzis 0000-0001-9150-6805,
Emmanouil Alexakis 0000-0002-8068-4152, Philip Mavrepis 0000-0002-2785-7106
Dimosthenis Kyriazis 0000-0001-7019-7214, Marcin Pospieszny 0000-0001-7715-2684

13.1 Introduction-- Vision and Challenges

Big-data analytics hosted by Cloud clusters are becoming more data-intensive and computation-intensive, mainly due to development in Artificial Intelligence（AI）applications. High Performance Computing (HPC) systems are often used to execute large-scale programs, such as programs performing engineering, scientific or financial simulations that demand low latency and high throughput. By taking advantage of HPC systems, AI applications have the potential to achieve better performance compared to that on Cloud. In general, an AI application incorporates a complex list of software and therefore its user needs flexibility to customize the working environment. However, HPC systems, supporting multi-tenant environments, typically provide complete stacks of software packages and often do not allow user customization in contrast to Cloud systems. Containerization could offer a solution for provisioning flexible execution environments for AI applications on HPC clusters.

Containerization is a virtualization technology. Rather than simulating a holistic Operating System (OS) as in a Virtual Machine (VM), a container only shares its host OS and utilizes its

dependencies. Typically, one container encapsulates one application and the corresponding libraries and configurations in an isolated environment thus enabling compatibility and portability. A container is booted by its host as a process, hence the start-up time of a container is similar to a native application [1–7]. Deployment of containers should focus on environment compatibility on HPC clusters, where a heavy stack of software packages is included, making the size of containers relatively large. Per contra, containers on Cloud, which are dedicated to run micro-services [8], are more light-weight.

Applications on Cloud are often hosted in containerized environments. Jobs on a Cloud cluster are often managed by a container orchestrator, such as Kubernetes [9] which is one of the most popular container orchestrators. An HPC cluster is usually controlled by a workload manager, e.g., TORQUE [10] or SLURM [11]. Typical HPC jobs are large workloads whose execution time is often fixed. Whereas jobs on Kubernetes are often small programs that run continuously to provide services. An HPC workload manager lacks its efficiency in micro-service supports and deeply-integrated container management in which a container orchestrator manifests its advantages. Nevertheless, a container orchestrator on its own cannot address all the requirements of HPC systems, thereby cannot replace the existing HPC workload manager.

The EU-founded research project CYBELE [1] enables convergence of HPC and Cloud technologies being applied in the domain of agriculture that has become a trending field for AI applications. The Cloud clusters host long-running service applications that supply straightforward graphic interfaces for designing application pipelines and workflows as presented in Chapter 13. The applications that require intensive computation or fast-data access

[1] CYBELE: Fostering Precision Agriculture and Livestock Farming through Secure Access to Large-Scale HPC-Enabled Virtual Industrial Experimentation Environment Empowering Scalable Big Data Analytics. https://www.cybele-project.eu/

are scheduled for execution on HPC clusters where program performance can be significantly enhanced. In this chapter, a hybrid architecture that enables the synergy of TORQUE or SLURM managed HPC systems and Kubernetes managed Cloud systems is given. Two use cases are detailed in order to illustrate how this architecture can be beneficial to AI applications and how AI applications should be adapted in order to scale on HPC systems.

13.2 Background

Agriculture is the basis of every economy worldwide [12]. In recent years, AI technologies have been applied in various domains of agriculture, such as crop yield prediction, disease detection, soil content sensing and crop monitoring, to better understand growth of crops and improve production efficiency [13]. Monitoring data about crops automatically collected by sensors or satellites, such as the images of fields, can be continuously processed by AI applications, and when a problem pattern is recognised, the applications can recommend immediate actions to tackle the issue [14]. Pattern recognition [15] is one of the most important technologies in AI and it is the underlying technology of many decision-support systems that help farmers optimize their production.

13.2.1 AI in Big Data Analytics on Cloud

Advancement of Machine Learning (ML) and Deep Learning (DL) has brought more accurate solutions to applications and enhances application performance significantly. ML/DL models have been implemented by industry and academia in Cloud clusters, where large datasets are crunched and algorithms are trained, thus allowing efficient scaling at a low cost.

Different parallelization algorithms and strategies have been developed in DL, where the

three predominant parallelization methods are commonly adopted, namely data parallelism, model parallelism and pipeline parallelism [16]. In *data parallelism*, each device in the cluster loads an identical copy of the DL model, and the whole training dataset is split into non-overlapping chunks which are fed into the model replicas of each device. Each model replica is trained on the data chunks, and the model parameters from different devices are synchronized after each step. However, when the model becomes too complicated and involves a large number of parameters, synchronization of the parameters on model replicas often becomes the bottleneck [17]. In contrast to data parallelism, *model parallelism* splits the model and each device loads a part of the model. The input is first fed to the device which holds the input layer, and subsequently its output is passed to the device that holds the next layer in the *forward pass* [18]. In the *backpropagation* [18] phase, the gradients are computed starting from the device which holds the output layer of the DL model, then this change is consequently passed to the previous layers. In theory, model parallelism could solve the problem raised in data parallelism. In practice, heavy communication among different devices is the bottleneck. *Pipeline parallelism* was thereby proposed. It combines both data parallelism and model parallelism, where not only the model is split into different devices, but also the dataset is fractured into chunks. This methodology is commonly adopted by most DL frameworks or libraries e.g., TensorFlow [19], PyTorch [20]. In addition to the types of parallelism supported by the above DL parallel frameworks, Apache Spark [21], as unified analytic engine, is widely used for large-scale data processing and provides high-level APIs helping users create and tune practical ML pipelines [2].

Computing resources and applications on Cloud are managed by orchestrators. Kubernetes

[2] A ML pipeline is a workflow running a sequence of algorithms to process and learn from data.

[9], as the trending container orchestrators, is based on a highly modular architecture that abstracts the underlying infrastructure and allows internal customization, such as deployment of different software defined network or storage solutions. It includes a powerful set of tools to control the life cycle of applications, e.g., parameterized redeployment in case of failures, state management, etc. It also supports various big-data frameworks (e.g., Apache Spark, Hadoop MapReduce [22] and Kafka [23]) and can be connected with Ansible [24] which is a popular software orchestration tool in Cloud clusters.

13.2.2 AI on HPC Systems

Compared to Cloud systems, HPC systems show the advantages in computational power, storage and security [25] . Exploiting HPC infrastructures for ML and DL training is becoming a topic of increasing importance [26]. AI applications are usually developed with high level scripting languages or frameworks, e.g., TensorFlow [19], and PyTorch [20], which often require connections to external systems to download a list of open-source software packages during execution. For example, an AI application written in Python cannot be compiled into an executable that includes all the dependencies necessary for execution as in C/C++. Therefore, the developers need flexibility to customize their execution environments. HPC environments, especially HPC production systems, are often based on closed source applications and their users have restricted account privileges and security restrictions [27], for instance access to external systems is blocked. Containerization enables easy transition of AI workloads to HPC while taking the full advantage of HPC hardware and the optimized libraries of AI applications, without compromising security of HPC systems.

HPC workload managers, such as TORQUE or SLURM, have been well designed to schedule conventional HPC parallel applications, e.g., Message Passing Interface (MPI) [28,

29] applications. Implementation and deployment of DL solutions on HPC systems are challenging in terms of parallelization, scheduling, elasticity, data management and portability. With a large number of DL models having a complex structure and being trained with huge amounts of data, adapting DL/ML models in order to be well scheduled by the workload managers, to achieve high parallelization is the *sine qua non* of leveraging optimal performance on HPC systems.

13.3 Hybrid Big Data and HPC Resource Provisioning for AI Applications in CYBELE

The CYBELE project proposed and implemented a hybrid architecture that enables convergence of Cloud and HPC clusters. This section gives a brief introduction to this architecture and more details are described in [30–33].

The architecture illustrated in Figure *1* consists of an HPC cluster with either TORQUE or SLURM as its workload manager and a Cloud cluster with Kubernetes serving as its container orchestrator. All the nodes on the left side are VM nodes. VM nodes rather than bare-metal nodes are more often utilized on Cloud clusters. For simplicity, Figure *1* only shows a limited number of nodes and a single HPC partition. It is worth noting that the architecture is not limited to this number of nodes and can be extended to support more nodes and HPC partitions. The only requirement of this architecture is the existence of one or more shared login node/nodes as highlighted in red dashed line in the middle of Figure *1*. The login node is a Kubernetes worker node and meanwhile functions as a login node for TORQUE or SLURM. Jobs are submitted via the login node in the form of *yaml* scripts, in which the TORQUE or SLURM job scripts are embedded. The embedded scripts are abstracted and scheduled to execute on the HPC cluster by a plugin named Torque-Operator [30, 31] or WLM-Operator [34]. The WLM-Operator implemented by Sylabs bridges Kubernetes with SLURM and the

Torque-Operator developed within the CYBELE project connects Kubernetes with TORQUE (see [30] for more details). The normal Kubernetes micro-service jobs are still deployed via the login node. They are, however, scheduled to run on the Cloud Cluster. The advantages that this architecture bring are four-fold:

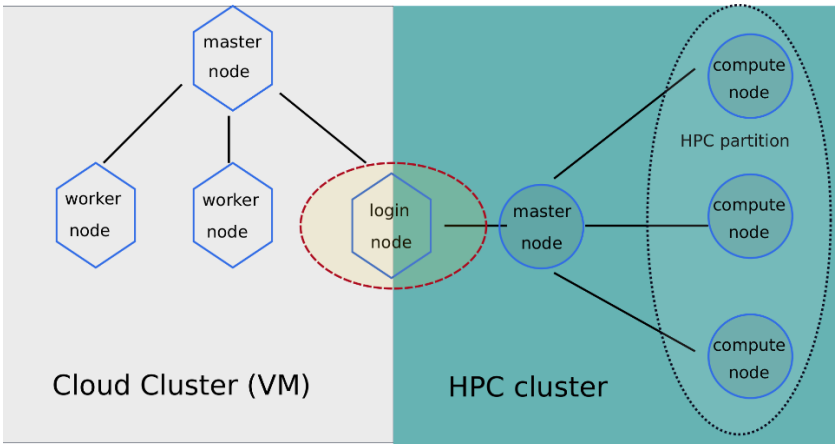


Figure 1 The hybrid architecture.

(1) It provides a unified interface for users to access the Cloud and HPC systems. All jobs are submitted in the form of *yaml* scripts.

(2) It keeps the modification of HPC systems to the minimum. HPC systems only need to install Singularity on top of their current environments. Singularity [35] is the *de facto* standard container runtime for HPC systems.

(3) Long-running micro-service applications stay on the Cloud cluster and computation-intensive or data-demanding programs that have fixed execution time are scheduled to the HPC cluster.

(4) Users have the flexibility to customize their working environment on Cloud clusters to develop their applications.

13.4 Parallelism and Deployment of AI Applications on HPC Systems

The key driving factor of success of AI, especially DL applications, is the optimization and

scalability of training [26]. New breakthroughs in model accuracy were achieved by increasing the size and the complexity of models. Relatively simple model structures have been replaced with more sophisticated architectures [36]. Moreover, the development of AI technologies has been pushed by raising amount of training data. DL models are data intensive, and model accuracy can be improved significantly with larger-quantity of training data [37]. Therefore, the availability of massive compute resources as in HPC infrastructures is the key to enable training of complex models with vast amount of data in reasonable time. This section showcases two AI applications in the field of agriculture, which are developed within the CYBELE project, namely the **Pilot Soybean Farming** and **Pilot Wheat Ear**. Table 1 summarizes the descriptions of the two pilots and the details are presented in Section 14.4.1 and Section 14.4.2.

Table 1. Descriptions of Pilot Soybean Farming and Pilot Wheat Ear

Names	**Descriptions**	**ML applications**	**Parallel models**
Pilot Soybean Farming	Yield prediction for soybean farming	yes	Open MPI, Apache Spark
Pilot Wheat Ear	Provisioning of autonomous robotic systems within arable frameworks	yes	PyTorch

13.4.1 Pilot Soybean Farming

This section describes the Pilot Soybean Farming and its methods of parallelization and deployment on HPC systems.

13.4.1.1 Pilot Description

Pilot Soybean Farming focuses on the application of ML in soybean farming. Its goal is to develop a prediction algorithm that is able to infer hidden dependencies between the input parameters and the yield. The input dataset consists of three Sentinel-2 [38] images for each of the 3-years analyzed. This is 9 satellite images in total and each having 12 spectral bands with a resolution of 10m, 20m and 60m, depending on the channel. Additionally, yield monitor data from Austrian soybean farms is used as the ground truth for algorithm training (total size: 614 MB). The application is composed of two procedures, i.e. the data preprocessing procedure followed by the training procedure. The input for the second procedure is a tabular file in the *xlsx* format yielded from the first procedure (total size: 7.3 MB). Both procedures are written in Python script and containerized in Singularity, as Singularity container can garantee environment compatibity.

13.4.1.2 Application Parallelization for HPC Systems

MPI is a conventional standard for parallel processing on HPC systems. The first procedure of Pilot Soybean Farming adopts MPI to speedup data processing. More specifically, mpi4py library, which is the Python support for MPI, is introduced to the program. In this case, the MPI launcher of the host system invokes Singularity container and Singularity launches the MPI application within the container, which is the Python script with MPI support in this case.

The dataset produced by the first procedure is fed as the input for the ML training model. The second procedure includes modules for image stacking, cropping to the field boundaries, yield map interpolation from point measurements and ML models for yield prediction. Its ML pipeline consists of the following steps:

1. The dataset is loaded into an Apache Spark data frame that can significantly decrement data access time. Since the ML model is coded in Python, Spark is implemented in PySpark library.
2. Filter out missing/erroneous records and split the training and testing datasets. In this step, the data is now ready to proceed further down the pipeline to feed the ML models, i.e. Linear Regression [39], Decision Tree Regressor [40] , Gradient Boosted Trees Regressor [41] and Random Forest Regressor [42]. The best results are selected out of the four aforementioned models.
3. The evaluation of the prediction models is performed over the test dataset providing a fair generalization approximation.

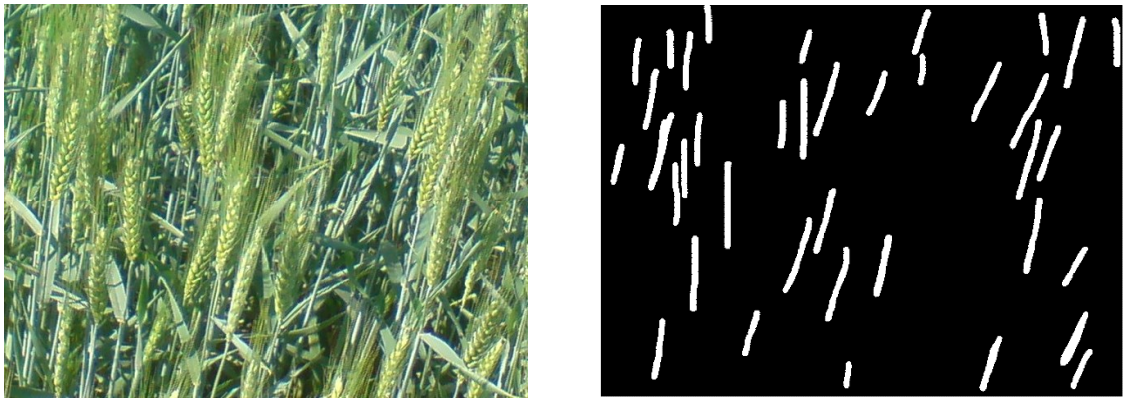
Description of the ML training procedure of Pilot Soybean herein is to demonstrate a method of scheduling a big-data framework to an HPC environment predominated by a workload manager. It is out of the scope of this chapter to disccuss prediction accuracy of various ML models. There are three ways of deploying Spark as Standalone engine, though Hadoop YARN [43] and via MapReduce [44]. Standalone deployment is favored in the HPC environment. In this case, HPC workload manager TORQUE serves as the Spark scheduler in preference to Mesos [45] or YARN. Spark standalone package is not required on an HPC system, and instead it is encapsulated inside Singularity. The master and workers of Spark are lauched within the container processes rather than natively on the HPC cluster.

13.4.2 Pilot Wheat Ear

This section depicts the Pilot Wheat Ear and its methods of parallelization and deployment for HPC systems.

13.4.2.1 Pilot Description

Pilot Wheat Ear is targeted for provisioning of autonomous robotic systems within arable frameworks [46]. More specifically, the application provides a framework for automatic detection and count of wheat ear kernels in fields from the images collected by sensors on ground so that it enables crop yield prediction and can suggest decisions for sales planning. A DL model, utilizing image segmentation based on U-Net architecture [47], is trained to automatically detect the wheat ears from a series of RGB images (138 images, 95 MB in total) captured in the crop fields in Serbia. Figure 2 illustrates an example of an input wheat-ear image and its output image. The output will serve as an input for yield prediction, which will relate the number of ears and their size to the actual yield.



(a) input wheat ear image (b) output ear counts

Figure 2 Input image of wheat ears and the output of ear counts generated by the DL model.

13.4.2.2 Application Parallelization for HPC Systems

In recent years, a wide range of potential solutions has been brought for the challenge of image segmentation. The solution can consist of a specific neural network architecture and a set of techniques applied during training and inference. Typically, the decisions will be made based on a number of experiments. A high degree of parallelism can greatly speed up this process and allow more selection of the network parameters.

In Pilot Wheat Ear, its DL model is written in Python script based on Fastai/PyTorch [20, 48] which are widely-used DL libraries. PyTorch can get scale-up by using its method torch.set_num_threads() or the environment variable OMP_NUM_THREADS to adjust the number of threads. The library is containerized in Singularity, correspondingly each Singularity process boots one PyTorch thread. The workload manager TORQUE schedules all the Singularity processes to the available HPC cores.

13.5 Performance Evaluation for Pilot Soybean Farming and Pilot Wheat Ear

The hybrid architecture elucidated in Section 14.3 is implemented on a testbed with relatively small amount of nodes. The testbed technical specifications are indicated in Table 2. The Cloud cluster comprises 4 VM nodes and the HPC cluster is composed of 3 bare-metal nodes. The HPC cluster has TORQUE installed as its workload manager while the Cloud cluster is managed by Kubernetes. Docker [49], as one of the most widely-used container runtime on Cloud, is utilized to continuously run the CYBELE service programs as described in Chapter 13. Due to security concerns, Docker container runtime is restricted to the Cloud cluster. AI applications are containerized in Singularity images. The core applications required for the Cloud and HPC systems are listed in Table 3. Flannel[3] is deployed for the container network interface (CNI) of Kubernetes. Without this additional network model, containers located in separate nodes could not establish communications among each other.

[3] https://github.com/flannel-io/flannel

Table 2 Technique specifications of the testbed.

Cluster name	HPC Cluster(bare-metal)	Cloud Cluster (VM)
Total number of nodes	3 (2 compute nodes)	4 (3 worker nodes)
Number of cores	20 (10 cores per CPU, 2 CPU per node)	2 cores per node
RAM per node (NUMA)	128 GB	8 GB
CPU frequency	Intel(R)Xeon(R) CPU E5-2630 v4, 2.20GHz	Intel i7 9xx (Nehalem Core i7, IBRS) 2.79GHz
Operating System	Ubuntu 18.04.3 LTS	Ubuntu 18.04.3 LTS
VM type	-	QEMU 2.11.1, KVM 2.11.1

Table 3 The list of core applications on the testbed.

Cluster type	HPC Cluster	Cloud Cluster
Orchestrator	TORQUE	Kubernetes
Container runtime and interface	Singularity	Singularity, Docker, Singulairy-CRI[4]
Plugin	-	Torque-Operator, Flannel
Compiler	Golang compiler	Golang compiler
Parallel model	Open MPI	-

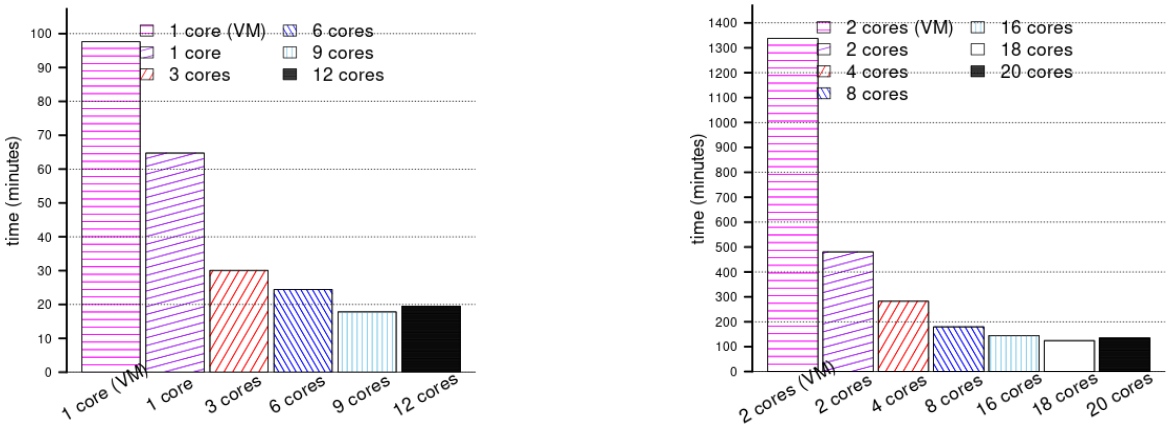
The results are compared between the Cloud cluster and the HPC cluster. In Pilot Soybean Farming, the size of the training dataset for the training procedure is relatively small, the performance evaluation of this pilot herein concentrates on the data preprocessing proedure which is computation-intensive and data-intensive. Pilot Soybean *Farming*

(b) Pilot Wheat Ear

[4] Singularity-CRI: it is Singularity-specific implementation of Kubernetes CRI (CRI: Container Runtime Interface).

Figure *3* compares the execution time with different levels of parallelism for data preprocessing procedure of Pilot Soybean Farming and Pilot Wheat Ear. It is worth noting that the execution time of the two pilots can vary from minutes to days depending on the amount of input data sets that are used for processing. The details are given in Table 4 and Table 5.

The 3-year data processed in Pilot Soybean Farming is organized in three independent groups. The number of process is, therefore, set to be a multiple of 3. However, for the sake of completeness, the results of a single process are also presented for both. On the HPC cluster, the execution time is measured with the number processes to be 3, 6, 9 and 12 processes, respectively. For Pilot Wheat Ear, the performance ranges from the execution with 2 processes (the maximal number of cores on one Cloud node) to 20 processes (the maximal number of the cores on one HPC node). The results of the two applications are calculated as the mean values of three executions, and the numbers are proved to be stable (as indicated by the variances in the table). The time spent in job scheduling from Cloud to HPC is negligible compared to the total execution time. The performance improvement of both applications is significant when running on an HPC node. In Figure 3(a), the application exibits optimal performance with 9 processes, whearas detoriating with 12 processes. Similarly, for Pilot Wheat Ear in Figure 3(b) 18 processes outperforms 20 processes. The reasons could be due to the fact that the volume of the dataset does not require too many processes running simultaneously, and the process synchronization cost rises with the increment of process number. It is out of the scope of this chapter to discuss the algorithms of process/thread parallelism.



(a) *Pilot Soybean Farming* *(b) Pilot Wheat Ear*

Figure 3. Time comparison for Pilot Soybean Farming (data preprocessing procedure) and Pilot Wheat Ear.

Table 4. Execution time on the Cloud (VM) and HPC (bare-metal) cluster for data preprocessing procedure on a single node. (unit: minutes). The numbers in the brackets are the mathematical variances.

Cores (on 1 node)	1 core	3 cores	6 cores	9 cores	12 cores
Cloud (VM)	97.60 (0.23)	-	-	-	-
HPC (bare-metel)	64.73 (0.16)	30.07 (0.01)	24.41 (0.01)	17.81 (0)	19.52 (0)

Table 5. Execution time on the Cloud (VM) and HPC (bare-metal) cluster for Pilot Wheat Ear on a single node. (unit: minutes). The numbers in the brackets are the mathematical variances.

Cores (on 1 node)	2 core	4 cores	8 cores	16 cores	18 cores	20 cores
Cloud (VM)	1337.13 (0.00)	-	-	-		-
HPC (bare-metel)	480.50 (4.97)	282.39 (0.28)	179.54 (1.02)	144.76 (0.12)	124.25 (0.06)	134.99 (0.55)

13.6 Discussion

The immense gain in performance, which comes out-of-the-box, is one major reason for leveraging HPC to AI applications, while containerization provides compatibility and portability to ease deployment of AI applications onto HPC systems. The use cases presented in this chapter illustrates how AI applications can be adapted to utilize the conventional HPC parallelization models (such as Open MPI), and how big-data or DL frameworks, such as Apache Spark or PyTorch can be adapted in order to fit the predominant HPC scheduling environments. The demands for computer resources from the two AI use cases are relatively low, however, the merits that the HPC infrastructures can bring are clearly demonstrated. The benefits will become more evident when the AI systems become operational and when a large number of farmers, researchers and members of extension services start adopting them in a large scale. The hybrid architecture that bridges Cloud and HPC systems demonstrated herein will be ready to serve this purpose.

The hybrid architecture elaborated in this chapter contains a login node that connects both the Cloud and HPC installations, where the login node is located in two network domains: the domains of HPC cluster and Cloud cluster. On an HPC production system, a more portable and secure approach would be restraining the login node within the Cloud network domain and connecting it remotely to the HPC cluster via `ssh`, although this could cause performance degradation in case of a large volume of data transmission. Lustre [50], a distributed file system that can provide fast data access, can be utilized as the intermediate storage for Cloud and HPC cluster instead.

13.7 Conclusion Remarks and Future Works

This chapter has exhibited the necessity of implementing AI technologies in the domain of

agriculture. It described the challenges that AI applications are facing. Containerization, which may have paved the way for AI applications on Cloud clusters, may also be applied to HPC clusters to ensure compatibility and portability. This chapter has presented a novel hybrid architecture and described necessary methods of parallelizing and deploying AI applications on HPC systems. This CYBELE hybrid architecture unifies the interface for job submission on Cloud and HPC systems. Via the same user interface, long-running service programs can be scheduled to Cloud clusters, while compute-intensive and/or data-intensive applications can be scheduled onto HPC clusters where their performance can be significantly improved. Two relevant AI applications are showcased to demonstrate the efficiency of incorporating HPC and AI technologies into the field of agriculture.

Continuous increase of data volume pushes the usage of Cloud for AI applications as Cloud supplies users with on-demand computer resources. Traditional HPC centers are evolving to support AI applications, not only in terms of powerful hardware resources, such as many GPU nodes, large-size memory and SSD, but also with regard to the provision software environments, such Singularity, Shifter [51] or Charliecloud [52]. Meanwhile, AI applications are being adapted to HPC system environment, e.g., the DL Framework Horovod [53] , which adopts MPI concept, allows to perform efficiently distributed training in HPC.

Acknowledgements

CYBELE project has received funding from the European Union's Horizon 2020 research and innovation programme under grant agreement NO.825355.

Yield monitor data from Pilot Soybean Farming has been generously granted by Donau Soya association from farms across the Lower Austria region, through CYBELE project and are

subject to NDA.

The authors would like to express the gratitude to Ms. Aikaterini Papapostolou and Dr. Sophia Karagiorgou for proof-reading the contents. Many thanks to Dr. Yiannis Georgiou for the support of this work. The authors also appreciate datasets support for Pilot Wheat Ear from Željana Grbović and Marko Panić.

References

[1]W. Felter, A. Ferreira, R. Rajamony, and J. Rubio, "An updated performance comparison of virtual machines and Linux containers," in *2015 IEEE International Symposium on Performance Analysis of Systems and Software (ISPASS)*, 2015, pp. 171–172.

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

[3]J. P. Martin, A. Kandasamy, and K. Chandrasekaran, "Exploring the Support for High Performance Applications in the Container Runtime Environment," *Hum.-Centric Comput. Inf. Sci.*, vol. 8, no. 1, 2018, doi: 10.1186/s13673-017-0124-3.

[4]M. Plauth, L. Feinbube, and A. Polze, "A Performance Survey of Lightweight Virtualization Techniques," in *Service-Oriented and Cloud Computing*, 2017, pp. 34–48.

[5]J. Zhang, X. Lu, and D. K. Panda, "Is Singularity-Based Container Technology Ready for Running MPI Applications on HPC Clouds?," in *Proceedings of The10th International Conference on Utility and Cloud Computing*, 2017.

[6]G. Hu, Y. Zhang, and W. Chen, "Exploring the Performance of Singularity for High Performance Computing Scenarios," in *2019 IEEE 21st International Conference on High Performance Computing and Communications; IEEE 17th International Conference on Smart City; IEEE 5th International Conference on Data Science and*

Systems (HPCC/SmartCity/DSS), 2019, pp. 2587–2593.

[7]A. J. Younge, K. Pedretti, R. E. Grant, and R. Brightwell, "A Tale of Two Systems: Using Containers to Deploy HPC Applications on Supercomputers and Clouds," in *2017 IEEE International Conference on Cloud Computing Technology and Science (CloudCom)*, 2017, pp. 74–81. [Online]. Available: https://doi.org/10.1109/CloudCom.2017.40

[8]L. Abdollahi Vayghan, M. A. Saied, M. Toeroe, and F. Khendek, "Deploying Microservice Based Applications with Kubernetes: Experiments and Lessons Learned," in *2018 IEEE 11th International Conference on Cloud Computing (CLOUD)*, 2018, pp. 970–973.

[9]K. Hightower, B. Burns, and J. Beda, *Kubernetes: Up and Running Dive into the Future of Infrastructure,* 1st ed. Sebastopol, California, US: O'Reilly Media, Inc, 2017.

[10] G. Staples, "TORQUE Resource Manager," in *Proceedings of the 2006 ACM/IEEE Conference on Supercomputing*, 2006, p. 8.

[11] Morris A. Jette, Andy B. Yoo, and Mark Grondona, "SLURM: Simple Linux Utility for Resource Management," in *In Lecture Notes in Computer Science: Proceedings of Job Scheduling Strategies for Parallel Processing (JSSPP) 2003*, 2002, pp. 44–60.

[12] Vijai Singh, Namita Sharma, and Shikha Singh, "A review of imaging techniques for plant disease detection," *Artificial Intelligence in Agriculture*, vol. 4, pp. 229–242, 2020, doi: 10.1016/j.aiia.2020.10.002.

[13] Tanha Talaviya, Dhara Shah, Nivedita Patel, Hiteshri Yagnik, and Manan Shah, "Implementation of artificial intelligence in agriculture for optimisation of irrigation and application of pesticides and herbicides," *Artificial Intelligence in Agriculture*, vol. 4, pp. 58–73, 2020, doi: 10.1016/j.aiia.2020.04.002.

[14] D. Budaev *et al.,* "Conceptual design of smart farming solution for precise agriculture," *International Journal of Design & Nature and Ecodynamics*, vol. 13, pp. 307–314, 2018,

doi: 10.2495/DNE-V13-N3-307-314.

[15] O. G. Selfridge, “Pattern Recognition and Modern Computers,” in *Proceedings of the March 1-3, 1955, Western Joint Computer Conference*, 1955, pp. 91–93. [Online]. Available: https://doi.org/10.1145/1455292.1455310

[16] T. Ben-Nun and T. Hoefler, “Demystifying Parallel and Distributed Deep Learning: An In-Depth Concurrency Analysis,” *ACM Comput. Surv.*, vol. 52, no. 4, 2019, doi: 10.1145/3320060.

[17] Z. Jia, M. Zaharia, and A. Aiken, “Beyond Data and Model Parallelism for Deep Neural Networks,” in *Proceedings of Machine Learning and Systems*, 2019, pp. 1–13. [Online]. Available: https://proceedings.mlsys.org/paper/2019/file/c74d97b01eae257e44aa9d5bade97baf-Paper.pdf

[18] R. E. Uhrig, “Introduction to artificial neural networks,” in *Proceedings of IECON ’95 - 21st Annual Conference on IEEE Industrial Electronics*, 1995, 33-37 vol.1.

[19] M.\’ı. Abadi *et al.,* “TensorFlow: A System for Large-scale Machine Learning,” in *Proceedings of the 12th USENIX Conference on Operating Systems Design and Implementation*, 2016, pp. 265–283. [Online]. Available: http://dl.acm.org/citation.cfm?id=3026877.3026899

[20] Adam Paszke *et al.,* “PyTorch: An Imperative Style, High-Performance Deep Learning Library,” in *Advances in Neural Information Processing Systems 32: Annual Conference on Neural Information Processing Systems 2019, NeurIPS 2019, 8-14 December 2019, Vancouver, BC, Canada*, 2019, pp. 8024–8035.

[21] M. Zaharia *et al.,* “Apache Spark: A Unified Engine for Big Data Processing,” *Commun. ACM*, vol. 59, no. 11, pp. 56–65, 2016, doi: 10.1145/2934664.

[22] S. Pandey and V. Tokekar, “Prominence of MapReduce in Big Data Processing,” in *2014 Fourth International Conference on Communication Systems and Network*

Technologies, 2014, pp. 555–560. [Online]. Available: https://doi.org/10.1109/CSNT.2014.117

[23] N. Narkhede, G. Shapira, and T. Palino, *Kafka: The Definitive Guide Real-Time Data and Stream Processing at Scale,* 1st ed. Sebastopol, California, US: O'Reilly Media, Inc, 2017.

[24] G. Sammons, *Exploring Ansible 2: Fast and Easy Guide*. North Charleston, SC, USA: CreateSpace Independent Publishing Platform, 2016.

[25] G. Mateescu, W. Gentzsch, and C. J. Ribbens, "Hybrid Computing-Where HPC Meets Grid and Cloud Computing," *Future Gener. Comput. Syst.*, vol. 27, no. 5, pp. 440–453, 2011, doi: 10.1016/j.future.2010.11.003.

[26] R. Mayer and H.-A. Jacobsen, "Scalable Deep Learning on Distributed Infrastructures: Challenges, Techniques, and Tools," *ACM Comput. Surv.*, vol. 53, no. 1, 2020, doi: 10.1145/3363554.

[27] D. Brayford, S. Vallecorsa, A. Atanasov, F. Baruffa, and W. Riviera, "Deploying AI Frameworks on Secure HPC Systems with Containers," in *2019 IEEE High Performance Extreme Computing Conference (HPEC)*, 2019, pp. 1–6.

[28] W. Gropp, E. Lusk, and A. Skjellum, *Using MPI: Portable Parallel Programming with the Message-passing Interface*. Cambridge, MA, USA: MIT Press, 1994.

[29] Message Passing Interface Forum and University of Tennessee, Knoxville,Tennesse, *MPI: A Message-Passing Interface Standard.* [Online]. Available: https://www.mpi-forum.org/docs/mpi-3.1/mpi31-report.pdf

[30] N. Zhou *et al.,* "Container Orchestration on HPC Systems through Kubernetes," *Journal of Cloud Computing: Advances, Systems and Applications*, 2021, doi: 10.1186/s13677-021-00231-z.

[31] Naweiluo Zhou, Yiannis Georgiou, Li Zhong, Huan Zhou, and Marcin Pospieszny,

"Container Orchestration on HPC Systems," in *2020 IEEE International Conference on Cloud Computing (CLOUD)*, 2020.

[32] Y. Georgiou *et al.,* "Converging HPC, Big Data and Cloud Technologies for Precision Agriculture Data Analytics on Supercomputers," in *High Performance Computing*, 2020, pp. 368–379. [Online]. Available: https://doi.org/10.1007/978-3-030-59851-8_25

[33] Naweiluo Zhou, "Containerization and Orchestration on HPC Systems," in *Sustained Simulation Performance 2019 and 2020*, 2021.

[34] Vadim Pisaruk and Sasha Yakovtseva, *WLM-operator. Gitlab: https://github.com/sylabs/wlm-operator (accessed on 13/02/2020)*.

[35] Gregory M. Kurtzer, Vanessa V. Sochat, and Michael Bauer, "Singularity: Scientific containers for mobility of compute," in *PloS one*, 2017. [Online]. Available: https://doi.org/10.1371/journal.pone.0177459

[36] D. C. Cireşan, U. Meier, L. M. Gambardella, and J. Schmidhuber, "Deep, Big, Simple Neural Nets for Handwritten Digit Recognition," *Neural Computation*, vol. 22, no. 12, pp. 3207–3220, 2010, doi: 10.1162/NECO_a_00052.

[37] Alon Halevy, Peter Norvig, and Fernando Pereira, "The Unreasonable Effectiveness of Data," *IEEE Intelligent Systems*, vol. 24, pp. 8–12, 2009. [Online]. Available: http://www.computer.org/portal/cms_docs_intelligent/intelligent/homepage/2009/x2exp.pdf

[38] M. Drusch *et al.,* "Sentinel-2: ESA's Optical High-Resolution Mission for GMES Operational Services," *Remote Sensing of Environment*, vol. 120, pp. 25–36, 2012, doi: 10.1016/j.rse.2011.11.026.

[39] X. Yan and X. G. Su, *Linear Regression Analysis: Theory and Computing*. USA: World Scientific Publishing Co., Inc, 2009.

[40] J. R. Quinlan, "Induction of Decision Trees," *Mach. Learn.*, vol. 1, no. 1, pp. 81–106, 1986, doi: 10.1023/A:1022643204877.

[41] Jerome H. Friedman, “Greedy function approximation: A gradient boosting machine,” *The Annals of Statistics*, vol. 29, no. 5, pp. 1189–1232, 2001, doi: 10.1214/aos/1013203451.

[42] L. Breiman, “Random Forests,” *Mach. Learn.*, vol. 45, no. 1, pp. 5–32, 2001, doi: 10.1023/A:1010933404324.

[43] V. K. Vavilapalli *et al.,* “Apache Hadoop YARN: Yet Another Resource Negotiator,” in *Proceedings of the 4th Annual Symposium on Cloud Computing*, 2013.

[44] J. Dean and S. Ghemawat, “MapReduce: Simplified Data Processing on Large Clusters,” *Commun. ACM*, vol. 51, no. 1, pp. 107–113, 2008, doi: 10.1145/1327452.1327492.

[45] B. Hindman *et al.,* “Mesos: A Platform for Fine-Grained Resource Sharing in the Data Center,” in *Proceedings of the 8th USENIX Conference on Networked Systems Design and Implementation*, 2011, pp. 295–308.

[46] Zeljana Grbovic, Marko Panic, Oskar Marko, Sanja Brdar, and Vladimir S. Crnojevic, “Wheat Ear Detection in RGB : and Thermal Images Using Deep Neural Networks,” in *Machine Learning and Data Mining in Pattern Recognition, 15th International Conference on Machine Learning and Data Mining, MLDM 2019, New York, NY, USA, July 20-25, 2019, Proceedings, Volume II*, 2019, pp. 875–889.

[47] O. Ronneberger, P.Fischer, and T. Brox, “U-Net: Convolutional Networks for Biomedical Image Segmentation,” in *Medical Image Computing and Computer-Assisted Intervention (MICCAI)*, 2015, pp. 234–241. [Online]. Available: http://lmb.informatik.uni-freiburg.de/Publications/2015/RFB15a

[48] Jeremy Howard and Sylvain Gugger, “Fastai: A Layered API for Deep Learning,” *Information*, vol. 11, no. 2, p. 108, 2020, doi: 10.3390/info11020108.

[49] D. Merkel, “Docker: Lightweight Linux Containers for Consistent Development and Deployment,” *Linux J*, vol. 2014, no. 239, pp. 76–90, 2014.

[50] R. Salunkhe, A. D. Kadam, N. Jayakumar, and S. Joshi, “Luster a scalable architecture file system: A research implementation on active storage array framework with Luster file system,” in *2016 International Conference on Electrical, Electronics, and Optimization Techniques (ICEEOT)*, 2016, pp. 1073–1081.

[51] Lisa Gerhardt *et al.,* “Shifter: Containers for HPC,” *Journal of Physics: Conference Series*, vol. 898, p. 82021, 2017, doi: 10.1088/1742-6596/898/8/082021.

[52] R. Priedhorsky and T. Randles, “Charliecloud: Unprivileged Containers for User-Defined Software Stacks in HPC,” in *Proceedings of the International Conference for High Performance Computing, Networking, Storage and Analysis*, 2017.

[53] Alexander Sergeev and Mike Del Balso, “Horovod: fast and easy distributed deep learning in TensorFlow,” *CoRR*, abs/1802.05799, 2018.