이학박사 학위논문

A study on the earth science data generation by the numerical modeling and machine learning on the cloud computing

클라우드 컴퓨팅 환경기반에서 수치 모델링과 머신러닝을 통한 지구과학 자료생성에 관한 연구

2022 년 8 월

서울대학교 대학원

지구환경과학부

정 광 욱

A study on the earth science data generation by the numerical modeling and machine learning on the cloud computing

지도 교수 조 양 기

이 논문을 이학박사 학위논문으로 제출함
2022 년 7 월

서울대학교 대학원
지구환경과학부
정 광 욱

정광욱의 이학박사 학위논문을 인준함
2022 년 7 월

Abstract

To investigate changes and phenomena on Earth, many scientists use high-resolution-model results based on numerical models or develop and utilize machine learning-based prediction models with observed data. As information technology advances, there is a need for a practical methodology for generating local and global high-resolution numerical modeling and machine learning-based earth science data.

This study recommends data generation and processing using high-resolution numerical models of earth science and machine learning-based prediction models in a cloud environment.

To verify the reproducibility and portability of high-resolution numerical ocean model implementation on cloud computing, I simulated and analyzed the performance of a numerical ocean model at various resolutions in the model domain, including the Northwest Pacific Ocean, the East Sea, and the Yellow Sea. With the containerization method, it was possible to respond to changes in various infrastructure environments and achieve computational reproducibility effectively.

The data augmentation of subsurface temperature data was performed using generative models to prepare large datasets for model training to predict the vertical temperature distribution in the ocean. To train the prediction model, data augmentation was performed using a generative model for observed data that is relatively insufficient compared to satellite dataset.

In addition to observation data, HYCOM datasets were used for performance comparison, and the data distribution of augmented data was similar to the input data distribution. The ensemble method, which combines stand-alone predictive models, improved the performance of the predictive model compared to that of the model based on the existing observed data. Large amounts of computational resources were required for data synthesis, and the synthesis was performed in a cloud-based graphics processing unit environment.

High-resolution numerical ocean model simulation, predictive model development, and the data generation method can improve predictive capabilities in the field of ocean science. The numerical modeling and generative models based on cloud computing used in this study can be broadly applied to various fields of earth science.

Keywords: Cloud computing, Numerical Ocean model, Machine learning, Containerization, Generative models, Vertical temperature distribution

Student Number: 2012-30096

Table of Contents

List of Figures

List of Tables

1. General Introduction

With the advancement of information technology (IT), numerical models based on the equation of motion have been used to predict future phenomena or develop and utilize statistical prediction models from various observation data (Than, 2017). Machine learning techniques have also been used to analyze and predict the changes in various phenomena on Earth (Hu, 2021).

Owing to the rapid development of IT, researchers have gradually extended the limits of specific−scale numerical model domains and regional analysis to global or large−scale model domains and performed improved modeling with high resolutions (Tintó et al., 2017). In addition, machine learning techniques have been used to analyze large amounts of data and establish predictive models (Ahmad et al., 2019). The cloud stores rapidly increasing observation data (NOAA, 2022).

Along with the expansion of numerical modeling experiments, results of high−resolution and precise numerical models are increasingly critical in predicting and analyzing various physical changes in the ocean (Sommer et al., 2018).

After the initial conditions are set in the main equations of motion in the ocean numerical model, various physical variables are calculated using a divided grid and time intervals. Increasing the resolution of the numerical model grid for precise numerical simulations increases the number of grids to be calculated. Sufficient computational resources are required to perform the increased computation level within a specific period, and more precise analysis is possible based on the computational resources. Extensive computing resources are required to perform such precise numerical modeling or global ocean model (Hamilton et al., 2008). The computing resource is a significant limitation in improving the resolution of numerical models.

High-performance computing (HPC) clusters are widely used to perform large-scale numerical models of oceans with a large amount of computation (Yang et al., 2018), and they are processing devices with expensive equipment (Bozzo-Rey et al., 2006). Recently, with the development of virtualization-based cloud technology, computing resources can be easily provided through a network.

Researchers of various numerical models have attempted to establish earth science numerical models in a cloud environment (Chen et al., 2017; Zhuang et al., 2020; Jung et al., 2021). Problems related to performance and the environment in the early days of cloud computing have been gradually resolved.

Recently, research cases applied to computationally intensive modeling that require significant resources have been introduced (Zhuang et al., 2020). The applicability of the earth science model to the cloud environment has been verified in many studies. Beyond verification of applicability, performance analysis to improve the resolution of the earth science numerical models or advanced research to improve and develop the earth science numerical models using cloud–native technology are required.

This study presents performance analysis and the environment configuration method for performing high–resolution numerical modeling using a virtual machine provided as a cloud computing resource and a method for achieving the calculation reproducibility of the numerical model using container technology. The reproducibility of research results is the basis for scientific research and sharing of the research process.

To achieve the computational reproducibility of numerical models, many researchers have attempted to configure the environment of the numerical model and the various environmental variables necessary for model execution (Markus et al., 2019; Grüning et al., 2019). This work can be regarded as essential but challenging in numerical modeling. Previous studies have recommended documenting and sharing various variables and developing a numerical model based on the shared data (Nüst et al., 2017).

The aim of this study is to design and apply a container−based numerical model execution architecture, which is a cloud−based technology, to resolve such difficulties and realize the computational reproducibility of numerical ocean models.

It is common to establish a prediction model based on observed data or analyze and predict by performing a statistical model to predict and analyze various phenomena on Earth along with numerical models. In addition to high−resolution numerical models, machine learning is an essential method for generating and analyzing data in earth science. Recently, in the ocean field, many researchers have analyzed and predicted various ocean phenomena using machine learning methods (Ahmad et al., 2019). Owing to the nature of ocean observation data, the location and time of observation are limited, and the amount of the observed data is also limited (Levin et al., 2019; Weller et al., 2019). With the development of various observation techniques, periodically observed and stored data such as satellite data are abundant, but the observed data is mainly concentrated in the surface layer. However, measured data are limited because observation data for areas where remote observation is difficult, such as the deep ocean, are constrained by location and time (Klemas et al., 2013; Levin et al., 2019). If observational data is supplemented in the ocean field, improved prediction and analysis capabilities can be achieved (Bolton et al., 2019). This study proposes an architecture that can generate observable data based on the observation data by applying the generative model technique of a neural network to solve the problem of insufficient ocean observation data. A comparison between the observed and synthetic data was performed to verify the generated data, and the synthetic dataset was used to train the prediction model. The performance of the predictive model can be improved using large synthetic datasets. This study proposes a valuable method for generating training datasets for machine learning and resolving missing data problems in earth science.

2. Performance of numerical ocean modeling on cloud computing[①]

2.1. Introduction

Numerical models are widely used to predict and analyze ocean circulation and various physical property changes. Large amounts of computational power are required for numerical experiments to simulate realistic global ocean circulation. However, preparing sufficient computer resources is difficult owing to economic and physical constraints. Even when the Information Technology (IT) infrastructure is sufficient, installing and preparing the ocean model setup is time−consuming. If IT infrastructures were free from setup and maintenance, ocean numerical models could be more easily and widely used. Efficient configuration and utilization of IT resources is being increasingly demanded in many fields, including ocean modeling. In order to satisfy this demand, many companies and organizations are considering utilizing public cloud computing services such as Amazon Web Services (AWS) and Google Cloud Computing (GCP). The number of applications that can be executed on cloud systems

[①] The results of the presented work have been submitted into 『바다』, (2022).

has been steadily increasing, and numerous studies are being conducted to determine whether and how applications and operations can be ported to cloud computing environments without performance penalty or technical issues. In the early days of commercial cloud services, many experiments associated with the operation of climate models in cloud computing environments were conducted. For example, Oesterle et al. (2015) compared the performance, disadvantages, and merits of cloud computing and grids for meteorological model applications. Montes et al. (2017) ported and tested AWS as an infrastructure for the Berkeley Open Infrastructure for Network Computing (BOINC) system. Chen et al. (2017) simulated the Community Earth System Model (CEMS) on AWS.

Cloud computing is a computing resource utilization method in which IT infrastructure resources are provided through N/W, with fees paid according to computing amount and time of usage. It allows researchers, research institutes, and numerical ocean model scientists with limited infrastructure resources (i.e., servers, storage, and electricity) to use numerical ocean models at optimal cost without physical difficulties. Many three-dimensional numerical ocean models are executed in High Performance Computing (HPC) environments using manycore and Software (S/W) systems such as Message Passing Interface (MPI). In order to execute large-scale numerical models in parallel, systems S/W such as MPI should be implemented properly with the configuration of high-speed Network (N/W) devices, such as InfiniBand, for communication among servers. Expensive Hardware (H/W) and N/W are usually managed by IT professional organizations and engineers. Various studies have been conducted on parallel processing using cloud computing to overcome the problem of high-cost IT infrastructure. However, the cloud environment has still been found to have several limitations for parallel processing owing to insufficient functionalities (Oesterle et al., 2015). Recently, AWS, GCP, and Azure, which are public cloud computing services, have begun to provide various technological bases such as high-performance instances, ethernet-based high performance N/W, and remote direct memory access (RDMA) for effective implementation of HPC. They enable users to easily prepare numerical model environments and conduct numerical experiments anytime and anywhere.

This study was conducted to analyze the performance of an ocean model on commercial clouds and ascertain how to effectively construct and execute large-scale three-dimensional numerical ocean models in commercial cloud computing environments using ethernet-based high-performance N/W, high-performance memory, and CPU. An additional goal was to also provide a method to improve or extend the performance of such systems in cloud computing environments with real case study data. For this study, the Regional Ocean Modeling System (ROMS), which is a typical community ocean model, was executed on commercial clouds. The various performance results and comparison analysis of performance data according to computing resource types are presented. Prior to this study, I investigated the feasibility of ROMS for the cloud computing environment, and also compared the performance of ROMS in a virtualization-based commercial cloud with that in a non-virtualization-based HPC cluster (Jung et al., 2017).

2.2. Cloud Computing

2.2.1. Cloud computing overview

Cloud computing provides virtualized and configurable computer resources (e.g., networks, servers, storage, applications, and services) in computing resource pools with functions such as self-service provision, automatic usage metering, and rapid provisioning. Users can access these resources through broadband networks (such as the internet). Various kinds of services are provided and classified according to the associated resources. They include Infrastructure as a Service (IaaS), Platform as a Service (PaaS), and Software as a Service (SaaS) (Figure 2.1).

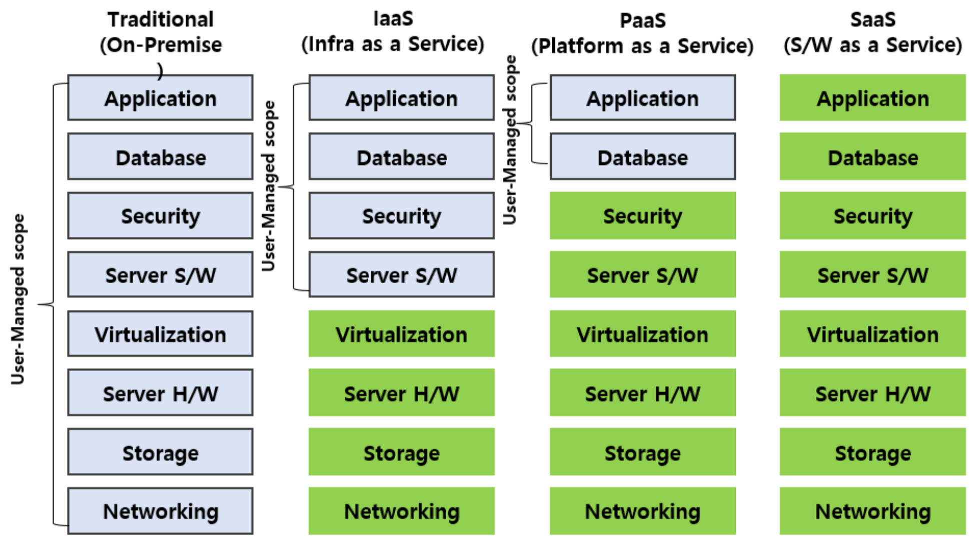


Figure 2.1 Conceptual diagram of cloud service types.

Depending on the deployment model, the cloud platform itself can be may also be categorized as public or private (Mell and Grance, 2011). IT companies such as Amazon, Microsoft, Google, and IBM provide public cloud commercial services: AWS, Azure, GCP, and Bluemix, respectively (Gartner, 2018). A private cloud is constructed by an organization for internal users and purposes. In this study, I used available public cloud services that can be used with the IaaS option for running a numerical ocean model. Virtualization is a key technology required to provide services such as IaaS. Through virtualization, physical servers, storage, and N/W resources can be logically segmented and allocated to users, and logically returned when jobs are completed.

Figure 2.2 shows a hypervisor, a server virtualization technology that can logically divide server resources.

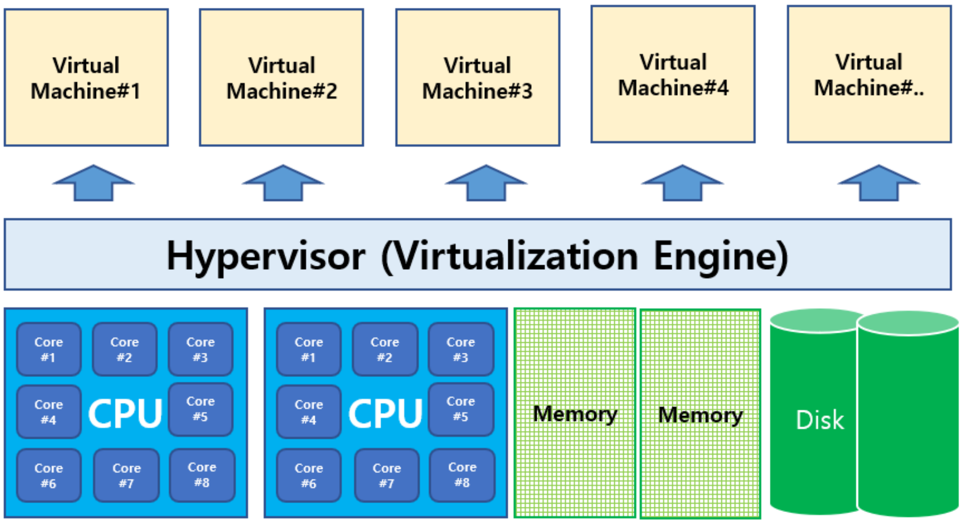


Figure 2.2 Conceptual diagram of an idealized hypervisor. Physical resources such as CPU, memory, and disk are virtualized through virtualization S/W (e.g., hypervisor) and can be logically allocated as instances.

A physical x-86 server can be logically separated and assigned as a virtual machine (VM) through such a hypervisor. The virtual servers in public cloud computing are examples of the utilization of these hypervisor technologies. The virtualized servers used in the commercial cloud are also optimized VMs through virtualization technology (e.g., KVM) (AWS, 2020). Because VMs can be created using predefined templates in a repository, it is possible to rapidly recreate numerous VMs with the same configurations simply by copying a VM template (AWS, 2022b). These techniques provide a useful method for easily preparing multiple nodes for large−scale numerical model experiments. This is also useful for researchers who need to setup highly complicated environments for numerical modeling.

2.2.2. Commercial cloud computing services

The number of users of public cloud services has increased rapidly for economic or technical reasons. Major commercial public cloud services in the global market such as Amazon's AWS, Microsoft's Azure, IBM's Bluemix, and Google's compute cloud service have numerous datacenters and provide many services in various countries (Gartner, 2018). Commercial cloud vendors provide PaaS and SaaS, as well as server resources, according to the user's purpose. In addition, the number of earth science organizations (such as NASA) that use commercial cloud to store and process earth−related information is increasing (Chen et al., 2017). In this study, I constructed and executed an environment for the ocean numerical model in commercial clouds using VM servers with high−speed N/W and memory to analyze the performance of various cluster configurations, inter−server communication, and I/O.

Table 2.1 represents an example of the various server resources provided (as of March 2018) by commercial clouds such as AWS (AWS, 2022c).

Table 2.1 Overview of purpose, specifications, and price of AWS instance types (us−west−2, Oregon)

Type	Purpose	Sub-Type	vCPU	Memory (GB)	N/W	Price/h ($)
T	General purpose	t3.large	2	8.0	Up to 5G	0.0832
		t3.xlarge	4	16.0	Up to 5G	0.1664
		t3.2xlarge	8	32.0	Up to 5G	0.3328
M	General purpose	m5.4xlarge	16	64.0	Up to 10G	0.768
		m5.12xlarge	48	192	10G	2.304
		m5.24xlarge	96	384.0	25G	4.608
C	Compute-optimized	c4.8xlarge	36	60.0	10G	1.591
		c5.9xlarge	36	72.0	10G	1.53
		c5.18xlarge	72	144.0	25G	3.06
R	Memory-intensive applications	r4.16xlarge	64	488.0	25G	4.256
		r5.12xlarge	48	384.0	10G	3.024
		r5.24xlarge	96	768.0	25G	6.048
P	GPU instance	p2.16xlarge	64	732	25G	14.40
		p3.8xlarge	64	488	10G	12.24
		p3.16xlarge	96	768	25G	24.48

As the performance and features vary according to server instance, it is possible to combine the required instances according to the purpose of the research. GPU−equipped instances, which are widely used for deep learning and high−speed image processing, are also available in cloud computing. Expensive IT resources can be used at a reasonable price according to the usage amount. Public cloud service’ s prices vary according to datacenter and resource type. The most economic server can be selected regardless of the distance between the user and the server. It is also possible to use IT resources at lower cost by using spot−instance type resources instead of on−demand type. Amazon Elastic Compute Cloud (EC2) spot−instances are spare compute capacity in the cloud, which can provide lower cost compared to on−demand instances (AWS, 2022d). GCP also provides pre−emptible instances at a lower price point (Google, 2022c). The datacenter and services in region Oregon, USA were selected for this study.

High−speed processors, high−bandwidth memory, and high N/W throughput are essential for large−scale modeling. In this study, I chose recent servers with 64−bit Linux for our numerical modeling experiment. High performance virtualized servers are appropriate for numerical models with MPI because commercial clouds provide them with high bandwidth over 10 Gbps and latency values are between 36 and 42 μs when the message size is less than 32 bytes (Table 2.2).

Table 2.2 Latency of cloud HPC clusters according to message size

Message size	1 byte	2 bytes	4 bytes	8 bytes	16 bytes	32 bytes
Latency (μs)	36.2	38.8	36.6	35.6	40.7	36

VM template copying for large-scale models in virtualized computing environments such as public cloud services can minimize preparation time.

2.3. Numerical model for performance analysis of commercial clouds

2.3.1. High Performance Linpack Benchmark

HPL, an implementation of Linpack Benchmarking, is a useful tool for evaluating the performance of High-Performance Computing Clusters (HPCC) (Rajan et al., 2012). It is a benchmarking software package that solves a random dense linear system in double precision (64 bit) arithmetic on distributed-memory computers such as MPI clusters. The general performance of clusters was tested using HPL and compared with the performance of ROMS. The cluster performance was evaluated as floating-point operations per second (Flops) according to the nodes and memory. The results are presented in section 2.4.

2.3.2. Benchmark Sustainable Memory Bandwidth and Memory Latency

Memory bandwidth is a crucial determinant of computing speed because numerical ocean modeling requires huge memory I/O in an MPI environment. In this study, I evaluated the bandwidth of memory in virtualized computing environments with the STREAM benchmark S/W for HPC. Memory bandwidth depends on CPU type and instance type in a virtualization environment. The memory bandwidth was measured using a single node and multiple nodes respectively in parallel with MPI.

Memory latency could also be an important factor in memory I/O performance. I measured the memory latency of the servers with Intel memory latency check tool for analysis of the memory I/O performance (Intel, 2021). I analyzed the latency and the cache hierarchy to evaluate total memory I/O performance.

2.3.3. Numerical Ocean Model

ROMS, the numerical model used in this study, is a free−surface ocean model with vertical terrain−following and horizontal curvilinear coordinates. It solves hydrostatic, free−surface primitive equations (Shchepetkin and McWilliams, 2005). A third−order upstream advection scheme and the K−Profile Parameterization scheme (Large et al., 1994) are used for horizontal advection and vertical mixing, respectively. Many ocean scientists use ROMS in a variety of ways to meet their research needs. ROMS comprises very modern and modular code written in F90/F95 using C−pre−processing to activate the various physical and numerical options. It has a generic distributed−memory interface that facilitates the use of several message passing protocols. Currently, data exchange among nodes is achieved using MPI. However, other protocols such as MPI2 and symmetrical hierarchical memory can be used without much effort. Further, the entire input and output data structure of the model is via Network Common Data Form (NetCDF) (ROMS, 2015).

The model domain used in this study extends from 115° E to 162° E and from 15° N to 52° N, which includes the Yellow Sea, East China Sea, and East/Japan Sea (Figure 2.3). It features 1/10° horizontal grid resolution and 40 vertical layers. The bottom topography data are based on the Earth Topography five−minute grid (ETOPO5) dataset of the National Geophysical Data Center (Amante and Eakins, 2009).

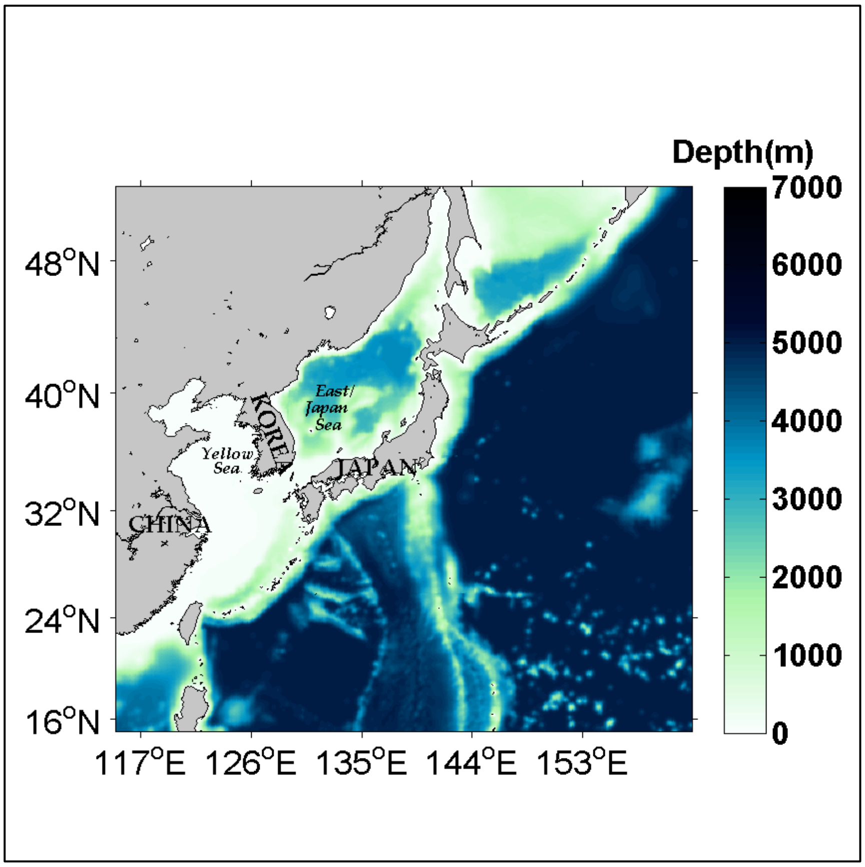


Figure 2.3 The domain of this study. The model domain covers 15–52 ° N and 115–162 ° E, which includes the East China Sea, Yellow Sea, East Sea, and the north-western part of the Pacific. Color signifies water depth.

The initial temperature and salinity were obtained from the National Ocean Data Center (NODC) World Ocean Atlas 2009 (WOA09) (Antonov et al., 2009; Locarnini et al., 2009). For the lateral open boundary, the monthly mean temperature, salinity, and velocity from the Simple Ocean Data Assimilation (SODA; Carton and Giese, 2008) for 2010 were applied. The surface forcing, which includes daily mean wind, solar radiation, air temperature, sea level pressure, precipitation, and relative humidity, was derived from the

ERA−Interim reanalysis data of the European Centre for Medium−Range Weather Forecasts for 2010 (Dee et al., 2011). These data were applied to calculate the surface heat flux with the bulk formulas (Fairall et al., 1996). Tidal forcing of 10 tidal components was provided by TPXO7 (Egbert and Erofeeva, 2002). Freshwater discharges from 12 rivers were also applied in the model (Vörösmarty et al., 1996; Wang et al., 2008). Details on the model configuration are given in Seo et al. (2014).

2.3.4. Deployment of Numerical Ocean Model and Benchmark Packages on Cloud Clusters

The numerical ocean model and benchmark S/W packages (STREAM and HPL) were setup for the performance analysis on commercial clouds using virtualized server resources. Tables 3 and 4 show details of the resource types and specifications of each cluster. I constructed five HPCs according to resource type. The performance of the benchmark S/Ws and ROMS were evaluated in each cluster with different CPUs and memory. I assigned the vCPUs up to 512 cores in each cluster, as considering the performance characteristics of hyper−threads in ROMS modeling and HPL the high−speed N/W environment configuration for MPI−based parallel processing is also absolutely necessary. The HPC environment is usually configured as an InfiniBand high-speed network capable of achieving a maximum bandwidth of 40 Gbps with very low latency. In commercial clouds, HPC can be configured as an environment supporting an Ethernet-based high-speed network having a bandwidth of up to 25 Gbps with low latency (Table 1). In order to secure a bandwidth of 10 Gbps or more and minimize latency, a separate N/W group has to be constructed and configured with Virtual Private Cloud (VPC) in the AWS commercial cloud (AWS, 2022e). Similar N/W group functions are provided by other commercial clouds. In this study, VPCs were constructed for the clusters and connected among nodes with a private Internet Protocol (IP) address. For the parallel processing of ROMS, PGI compiler (NVIDIA, 2022) and Open-MPI were configured and NetCDF installed for the input and output data structure of the model.

Virtualized IT resources are more flexible and configurable to allocate and manage than physical resources. However, there is a small decrease in performance because computing resources (e.g., CPU, Memory, NW) are provided through the software layer such as a hypervisor. The virtualized network also causes a little decrease in performance (Younge et al., 2011, Gupta et al., 2013, Jung et al, 2017). However, virtualization technology has been improved greatly. One of the technologies that have been applied to improve the speed of virtualized N/W resources is Single Root I/O Virtualization (SR−IOV). SR−IOV is a technical approach to device virtualization that provides higher I/O performance and lower CPU utilization than traditional virtualized network devices. Commercial clouds also adapt this technology to prevent performance decrease in some high−performance instances. They provide an additional high−speed N/W environment called ENA to support up to 25 Gbps bandwidth in the commercial cloud such as AWS (AWS, 2016). If the amount of communication between nodes is large or the number of nodes increases, it is possible to configure the environment using instance types providing these high−performance features and to achieve better numerical modeling performance. Using the Ethernet based on the enhanced N/W and high−performance servers, good performance can be achieved in numerical ocean modeling.

2.4. Simulation results

2.4.1. Benchmark simulation

Commercial cloud companies and private organizations provide their own instances or CPU resource types for their customers. Figure 2.4 shows the HPL performance of the clusters with various

CPU types respectively.

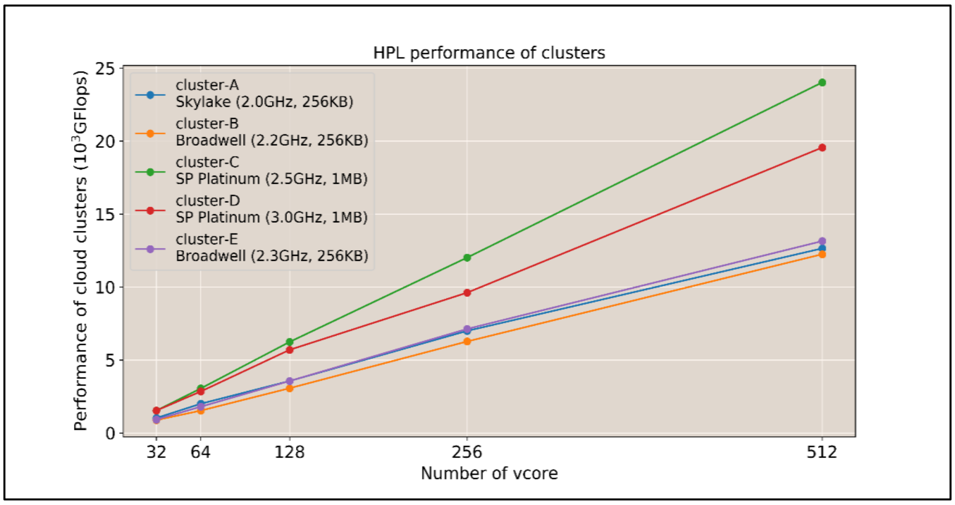


Figure 2.4. Performance of the cloud clusters according to number of vCores using the HPL S/W package.

The performance of all clusters increases linearly with the number of cores. However, the increase rate is different for each cluster. The performance of cluster C increases rapidly, whereas that of clusters A, B, and E increases slowly. It is remarkable that the performance of cluster−C is better than that of cluster−D despite the relatively low CPU clock. This result suggests that the performance of the cluster depends on the resource specification (Table 2.3).

This information helps us to select the best configuration for the running of our numerical ocean model, because the H/W efficiency according to resource type of the cluster is different.

Table 2.3 Hardware and software configuration of commercial clouds

Type	CPU Type	VCores per node	Nodes	OS	Compiler
Cluster-A	Xeon® Skylake (2.0 GHz)	64/72G	16	Linux (64Bit) (CentOS 6.9)	PGI Compiler 18.4 Open-mpi 2.01 NetCDF4 (4.4) gcc 4.4 Intel Parallel Studio XE 2019 Initial
Cluster-B	Xeon® Broadwell (2.2 GHz)	64/72G	16	Linux (64Bit) (CentOS 6.9)	PGI Compiler 18.4 Open-mpi 2.01 NetCDF4 (4.4) gcc 4.4 Intel Parallel Studio XE 2019 Initial
Cluster-C	Xeon® Skylake-SP (2.50 GHz)	96/144G	16	Linux (64Bit) (Customized)	PGI Compiler 18.4 Open-mpi 2.01 NetCDF4 (4.4) gcc 4.4 Intel Parallel Studio XE 2019 Initial
Cluster-D	Xeon® Skylake-SP (3.00 GHz)	72/144G	16	Linux (64Bit) (Customized)	PGI Compiler 18.4 Open-mpi 2.01 NetCDF4 (4.4) gcc 4.4 Intel Parallel Studio XE 2019 Initial
Cluster-E	Xeon® Broadwell E5-2686 v4 (2.3 GHz)	64/488G	16	Linux (64Bit) (Customized)	PGI Compiler 18.4 Open-mpi 2.01 NetCDF4 (4.4) gcc 4.4 Intel Parallel Studio XE 2019 Initial

The STREAM benchmark results of all clusters also increase with the number of nodes, but the bandwidth per node decreases (Figure 2. 5). This result suggests that the memory performance of the nodes decreases slightly with the increment of remote node I/O. The increase ratio of the memory bandwidth according to the core is different in each cluster. It is larger in clusters C and D, which corresponds to the performance result of the cluster, as shown in Figure 2.4.

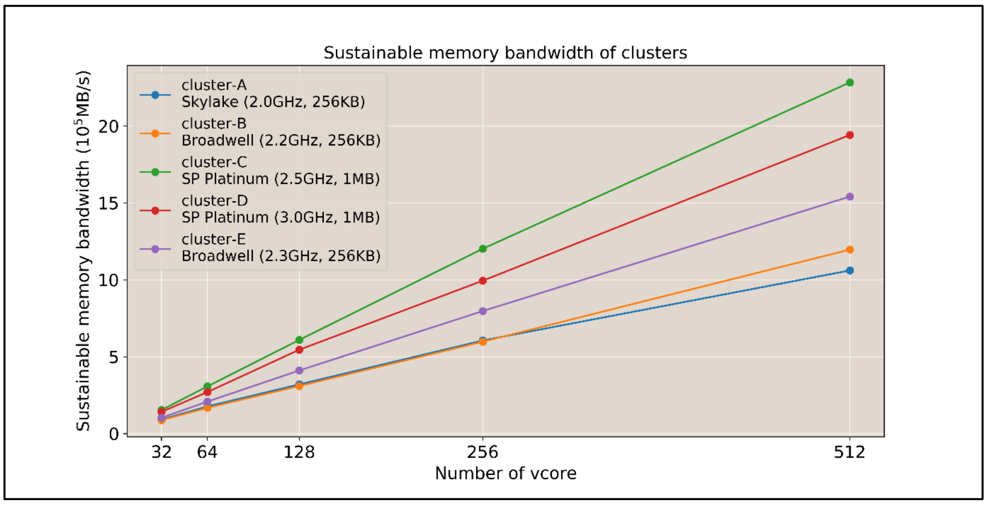


Figure 2.5 Sustainable memory bandwidth of the cloud clusters according to number of vCores.

This memory bandwidth might determine the performance of MPI-based numerical models using large memory I/O.

2.4.2. Ocean model simulation

Figures 2.6 and 2.7 show the simulated Sea Surface Temperature (SST) and surface velocity initially and after 3 days of execution from 1 January 2010, respectively. The Kuroshio Current, characterized by warm water and high speed, is well simulated along the Okinawa trough and the eastern coast of Japan. Cold water appears in the Okhotsk Sea, the northern East/Japan Sea, and the coast of the Yellow Sea as a result of the atmospheric cooling and vertical mixing (Seo et al., 2014).

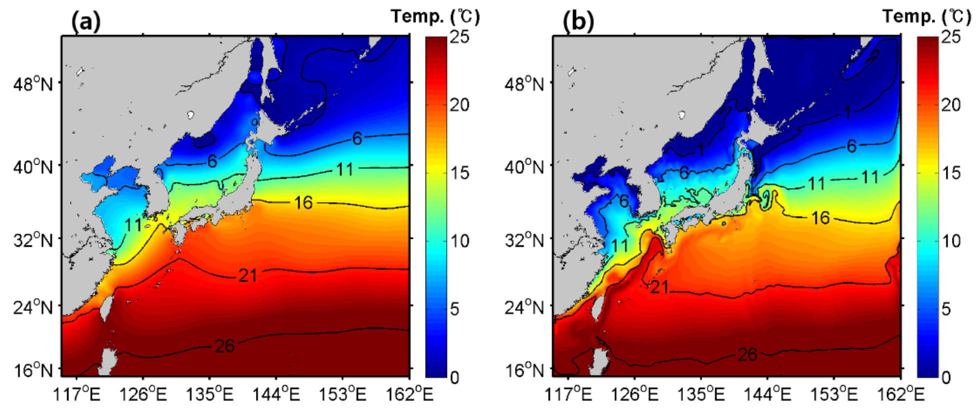


Figure 2.6 (a) Initial sea surface temperature and (b) simulated sea surface temperature after 3 days from 1 January 2010.

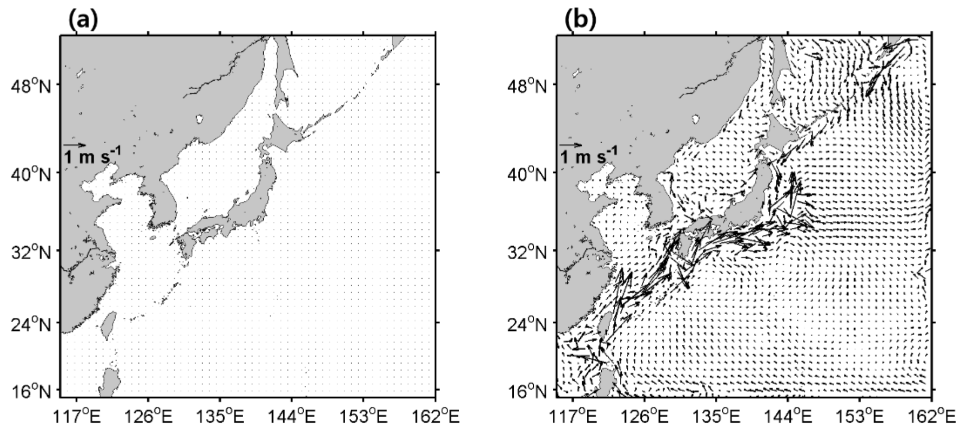


Figure 2.7 (a) Initial surface horizontal velocity and (b) simulated surface horizontal velocity after 3 days from 1 January 2010. Vector signifies current speed and direction.

Comparison of the models simulated by the various commercial servers according to cores shows that the Root−Mean−Square Error

(RMSE) of the SST is 0.0057–0.0097 ℃ and the RMSE of the u−component and v−component of the velocity is about 0.0005 ms^{-1}. This means that the difference among the simulation results from commercial cloud systems is small. I found that the RMSE between the physical servers and virtualized servers is also very small (Jung et al., 2017)

2.5. Analysis of ROMS performance on commercial clouds

The CPU and memory might be major factors that determine the execution performance in numerical modeling. Analysis of the CPU and the memory performance for ROMS and benchmark S/W were conducted in virtualized commercial clouds, where users can select the best resource type to minimize time and cost based on evaluation of CPU and memory.

2.5.1. Performance of ROMS according to H/W resources

Commercial cloud companies generally use Intel Xeon CPUs (Table 2.4). The recent Intel Xeon Scalable Processor (SP) family has more features than previous generations (formerly Broadwell microarchitecture).

Table 2.4 CPU specification of cloud HPC Clusters

Type	Cluster-A	Cluster-B	Cluster-C	Cluster-D	Cluster-E
Architecture	x86_64	x86_64	x86_64	x86_64	x86_64
CPU(s)	64	64	94	72	64
On-line CPU(s)	0–63	0–63	0–93	0-71	0-63
Thread(s) per core	2	2	2	2	2
Core(s) per socket	32	32	24	18	16
Socket(s)	1	1	2	2	2
CPU family	6	6	6	6	6
Model name	Xeon® Skylake (2.0 GHz)	Xeon® Broadwell (2.2 GHz)	Xeon® Skylake-SP (2.50 GHz)	Xeon® Skylake-SP (3.00 GHz)	Xeon® Broadwell E5-2686 (2.3 GHz)
Hypervisor vendor:	KVM	KVM	KVM	KVM	Xen
L1d cache	32 K	32 K	32 K	32 K	32 K
L1i cache	32 K	32 K	32 K	32 K	32 K
L2 cache	256 K	256 K	1024 K	1024 K	256 K
L3 cache	56320 K	56320 K	33792 K	25344 K	25 600 K

In particular, the cache structure is significantly different from the previous generation. In Broadwell and Haswell CPUs, the Mid-Level-Cache (MLC) was 256 KB per core and the Last-Level-Cache (LLC) was a shared inclusive cache with up to 2.5 MB per core. In the Intel Xeon SP family, the cache hierarchy has changed to provide a larger MLC of 1 MB per core and a smaller shared non-inclusive of up to 1.375 MB per core. A larger MLC increases the hit rate in the MLC, which results in shorter effective memory latency and lower demand on the mesh interconnect and LLC. The shift to a non-inclusive cache for the LLC allows more effective utilization of the overall cache on the chip versus an inclusive cache (Intel, 2019). In our study, cluster−C and cluster−D with the new CPU architecture showed better performance than the clusters with the old cache architecture. Cluster−C and cluster−D had a larger MLC of 1 MB per core and a smaller shared−non−inclusive LLC per core.

Table 2.5 shows the running time of ROMS according to the resource type and grid size of each cluster. The wall−clock time is greatly different according to resource type and grid size. This result signifies that resource analysis in evaluation of the execution performance is highly important.

The running time of ROMS might depend on cache capacity and hierarchy. Cluster−C and cluster−D show fast running time regardless of grid size. The larger MLCs of cluster−C and cluster−D result in high performance by reducing the latency between memory and CPU.

Table 2.5 Wall-clock time for ROMS modeling per unit node with 32 vcores

CPU(s)	32	32	48	36	32
Socket(s)	1	1	2	2	2
Vendor	Intel	Intel	Intel	Intel	Intel
Model name	Skylake (2.0 GHz)	Broadwell (2.2 GHz)	Skylake-SP (2.50 GHz)	Skylake-SP (3.00 GHz)	Broadwell E5-2686 (2.30 GHz)
L1d cache	32 K	32 K	32 K	32 K	32 K
L1i cache	32 K	32 K	32 K	32 K	32 K
L2 cache	256 K	256 K	1024 K	1024 K	256 K
L3 cache	56320 K	56320 K	33792 K	25344 K	46080 K
Wall-clock time (sec) for (51x50x20)	26	24	14	12	18
Wall-clock time (sec) for (100x98x40)	76	74	45	45	62
Wall-clock time (sec) for (210x206x40)	266	266	176	184	232
Wall-clock time (sec) for (422x412x40)	1011	983	663	698	886
Wall-clock time (sec) for (846x826x40)	4211	4294	2872	3032	4011

The execution performance of ROMS shows a similar pattern to that of HPL. The CPU with the high-performance cache memory showed better performance for HPL, as shown in Figure 2.4. Even though the CPU clock is fast, the performance of ROMS is relatively low on the cluster with long latency between the CPU and memory. The latency of the memory I/O might be long in large memory loading. The performance of ROMS in virtualized servers definitely depends on cache hierarchy, which reduces the latency between CPU and memory.

The virtualized CPU resource type related with the hyperthreads is also an important factor in parallel systems. Because many nodes with multiple cores are used to support parallel processing in large-scale numerical models, it is necessary to consider the number of servers and the best performance of the servers in parallel processing. Allocation of the optimal vCPUs for each node and optimizing the load balance of each node to achieve enhanced performance are important in HPC computing. Commercial cloud vendors usually provide virtualized servers with hyperthreads-enabled CPUs, called as a virtualized CPUs or vCPUs. Fundamentally, two threads are given based on one physical CPU core (Figure 2.8).

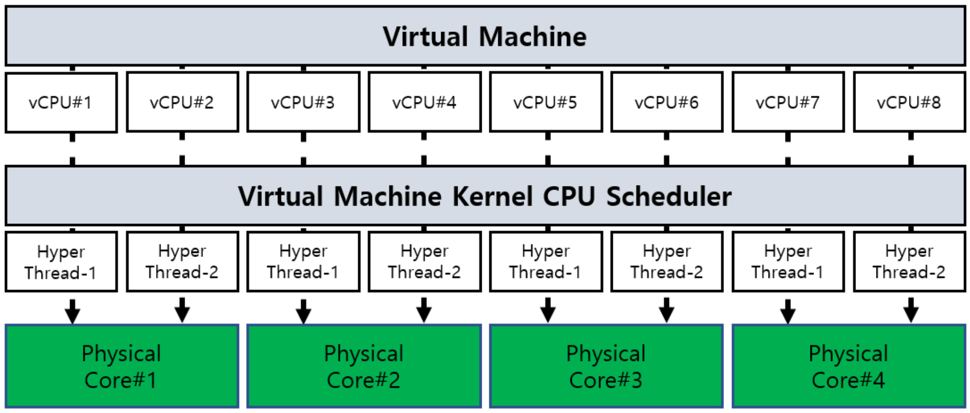
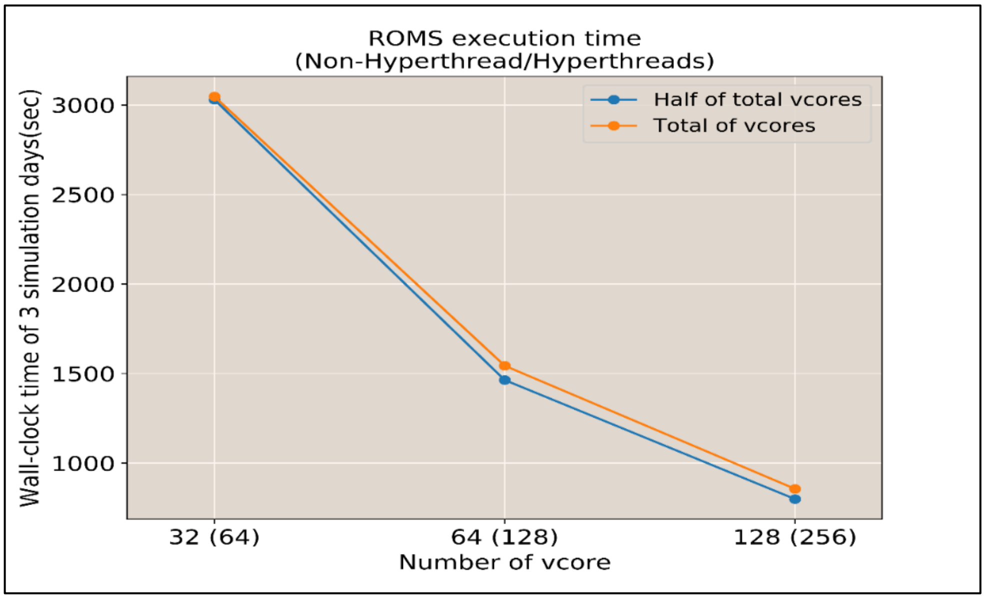


Figure 2.8. Conceptual mechanism of vCPU in the virtualization of physical quad-core. vCPUs of cloud instances are provided through the hyperthreads-enable mode. The number of physical CPUs is one-half of the vCPU instances.

A virtualized server uses one thread as a virtual core but appears to show twice the number of physical cores. Sometimes poor knowledge of these configurations can lead to misunderstanding of a vCPU's performance and consideration of it as being similar to a physical CPU's. There is little difference in the performance between running time of half of a vCPU and a full vCPU in each cluster (Figure 2.9).



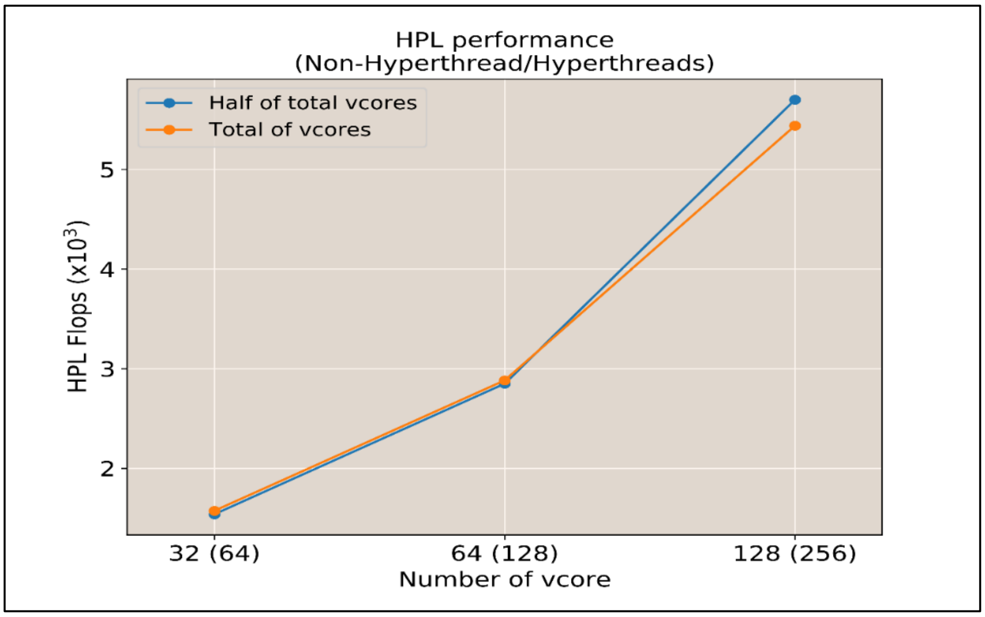


Figure 2.9. (Upper) Wall−clock running time for 3 days simulation and (lower) HPL Flops according to the number of vCores on cluster−C.

If one physical CPU is used fully, the two threads such as MPI job task eventually have to share one physical CPU core resource. In this case, there is little gain in the performance of MPI jobs in hyperthreads mode. Therefore, it is desirable to disable hyper−threading or allocate vCPU in a commercial cloud considering this characteristic.

The memory I/O performance is important in numerical models using huge memory I/O in the HPC clusters. The sustainable memory bandwidth and latency time determines the execution performance of ocean modeling with large memory I/O. Figure 2.10 shows the latency of the memory in three clusters, which provides latency information.

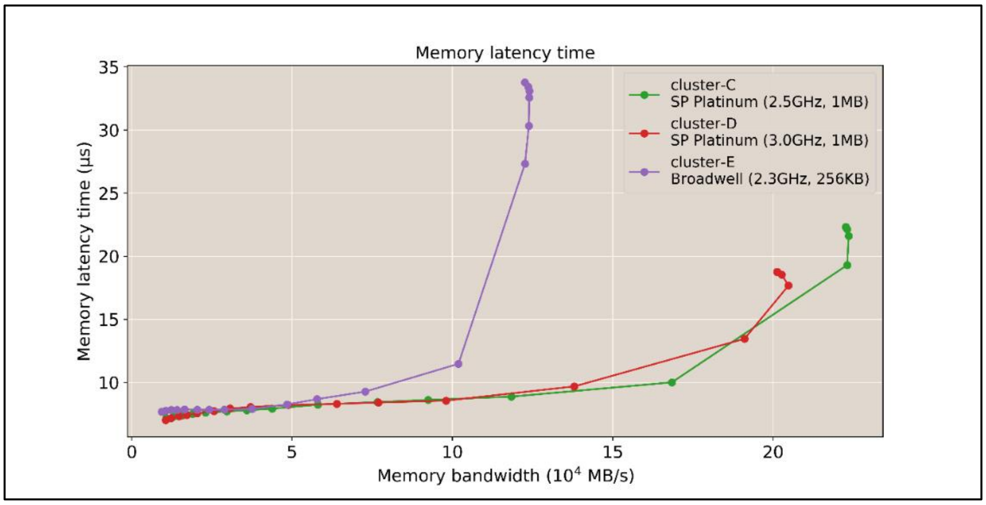
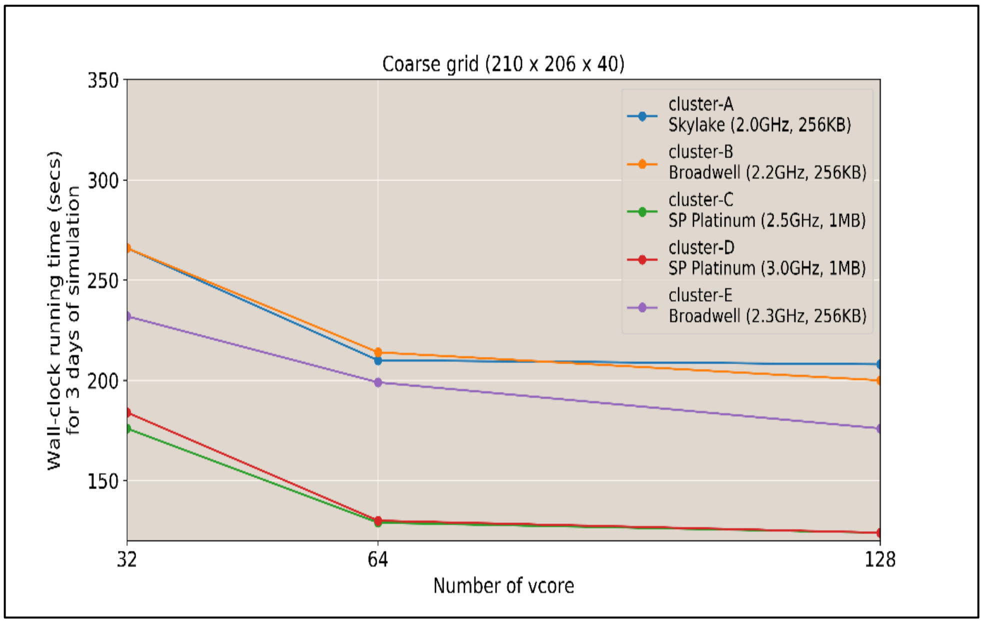


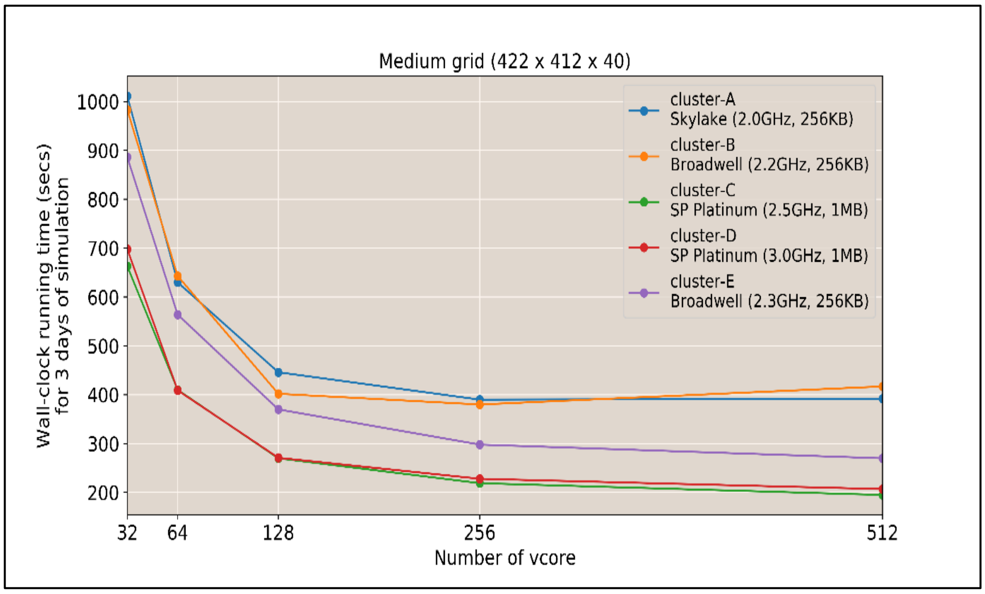
Figure 2.10. Memory latency time according to memory bandwidth in clusters C, D, and E.

Cluster−C with a 2.5 GHz CPU and 1 MB MLC type has the low memory latency. Its running time for ROMS is similar to that of Cluster−D with a 3.0 GHz CPU and 1 MB MLC cache. The latency time of the memory affects its I/O performance. In this experiment, the best execution performance of the ROMS and benchmark S/W appears in the resource type with large MLC cache and lower memory latency, as in clusters C and D. This suggests that latency should be also seriously considered in numerical modeling that needs huge memory I/O.

2.5.2. Performance of ROMS according to grid size

I divided our experimental results into three based on Degree Of Freedom (DOF) size (Table 2.6), to evaluate the execution time according to CPU resource type and memory I/O. The execution times differ according to CPU resource type, but the maximum performance of each cluster appears with similar number of cores (Figure 2.11).





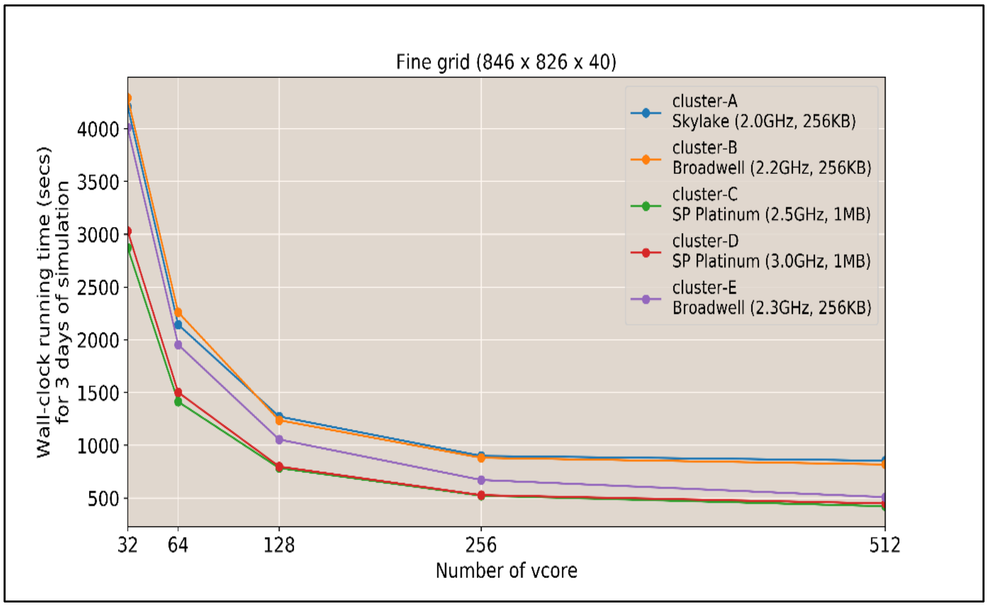


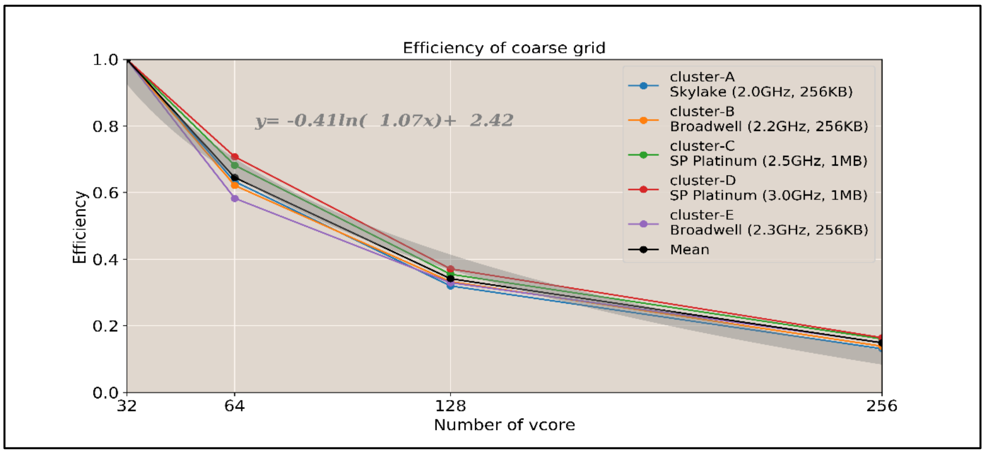
Figure 2.11 Wall−clock running time for 3 simulation days as a function of number of vCores for various grid sizes.

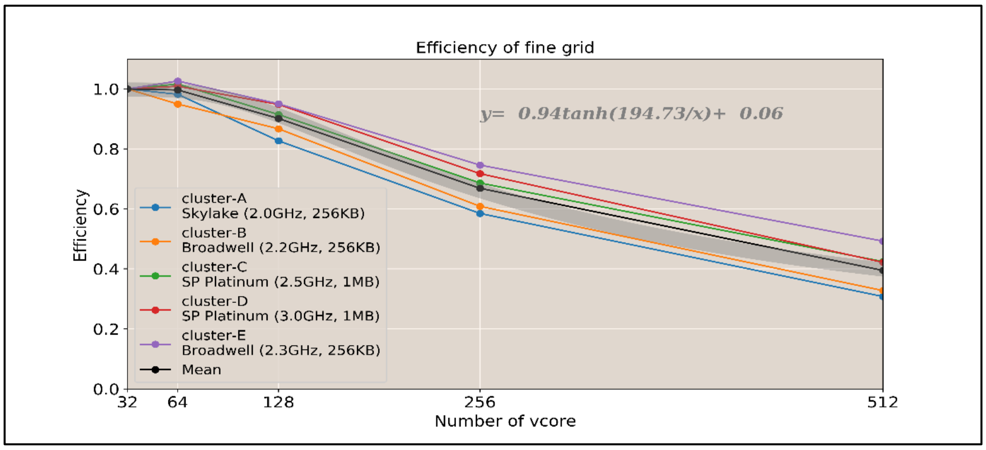
The execution time in the coarse grid (small DOF size) decreases as the number of cores increases. The minimum time appears at about 128 cores regardless of CPU resource type. The execution times decreases with the number of cores in all sizes. However, the decrease rate is small after 64 cores in the coarse grid and 256 cores in the medium and fine grids, respectively.

Table 2.6 Grid−Size Type of numerical ocean model

Resolution of grid	Coarse	Medium	Fine
Dimensions of grid	210 x 206 x 40	422 x 412 x 40	846 x 826 x 40
Degree Of Freedom (DOF)	1,730,400	6,954,560	27,951,840

I calculated relative efficiency by assuming one as the efficiency of 32 vcores. The efficiency according to the number of cores decreases rapidly regardless of grid size (Figure 2.12). As the number of cores increases, the efficiency decreases in all grids. The efficiency decreases more rapidly in coarse grid than in fine grid.





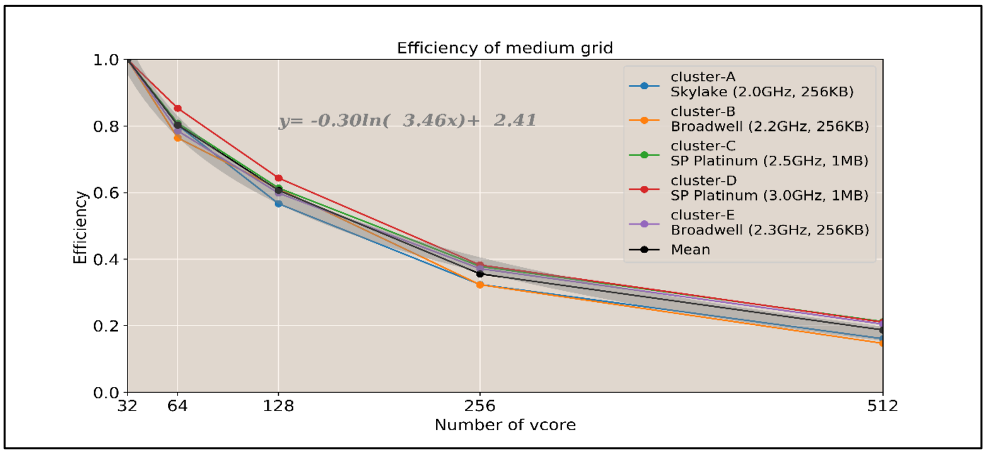


Figure 2.12 Efficiency of cluster as a function of number of vCores for various grid sizes. Thick gray fitting lines represent mean efficiency of clusters.

The fitting equations of mean efficiency with number of cores show quantitatively the decease rate according to the grid sizes. When the number of cores increases by a factor of four, the efficiency decreases to about 0.35 in the coarse grid with 1,730,400 of DOF but 0.82–0.95 in the fine grid with 27,951,840 of DOF (Figure 2.13).

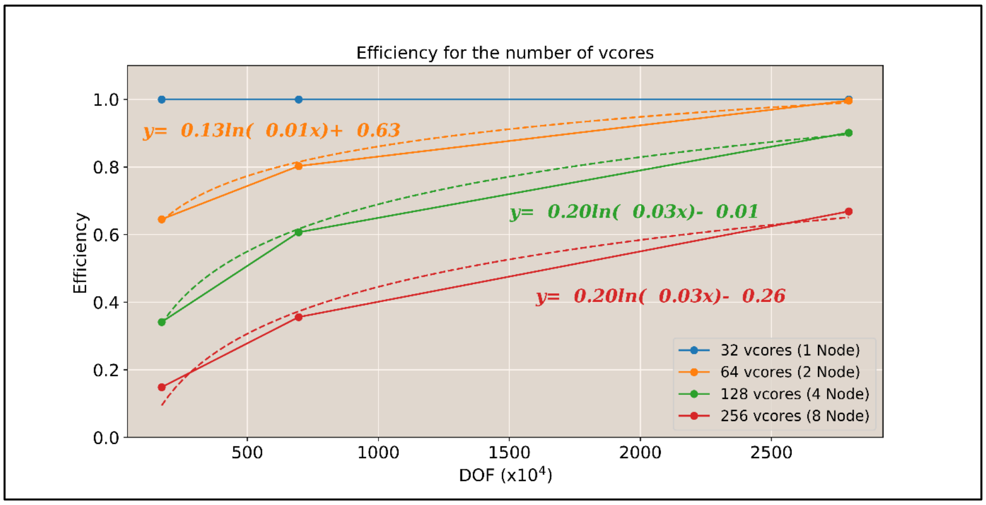
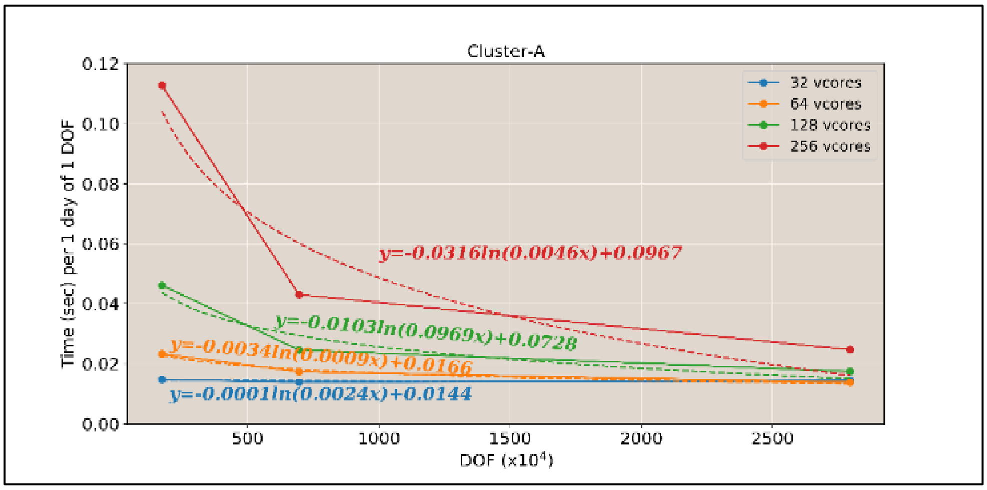
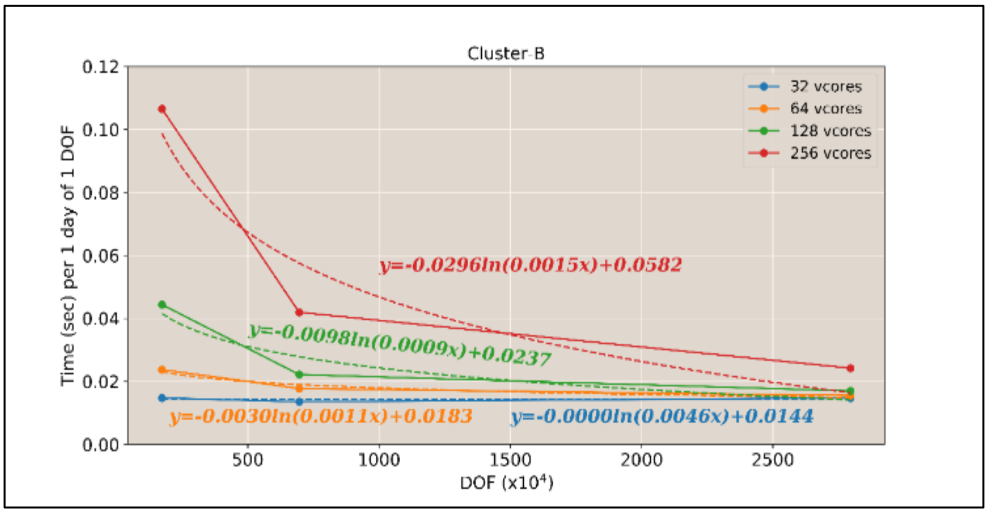


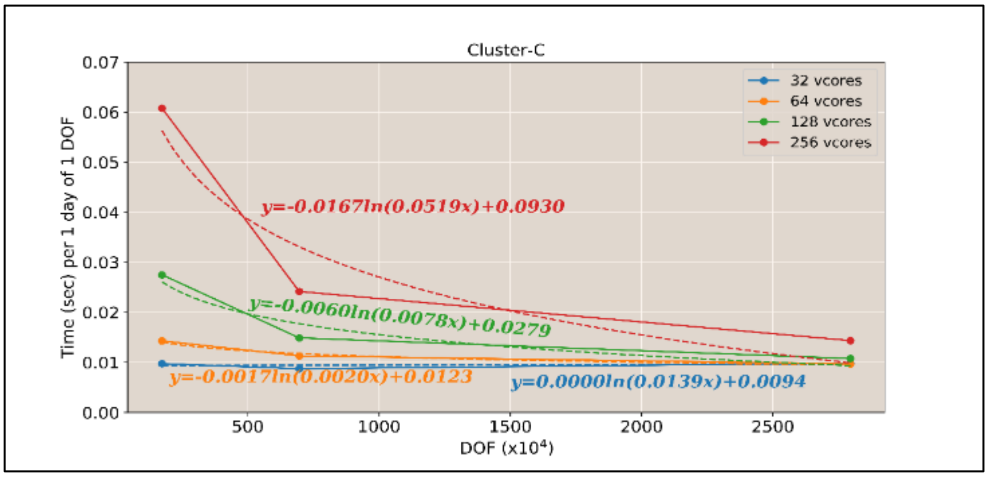
Figure 2.13 Mean efficiency of clusters as a function of grid size in different cores.

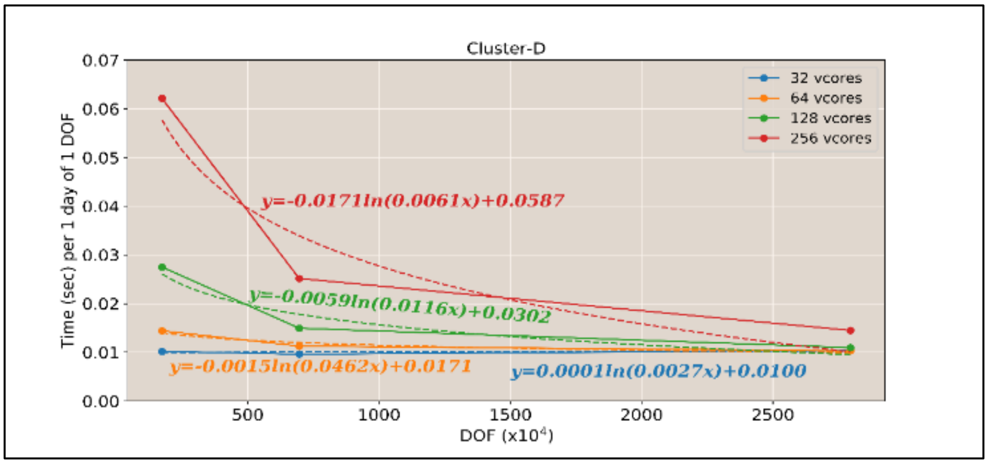
This result suggests that increasing the number of cores is less effective in a small DOF than in a large DOF. The fitting equations in Figure 2.13 provide useful information for selecting the best number of vcores according to the DOF.

Figure 2.14 shows the running time of one DOF per vcore for 1 day of simulation with the DOF according to the number of vcores for each cluster. Increasing the number of vcores is more effective in the large DOF than the small DOF regardless of clusters. The fitting equations in Figure 2.14 enable us to estimate the running time of ROMS according to DOF size and resource types in the virtualization environments or clouds.


Cluster-A
Time (sec) per 1 day of 1 DOF
DOF ($\times 10^4$)
32 vcores
64 vcores
128 vcores
256 vcores
y=-0.0316ln(0.0046x)+0.0967
y=-0.0103ln(0.0969x)+0.0728
y=-0.0034ln(0.0009x)+0.0166
y=-0.0001ln(0.0024x)+0.0144


Cluster-B
Time (sec) per 1 day of 1 DOF
DOF ($\times 10^4$)
32 vcores
64 vcores
128 vcores
256 vcores
y=-0.0296ln(0.0015x)+0.0582
y=-0.0098ln(0.0009x)+0.0237
y=-0.0030ln(0.0011x)+0.0183
y=-0.0000ln(0.0046x)+0.0144





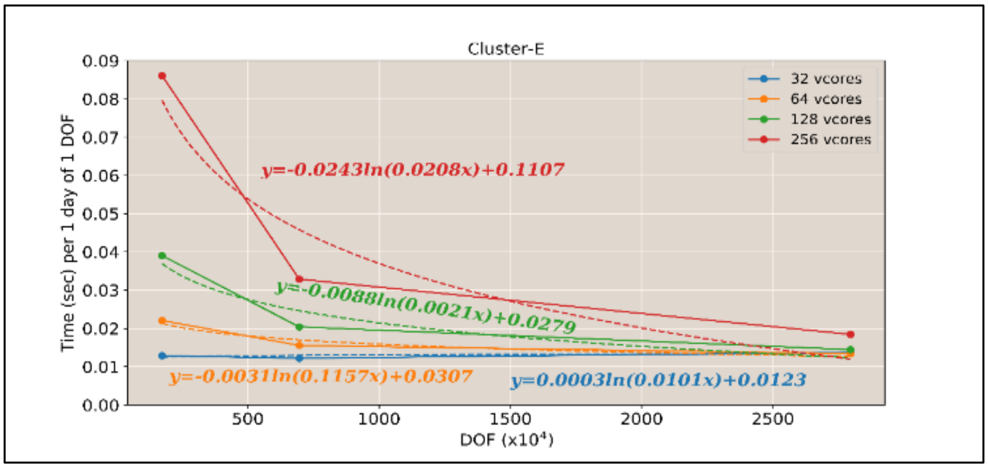


Figure 2.14 Wall−clock running time as a function of DOF for 1 day simulation of one DOF according to number of vCores.

2.6. Summary

In this study, I investigated how computing resources affect the performance of an MPI-based ROMS in virtualized cloud environments. To evaluate the performance more objectively, not only ROMS but also the benchmark S/Ws such as STREAM and HPL were executed in the virtualized clouds. Five clusters with different CPU and memory were tested to evaluate the performance of ROMS for three different grid sizes in the commercial clouds. I found that the cache hierarchy and capacity between the CPU and main memory play important roles in the performance of ROMS using huge memory. The performance of clusters absolutely depends on the MLC. Clusters of MLC (256 KB) show lower performance than those of the MLC (1 M) owing to relatively small cache memory capacity. The memory latency is also a key factor in the execution performance in the commercial cloud HPC environment. Clusters comprising many virtual cores and Ethernet based N/W in the commercial clouds were found to provide good performance in numerical ocean modeling. This result shows that increasing the number of cores is more effective in large DOFs than in small DOFs. The efficiency decreases more rapidly in a coarse grid than in a fine grid. If the number of cores increases by a factor of four, the efficiency decreases to about 0.35

in a coarse grid with 1,730,400 of DOF but 0.82–0.95 in a fine grid with 27,951,840 of DOF.

The performance of cloud computing environments is constantly improving, and various numerical models are being tested in cloud computing environments. Microsoft's Azure already supports InfiniBand N/W technology. N/W sensitive models may be tested in InfiniBand−supported cloud environments easily in the near future. Some numerical models might depend on the size of the memory according to the grid size and the communication latency between the nodes as well as the computation. These constraints can be satisfied by suitable resource selection and various configurations in cloud computing environments. The best performance of ROMS in commercial cloud computing environments can be achieved by selecting the appropriate CPU and memory and optimizing the modeling environment. The commercial cloud computing environment is a cost−effective solution for large−scale modeling. Various technologies and resource configurations are available for enhancing the security of cloud computing to the level of that of local HPC. Moreover, VM image copying techniques can be used to copy and share the model environment configuration of the ocean numerical model rapidly in cloud environments. This makes it easier to collaborate among researchers in a multinational context. Thus, cloud computing can provide an opportunity to focus on research and to minimize the time and cost of resources needed to construct a modeling environment.

3. Reproducibility of numerical ocean model on the cloud computing[②]

3.1. Introduction

Numerical ocean models are used to simulate the interactions among various elements of ocean systems. These models play very important roles in helping us understand and predict ocean dynamics. Many ocean models have been coupled with atmospheric models to consider the interactions of air and sea (Blackport et al., 2018). Recent advances in information technologies (IT), such as cloud computing, have enabled scientists to more easily run scientific models of this nature (Chen et al., 2017; Zhuang et al., 2020). Hence, emerging cloud technologies and case studies for ocean observations and modeling have been performed (Vance et al., 2019; Signell et al., 2019).

The IT infrastructures required for numerical models vary from personal computers (PC) to virtualized servers and various cloud services. Additionally, there are many software (SW) combinations of operating system (OS), compilers, and libraries used to support

[②] The results of the presented work have been published in Jung et al. (2021).

and implement the numerical models. The complexity of the IT environment increases rapidly with more complex SW model configurations. Fortunately, various cloud IT resources are easily procured. Nonetheless, complex environments tend to hamper model-building for computational reproducibility.

Survey results have revealed that some code execution environments can hinder the reproducibility required by geoscientists (Konkol et al., 2019). Therefore, several efforts (e.g., open reproducibility research) have resulted in improvements to such environments to support computational reproducibility (Open Reproducible Research, 2020) and to help researchers share experimental information and supportive infrastructures, including SW and data configurations. The Executable Research Compendium was designed for such reproducibility. It uses Docker containers, which applies an OS-level virtualization to deliver SW-container packages (Nüst et al., 2017). Containers are isolated SW bundles having libraries and configuration files that can communicate with each other through well-defined channels to provide flexibility and portability, enabling applications to run at various locations. Virtual boxes, containers, and Conda distributions have been used to abstract analytical environments at smaller instances, which has been helpful for the reproducibility of computational biology works (Grüning et al., 2019). Additionally, studies have tested Kubernetes container-orchestration systems and the performance of other SW benchmark for the feasibility of various scientific workloads (Beltre et al., 2019).

Prior studies have shown the advantages and future possibilities of container environments in benchmark cases. In this study, I apply a container-based architecture for geoscientific studies, including numerical ocean modeling. I propose that the Kubernetes-managed container cluster architecture for numerical ocean modeling be used to increase computational reproducibility and achieve the needed portability to support numerical ocean models in various public and private clouds. This architecture saves time when setting up numerical ocean models with their pre-built container images, and it resolves the vendor lock-in problem of cloud computing. These benefits allow greater flexibility of model transfer among private and public clouds.

The rest of the paper is organized as follows. Section 3.2 introduces our regional ocean-modeling system (ROMS) and its container and container-based numerical modeling architecture. Section 3.3 explains the containerization of ROMS and the implementation of container orchestration for parallel processing in various environments. Section 3.4 presents the results of our ROMS modeling reproducibility tests in various runtime environments, highlighting architectural feasibility. Finally, in Section 3.5, I summarize the achievements of our containerized ROMS execution architecture and provide necessary future improvements for reproducible and portable SW architectures.

3.2. Containerization of numerical ocean model

3.2.1. Container virtualization

Container virtualization is a lighter−weight virtualization technology than traditional hypervisor−based server virtualizations. For comparison purposes, server virtualizations use a hypervisor, which functions to logically divide and allocate physical server resources (Figure 3.1). Each guest OS is installed on the virtualized server, and the user's SW is installed atop it. In contrast, container virtualization uses a container daemon or a SW engine to create a logically isolated unit (i.e., container) based on the capabilities of the control groups (cgroup) and namespaces of Linux OS kernels. The containers work like independent servers on the host OS. Unlike a virtual machine (VM), separated cgroups and namespaces within the same OS allow separate central−processing unit (CPU), input/output (I/O), internet protocols (IP), and user spaces for each container. A cgroup is a Linux kernel feature that isolates and limits resource types (e.g., CPU, memory, disk I/O, and network (NW)) (Sultan et al., 2019).

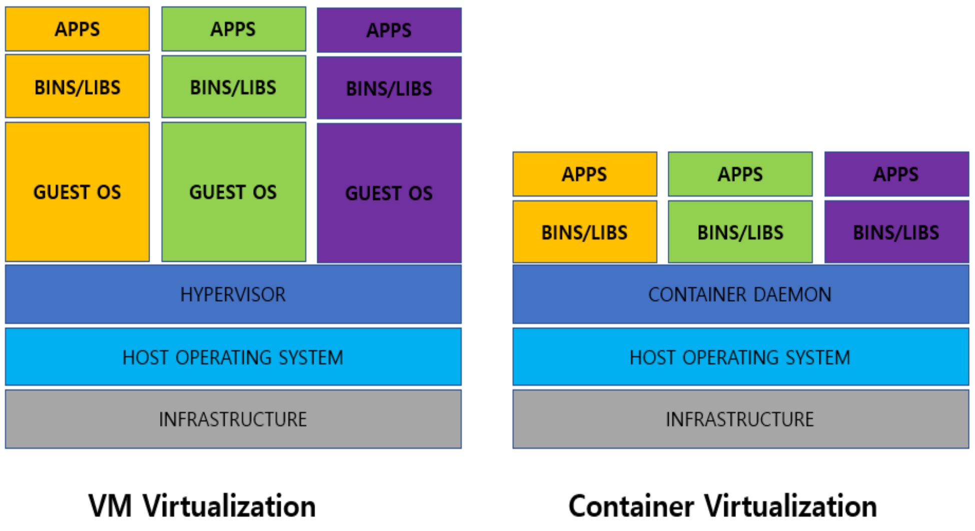


Figure 3.1 Conceptional architecture of VM and container virtualization.

Logically separated containers using Linux kernels do not have independent guest OS layers. They instead share features of the host−OS kernel. This architecture characteristic renders a lighter weight environment. Many users use the de facto Docker environment to deliver SW container packages (Shah et al., 2019). Various container runtimes are compatible with Kubernetes (Kubernetes Container Runtime, 2022) and the Singularity container engine can be used for other high−performance computing (HPC) to package entire scientific workflows, SW and libraries, and data to leverage the performance of local servers. Furthermore, InfiniBand, a computer−NW communications standard used in HPC, features very high throughput with very low latency and is used to improve security (Veiga et al., 2019). These engines and compatible environments are enabled by an open−source compatibility standard specification (i.e., Open Container Initiative (OCI)), which is used to maintain collaboration capability and compatibility with other tools (Linux Foundation, 2020). Kubernetes can orchestrate various containers using OCI standard specification.

3.2.2. Container−based architecture for HPC

Traditionally, for numerical ocean modeling tasks, a large number of physical servers and high−speed NW switches (e.g., InfiniBand) is required to create HPC clusters. As virtualization technologies and cloud environments have become more commonplace, many earth−science numerical models have been deployed on MPI clusters using VMs in public clouds (Montes et al., 2017). A container−based cluster can be configured on multiple nodes using container runtimes and Kubernetes settings instead of installing individual SW packages for MPI on physical servers or logically configuring virtual servers. Container clusters can also be configured on a SW−defined logical

NW layer (e.g., overlay NW), such that the containers of each node are well connected. In a native container-based cloud environment, the NW-plugin container of each node enables rapid NW configuration (Luksa, 2018). Figure 3.2 illustrates the configuration of the overlay NW used by the Kubernetes container cluster (Kubernetes Cluster Networking, 2022). With this configuration, it is easy to extend and run numerical models from a single-node container cluster to a multi-node one. In this study, I construct clusters of homogeneous and heterogeneous OS environments and verify their portability, scalability, and computational reproducibility using overlay NW features. Note that an *etcd* is a strongly consistent, distributed key-value store that provides a reliable way to store data so that it can be accessed by a distributed system or a cluster of machines. Additionally, communication between nodes is made possible via the application programming interface (API) server and NW topology using overlay NW (Luksa, 2018; Kubernetes Components, 2022). Furthermore, a Kube proxy is a NW Kubernetes proxy that runs on each node in the cluster, implementing parts of the Kubernetes service. The kubelet is the primary node agent that runs on each node, and it manages *pods*, the smallest deployable unit of computing that one can create and manage in

Kubernetes.

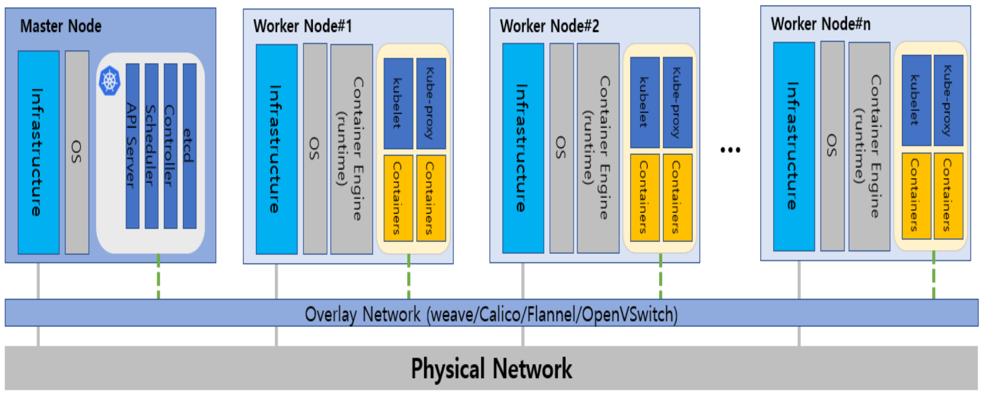


Figure 3.2 Conceptual diagram of the Kubernetes overlay NW.

With the advancement of container virtualization technologies, many orchestration SWs have been developed to effectively manage and operate large numbers of containers. Currently, the most commonly used container orchestration SW is Kubernetes, which was developed by Google as an opensource project in 2014 (Kubernetes overview, 2022). Figure 3.3 shows the conceptual architecture. Kubernetes controls and manages containers as basic units, (i.e., pods) (Kubernetes Pods, 2022), making it possible to communicate among them at every node using NW components based on an overlay NW. I use StatefulSet, a Kubernetes controller for HPC container orchestration, to deploy numerical model containers to multiple nodes. Note that *kubectl* is a command-line tool that allows operators to run commands against Kubernetes clusters. The *kubectl* tool can be used to deploy applications, inspect and manage cluster resources, and view logs. YAML is a human−readable data−serialization language commonly used for configuration files and in applications where data is being stored or transmitted.

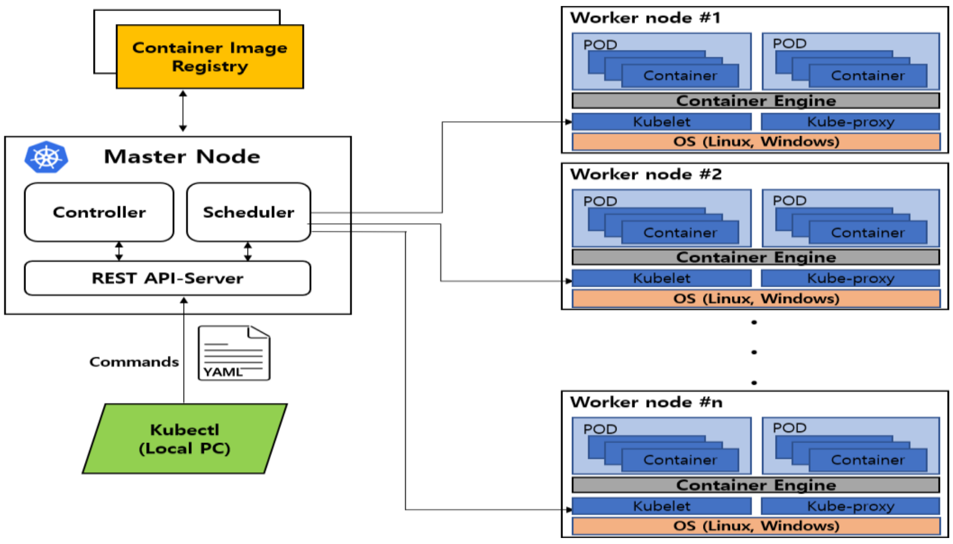


Figure 3.3 Architecture of Kubernetes clusters.

The Kubernetes orchestration solution provides scalability and easy container management for numerical modeling. Various nodes can easily be merged into a container cluster regardless of the homogeneous or heterogeneous nature of the OS. StatefulSet codes can then be executed for deployment, distribution, and parallelism of containers. When code is executed on the Kubernetes engine, the executable numerical model image is downloaded and configured on designated nodes. Users can then easily run numerical models on the container cluster.

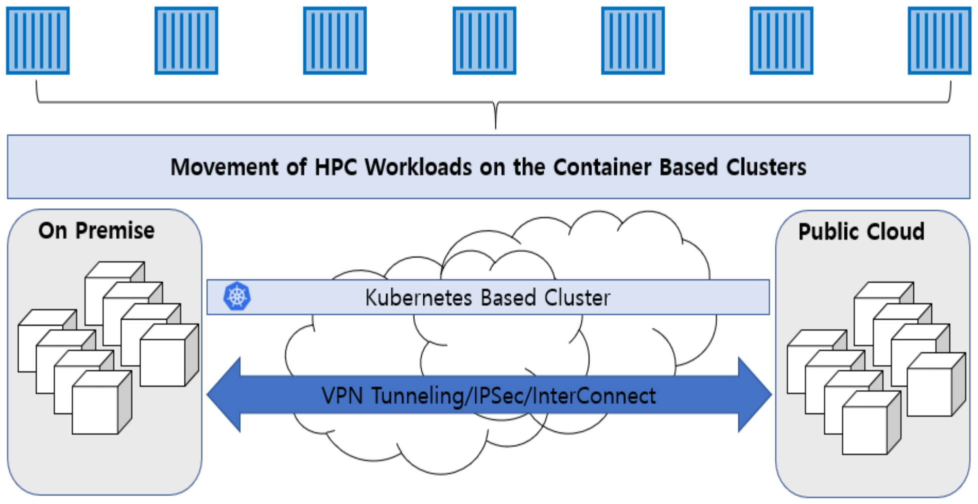
Kubernetes supports a variety of platforms (e.g., Linux and Windows Server). Kubernetes Compatible SWs (e.g., MicroK8s) (Canonical, 2022) can be used to run MPI jobs on a PC or a single node. End-users can manage container clusters for numerical modeling using container deployment techniques and numerical lump modeling. Users can register various numerical model images in public repositories for sharing publicly or privately. For detailed technical specifications, please visit the Kubernetes official documentation site (Kubernetes Overview, 2022).

3.2.3. Container-based architecture for hybrid cloud

Hybrid clouds combine a private cloud with one or more public clouds by using proprietary SW and NW environments that enables communication between distinct infrastructures (Hybrid clouds, 2022). A hybrid cloud provides large flexibility in HPC infrastructures, because it enables users to easily move HPC workloads between public and private infrastructures according to environmental changes. Moreover, researchers can back-up data on a private cloud or local data center and leverage more economical and available computational resources in the public cloud. Network infrastructures, such as virtual-private-network (VPN) tunneling environments or cloud interconnect lines, enable users to connect each infrastructure organically for HPC jobs (AWS, 2022f).

When researchers want to move workloads for simulation from a premise cluster to public cloud, it is important to easily deploy HPC workloads into heterogeneous cloud environments without barriers. If the suitable architecture for flexibility of infrastructure between public clouds and private infrastructure is available, users can extend to the public cloud when computational resources exceed local availability.

In this research, I suggest that the container-based architecture is suitable for the flexibility of workloads in the hybrid cloud. There are several methods of creating secure NW infrastructures between public clouds and private infrastructure. These methods include CloudVPN, IPsec, routers, etc. (Google, 2022b). After configuring the NW infrastructure between two environments, the server nodes can communicate with each other as if they were on the same local infrastructure. Figure 3.4 shows the container-based architecture for HPC workload distribution between public and private clouds.

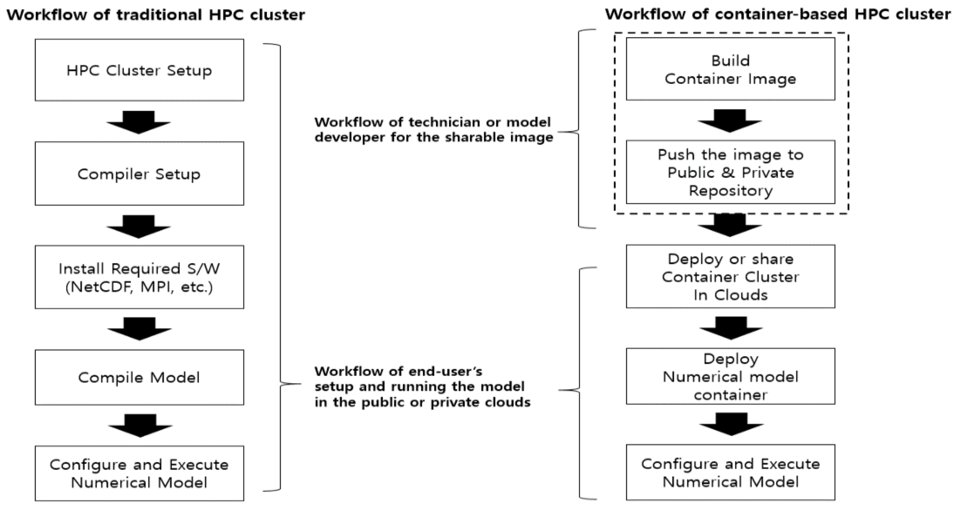


Movement of HPC Workloads on the Container Based Clusters
On Premise
Kubernetes Based Cluster
VPN Tunneling/IPSec/InterConnect
Public Cloud

Figure 3.4 HPC workload distribution above hybrid cloud with Kubernetes

3.3. Materials and Methods

3.3.1. Comparison of traditional and container based HPC cluster workflows



Workflow of traditional HPC cluster
HPC Cluster Setup
Compiler Setup
Install Required S/W (NetCDF, MPI, etc.)
Compile Model
Configure and Execute Numerical Model
Workflow of technician or model developer for the sharable image
Workflow of end-user's setup and running the model in the public or private clouds
Workflow of container-based HPC cluster
Build Container Image
Push the image to Public & Private Repository
Deploy or share Container Cluster In Clouds
Deploy Numerical model container
Configure and Execute Numerical Model

Figure 3.5 Conceptual workflow of VM and container clusters for numerical modeling.

Figure 3.5 shows the workflow of the preparation and execution of both the traditional and container numerical−model architectures. In the container−based architecture, a scientist or a technician builds a runnable container image and pushes it to a public or private repository for end users who then download the model image into public or private container clusters. After constructing the container cluster, it becomes possible to run the numerical model container. StatefulSet codes (i.e., workload API objects used to manage containers) are implemented on cluster nodes so that the end user can run the numerical−model HPC clusters. In contrast, end users of the traditional cluster prepare additional jobs, such as preparation of prerequisite SW for the traditional architecture (not needed for the container−based cluster). In particular, end users of the traditional architecture must compile their own model codes and required SWs (e.g., NetCDF and MPI libraries), depending on the OS version and compiler type. Moreover, the compiled executable program is not shareable to other users because of SW and infrastructure dependencies. Thus, it becomes difficult and time−consuming to prepare model environments based on VMs or bare−metal clusters in public or private clouds.

3.3.2. Model domain and datasets for numerical simulation

For our study, the model domain includes the northwestern Pacific region and several marginal areas, such as the East China Sea, the Yellow Sea, and the East/Japan Sea, ranging meridionally from 15° N to 52° N and zonally from 115° E to 162° E (Figure 3.6).

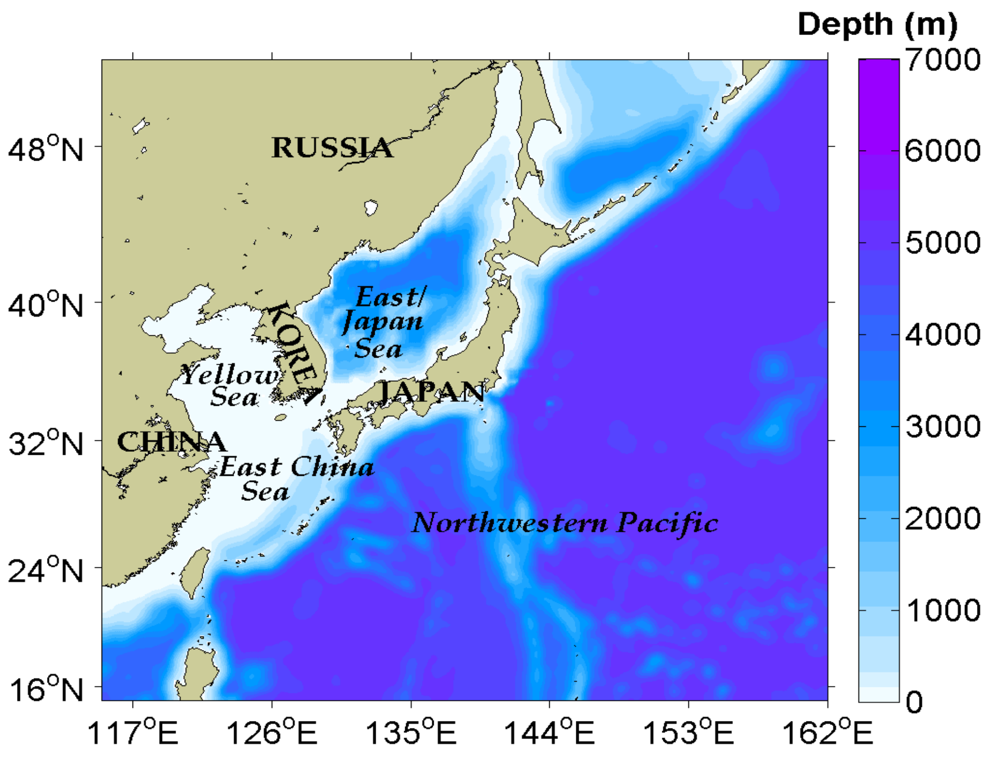


Figure 3.6 Model domain for ROMS simulation.

The models have horizontally 1/20°, 1/10°, and 1/5° grid resolutions and 40 vertical layers. The Earth Topography 1-min grid (ETOPO1) dataset of the National Geophysical Data Center was used for the bottom topography (Amante et al., 2009). Initial temperature and salinity were derived from the World Ocean Atlas 2009 (WOA09)

(Antonov et al., 2010; Locarnini et al., 2010). The air−forcing datasets, including daily mean solar radiation measures, air temperatures, wind, sea−level pressures, relative humidity, and precipitation counts, were obtained from the ERA−Interim reanalysis dataset of the European Center for Medium−Range Weather Forecasts (Dee et al., 2011). Monthly averaged velocities, temperatures, and salinities from the Simple Ocean Data Assimilation were applied for the lateral open−boundary condition (Carton, 2008). These data were employed for estimating the bulk formulae (Fairall et al., 1996). Freshwater discharges from 12 rivers were included (Vörösmarty et al., 1996; Wang et al., 2008). Tidal forces extracted from TPXO7 were applied at the open boundary (Egbert et al., 2002). The simulation time was 30 days from 1 to 30 January 2010. The timestep size for 3D equations was 90 s, and the number of time steps for 2D equations between each 3D time steps was set to 10. Two types of output files were created for intervals of 480 time steps; one was an averaged file for the interval, and the other was a snapshot file at intervals. The output files included the sea surface height, 2D and 3D horizontal velocities, and tracers. The Chapman implicit, Flather, and clamped schemes were applied to lateral boundary conditions for the sea−surface height, barotropic component of velocity and 3D velocity, and tracers, respectively

(Chapman et al., 1985; Flather et al., 1976). Horizontal harmonic mixing coefficients for tracers and momenta were set as 20 and 100 m^2s^{-1}, respectively. A quadratic bottom stress scheme was employed, and its coefficient was 0.0026. Surface- and bottom-stretching parameters for controlling thickness of the vertical layer were set to 5 and 0.4.

I then compared the performance using three types of grids. Table 3. 1 shows the specifications of the three grid-model resolutions.

Table 3.1 Grid Size and Degree of Freedom in Each Experiment

Type	Coarse	Medium	Fine
Dimension of grid	210×206×40	422×412×40	846×826×40
Degree of Freedom	1730400	6954560	27951840

3.3.3. Building the container image and registration in the repository

I designed and implemented the execution architecture of the numerical ocean model using Kubernetes and Docker containers. The Docker-compatible image was a lightweight, standalone, executable SW package that included dependent SWs with appropriate libraries needed to run ROMS 3.6. Most geoscientific numerical models are compiled and executed using FORTRAN or C/C++ typically on a Linux OS. ROMS can also be compiled to generate executable files.

They utilize NetCDF libraries for data I/O and MPI libraries and an execution environment for parallel processing. The Docker file can be executed to create the base image, including configured environments and required SW installations. Table 3.2 lists the SW and libraries in the ROMS containers used for this study. The model scientists build the specific image includes required SWs using Docker commands before pushing the image into the public repository.

Table 3.2 SW Configurations for the Numerical Model Images.

Software	Name	Version	Purpose
Compiler	*gcc/gfortran*	4.3	Compile numerical ocean model and I/O library
Ocean Model	ROMS	3.6	Simulate ocean physical properties such as Sea Surface Temperature and u and v vectors of ocean current
I/O Library	NetCDF	4.1	Read and write model input/output data
MPI	OpenMPI	3.1.4	Parallelize ocean-model processes

Figure 3.7 shows the Docker file for generating the ocean modeling image. In the head of the file, base Linux type, such as Ubuntu or CentOS, were written. Then, required SW and compiler steps for Linux OS were written. The Fortran compiler, OpenMPI, and NetCDF are required for generating ROMS executable codes. This Docker code is the main part of the required SW installation for the ocean modeling program. Other required SW can be installed using the same

syntax styles. Users can also enter deployed pods and check compiled programs to modify or reconfigure their sources inside the pods. A full compiled image of this research is downloadable from the Docker hub (next7885/ubuntu_roms_k8s_hpc) and deployment codes of the containerized ocean model are available at Zenodo (https://doi.org/10.5281/zenodo.4015246).

```
# Install ubuntu
FROM              ubuntu:18.04
MAINTAINER          next7885@snu.ac.kr
RUN               apt-get -y update
RUN               apt-get install -y openssh-server

# Install gcc
RUN apt-get -y install apt-utils
ENV DEBIAN_FRONTEND noninteractive
RUN apt-get -y install gcc
RUN apt-get -y install g++
RUN apt-get -y install gfortran
RUN apt-get -y install wget
RUN apt-get -y install file

# Install Open-MPI
RUN         wget         https://download.open-mpi.org/release/open-
mpi/v3.1/openmpi-3.1.4.tar.gz
RUN tar -xvf ./openmpi-3.1.4.tar.gz
RUN export CC=gcc
RUN export CXX=g++
RUN export FC=gfortran
RUN export PATH=$PATH:/usr/bin:/usr/local/mpi/bin
RUN                                                                     export
LD_LIBRARY_PATH=$LD_LIBRARY_PATH:/usr/lib:/usr/lib64:/usr/local/m
pi/lib
WORKDIR /openmpi-3.1.4
RUN apt-get -y install make
RUN ./configure --prefix=/usr/local/mpi
RUN make
RUN make install
RUN apt-get -y install git
RUN apt-get -y install iputils-ping
RUN apt-get -y install net-tools

# Install netcdf
RUN  wget  https://www.unidata.ucar.edu/downloads/netcdf/ftp/netcdf-c-
4.7.1.tar.gz
RUN   wget   https://www.unidata.ucar.edu/downloads/netcdf/ftp/netcdf-
fortran-4.5.1.tar.gz
RUN wget http://www.zlib.net/zlib-1.2.11.tar.gz
RUN     wget     https://support.hdfgroup.org/ftp/HDF5/current/src/hdf5-
1.10.5.tar.gz
RUN tar -xvf ./hdf5-1.10.5.tar.gz
RUN tar -xvf ./zlib-1.2.11.tar.gz
WORKDIR /openmpi-3.1.4/zlib-1.2.11
RUN ./configure --prefix=/usr/local/zlib
RUN make clean
RUN make && make install
RUN export PATH=$PATH:/usr/local/mpi/bin
WORKDIR /openmpi-3.1.4/hdf5-1.10.5
RUN export CC=mpicc CPPFLAGS=-I/usr/local/hdf5/include LDFLAGS=-
L/usr/local/hdf5/lib
RUN  ./configure  --prefix=/usr/local/hdf5  --with-zlib=/usr/local/zlib  --
```

```
enable-hl
RUN make clean
RUN make && make install
WORKDIR /openmpi-3.1.4
RUN tar -xvf ./netcdf-c-4.7.1.tar.gz
RUN tar -xvf ./netcdf-fortran-4.5.1.tar.gz
```

Figure 3.7 Required SW installation codes for generating ocean-modeling image in Docker file

It is possible to share the Docker file to create an image or model provision. If an image is registered in the public repository, any user can download the container image to run the model. The Docker file used to create the numerical model setup is a text file, and the images can be re-created via Docker-file modification and rebuilding. Several web sites (e.g., Docker-hub) offer a variety of container images for public users. Every image has a specific uniform resource locator and is accessible without requiring additional effort. In this study, I created a ROMS 3.6 image and shared it to the Docker-hub repository for scientific reproduction. Any end user can download it using a simple Docker command (i.e., docker pull next7885/ubuntu_roms_k8s_hpc).

If the size of the input dataset for the model is small, it is possible to merge the data and the executable file when creating the image.

3.3.4. Configuring a numeric model execution cluster

After registering the container images to the repository, the container cluster was configured for numerical modeling. There are two ways to do this. The first method requires installation of the Kubernetes or compatible SW directly to the server or PC. The other method requires the use of a Kubernetes cluster provided by the public cloud. Because GCP, AWS, and Azure have recently provided Kubernetes-based clusters, users can easily use them to reduce runtime costs (AWS, 2022a; google 2022a; Azure 2022). I manually placed the Kubernetes clusters on public clouds, a private cloud, and a PC for various cloud environmental testing. A ROMS image registered in the public repository was downloaded to all nodes, and the model running environment was constructed using the developed StatefulSet codes. Table 3.3 and 3.4 and Figure 3.8 shows various SW and hardware configuration of local clusters for the numerical ocean model of this study.

Table 3.3 HW and SW Configuration of Local Clusters

	Laptop−PC	Local Cluster#1	Local Cluster#2
CPU Type	Intel−i7	Intel Xeon	Intel Xeon
Hypervisor	VirtualBox	KVM	KVM
Guest OS	Windows 10	CentOS 7.5	Ubuntu 18.04
Nodes	1 Node	4 Nodes	3 Nodes
vCores/Memory		8Cores/16G	4Cores/8G
Container Runtime	Containerd	Docker v19.03.12	Docker v19.03.6
N/W Interface	LAN	LAN	LAN
Orchestration Tool	Microk8s	Kubernetes v1.18.3 minikube	Kubernetes v1.18.3

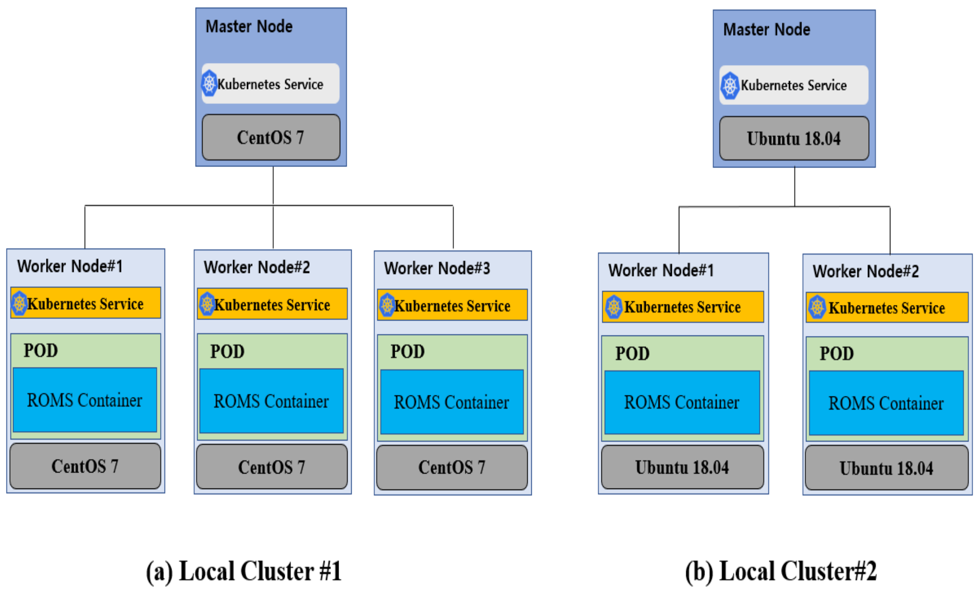


Figure 3.8 Kubernetes Cluster Configurations of local clusters

Table 3.4 HW and SW Configuration of Clusters on Public Cloud

	AWS	Google	Azure
CPU Type	Intel Xeon, AMD	Intel Xeon	Intel Xeon, AMD
OS	Ubuntu 18.04, CentOS 7.5	Ubuntu 18.04	Ubuntu 18.04, CentOS 7.5
Container Runtime	Docker v19.03.6	Docker v19.03.6	Docker v19.03.6
Orchestration Tool	Kubernetes v1.18.3	Kubernetes v1.18.3	Kubernetes v1.18.3

Notes: Containerd is an industry-standard container runtime that emphasizes simplicity, robustness, and portability. Ubuntu is an open-source SW OS that runs from a PC to the cloud. Minikube is tool to run single node Kubernetes cluster in VM on PC or servers locally.

3.3.4.1. Codes for deploying containers on cluster

The StatefulSet controller sequentially distributes and manages container distributions to nodes (Kubernetes Controller, 2022). The internal domain name-service function provided by Kubernetes was utilized for communication between the pods. Figure 3.9 shows an example of the StatefulSet code for deploying the numerical ocean-modeling containers into worker nodes for parallel processing. It contains various information of the container distribution (i.e., replicas standing for the number of containers, image name in the repository, container ports, etc.) for users intuitively.

Users can allocate modeling containers to nodes by easily checking and changing configuration, such as replicas according to their environment.

```
apiVersion: apps/v1
kind: StatefulSet
metadata:
  name: roms-ssh-statefulset
  namespace: roms-k8s
  labels:
    app: roms-ssh
spec:
  replicas: 2
  selector:
    matchLabels:
      app: roms-ssh
  serviceName: "roms-ssh"
  template:
    metadata:
      labels:
        app: roms-ssh
    spec:
      serviceAccountName: sa-roms
      containers:
      - name: roms-ssh
        image: next7885/ubuntu_roms_k8s_hpc
        resources:
          limits:
           cpu: "8"
          requests:
           cpu: "3"
        volumeMounts:
        - mountPath: /NWP
          name: pvroms
        command: ["/bin/sh", "-c"]
        args:
          - echo starting;
            /usr/sbin/sshd;
            sleep 360000;
            echo done;
        ports:
        - containerPort: 22
```

```
          lifecycle:
              postStart:
                exec:
                  command:      ["/bin/sh",      "-c",      "cat
/NWP/id_rsa.pub >> /root/.ssh/authorized_keys"]
        volumes:
        - name: pvroms
          persistentVolumeClaim:
            claimName: romsclaim
```
Figure 3.9 ROMS StatefulSet code.

Users can construct their own environment by deploying container images to cluster nodes without setting up an additional MPI environment in the Kubernetes cluster. This method is helpful to the portability of MPI clusters. Various models and versions can be easily deployed and tested in the same cluster environment using separately compiled container images. Figure 3.10 shows the conceptual diagram that uses Kubernetes for the numerical model of the container clusters in the private or public clouds. Users can perform modeling in various public and private environments to meet their purposes and to share environments with coworkers. Various computational environments might be simultaneously needed for ensemble modeling, which requires a vast amount of IT resources. Thus, this architecture can be a cost-saving and efficient alternative to specific vendor lock-in scenarios and IT infrastructure dependencies.

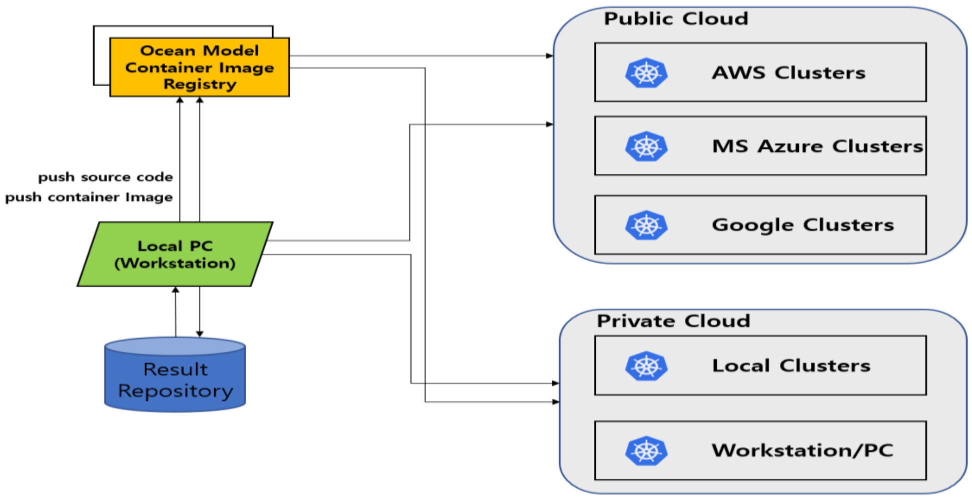


Figure 3.10 Conceptual diagram of the container−based clusters for public or private clouds.

Users can control and manage various infrastructure environments by using the StatefulSet code. After Kubernetes installation, users can add a configuration file to support the distribution of the MPI−contained image parallelized across multiple nodes. In this study, the StatefulSet code was developed and tested to distribute the image to worker nodes using *yaml*, a human−readable data−serialization language, which is commonly used for configuration files and applications where data are stored or transmitted (YAML, 2021). I configured our environment using standard StatefulSet codes, used to test various execution environments (e.g., macOS, Windows, local servers, and public clouds). The code included the number of pods, the container image, the type of Kubernetes controller, and disk information. I also configured the NW file−system volume type so that the persistent volume could share the volume among containers (Luksa, 2018; Kubernetes Volumes, 2022).

Kubernetes utilizes a SW-defined overlay NW that resides on the physical NW (Kubernetes Cluster Networking, 2022), and it deploys various NW driver plugins (e.g., Calico, Weave-Net, and Flannel) for various purposes. The Flannel NW driver is used to carry out inter-pod communications. Traditionally, InfiniBand is used to reduce the inter-node latency for MPI performance. Additionally, remote direct-memory access (RDMA) support is available through a container NW interface (Beltre, 2019). If users already have the Kubernetes cluster or a Docker-container cluster, they can deploy the registered container image to the worker node without any additional work. This setup helps users immediately verify and utilize the model. I configured the ROMS model environment in 30 min on the public cloud using developed codes and images.

3.3.4.2. Deployment of a model container on a worker node

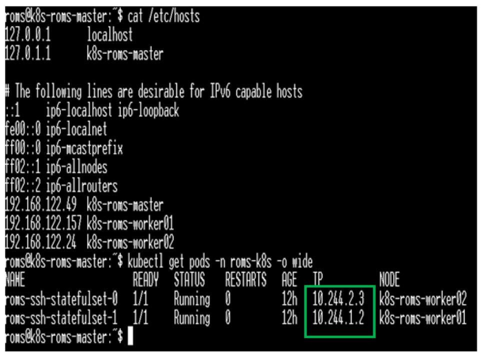
The downloaded image can be executed by the container runtime engine installed at the node. For parallel processing of the numerical models, the container runtime at each node must be tightly connected and controlled. In the case of numerical models, I recommend that one container be run on each server to minimize pod communications. Table 3.5 shows the performance penalty difference between one and two containers on one node. If two or more containers are deployed on the same server, communications between containers will cause a performance penalty in the HPC cluster. This result shows a performance penalty of 4%–7% according to the number of containers per worker node.

Table 3.5 Runtimes and Performance Penalty According to Grid Resolution

Resolution of grid	Coarse	Medium	Fine
Dimension of grid	210 × 206 × 40	422 × 412 × 40	846 × 826 × 40
Wall-clock time (s) of one container per a VM	288	1178	5825
Wall-clock time (s) of two containers per a VM	308	1222	6135
Performance Penalty	7%	4%	5%

3.3.4.3. Deployment of multiple containers and running MPI jobs on container cluster

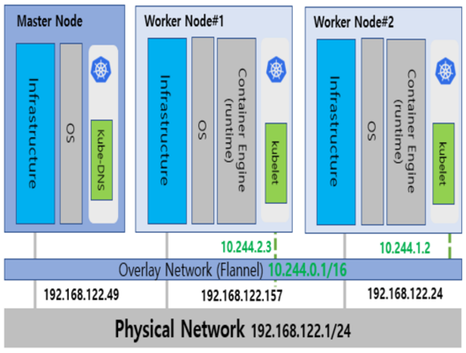
There are a few preparation steps for deploying the StatefulSet in the multiple nodes. The first step is to create a namespace for the specific logical area in the Kubernetes cluster (Kubernetes Namespace, 2022). The second step is to declare the storage volume for multiple pod access, to read and write for modeling, and to create the service account. The final step is to deploy StatefulSet into multiple server nodes. All steps are coded, and the code created for this study can be downloaded from Zenodo (https://doi.org/10.5281/zenodo.4015246). After deploying the pods to multiple nodes, their status in the clusters can be checked before running the numerical ocean modelling jobs. Each pod of the node has a private overlay NW IP address above their static IP address. This overlay IP address is used for executing *mpirun* instead of node's static IPs in the Kubernetes cluster. During the stage of running the MPI jobs in Kubernetes cluster, the pod's IP address inside the pod should be selected to execute the *mpirun* command. Figure 3.11 shows overlay NW configuration of the pods and the static IP configuration of the nodes in the cluster.



```
roms@k8s-roms-master:~$ cat /etc/hosts
127.0.0.1       localhost
127.0.1.1       k8s-roms-master

# The following lines are desirable for IPv6 capable hosts
::1     ip6-localhost ip6-loopback
fe00::0 ip6-localnet
ff00::0 ip6-mcastprefix
ff02::1 ip6-allnodes
ff02::2 ip6-allrouters
192.168.122.49  k8s-roms-master
192.168.122.157 k8s-roms-worker01
192.168.122.24  k8s-roms-worker02
roms@k8s-roms-master:~$ kubectl get pods -n roms-k8s -o wide
NAME                    READY   STATUS    RESTARTS   AGE   IP            NODE
roms-ssh-statefulset-0  1/1     Running   0          12h   10.244.2.3    k8s-roms-worker02
roms-ssh-statefulset-1  1/1     Running   0          12h   10.244.1.2    k8s-roms-worker01
roms@k8s-roms-master:~$
```

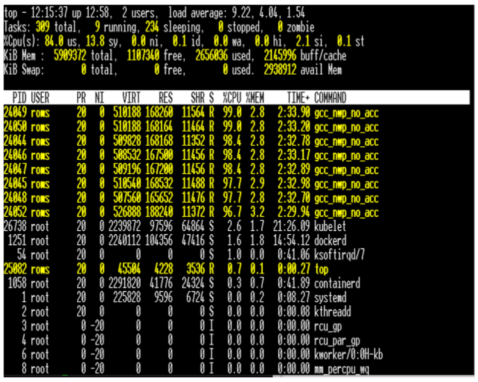
(a) Network configuration of pods



(b) Network conceptual diagram

Figure 3.11 a) Network configuration of pods and b) NW diagram on Kubernetes clusters

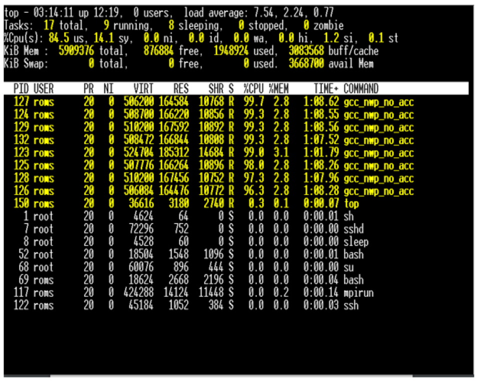
A pod plays the role of an abstraction layer above the node using an overlay NW and a container runtime. After running the jobs, MPI job processes are generated from the pod in each node. Compared to the traditional cluster node, each pod has only related processes of user applications. Conceptually, I can visualize the pod as an abstraction node for running MPI jobs. That is why I chose the resource controller type as the StatefulSet for MPI jobs in the Kubernetes cluster. In Section 3.4.2, I suggested one pod per node for improved performance. I can also operate each pod as an abstracted host node. Figure 3.12 shows the processes of MPI jobs inside the pod compared with the processes of the node. Container virtualization shares the common OS layer and isolates the user processes for the user application. The pod shows process collections of the user application layer. During the process monitoring of the host OS, users can see real MPI processes that are generated from ocean−model pods.



```
top - 12:15:37 up 12:58,  2 users,  load average: 9.22, 4.04, 1.54
Tasks: 309 total,   9 running, 234 sleeping,   0 stopped,   0 zombie
%Cpu(s): 84.0 us, 13.8 sy,  0.0 ni,  0.1 id,  0.0 wa,  0.0 hi,  2.1 si,  0.1 st
KiB Mem :  5909372 total,  1107340 free,  2656036 used,  2145996 buff/cache
KiB Swap:        0 total,        0 free,        0 used.  2938912 avail Mem

  PID USER      PR  NI    VIRT    RES    SHR S  %CPU %MEM     TIME+ COMMAND
24049 roms      20   0  510188 168260  11564 R  99.0  2.8   2:33.90 gcc_nwp_no_acc
24050 roms      20   0  510188 168164  11464 R  99.0  2.8   2:33.20 gcc_nwp_no_acc
24044 roms      20   0  509828 168168  11352 R  98.4  2.8   2:32.78 gcc_nwp_no_acc
24046 roms      20   0  508532 167500  11456 R  98.4  2.8   2:33.17 gcc_nwp_no_acc
24047 roms      20   0  509196 167200  11456 R  98.4  2.8   2:32.89 gcc_nwp_no_acc
24045 roms      20   0  510540 168532  11488 R  97.7  2.9   2:32.98 gcc_nwp_no_acc
24048 roms      20   0  507560 165652  11476 R  97.7  2.8   2:32.70 gcc_nwp_no_acc
24052 roms      20   0  526888 188240  11372 R  96.7  3.2   2:29.94 gcc_nwp_no_acc
26738 root      20   0 2239872  97596  64864 S   2.6  1.7  21:26.09 kubelet
 1251 root      20   0 2240112 104356  47416 S   1.6  1.8  14:54.12 dockerd
   54 root      20   0       0      0      0 S   1.0  0.0   0:41.06 ksoftirqd/7
25082 roms      20   0   45504   4228   3536 R   0.7  0.1   0:00.27 top
 1058 root      20   0 2291820  41776  24324 S   0.3  0.7   0:41.89 containerd
    1 root      20   0  225828   9596   6724 S   0.0  0.2   0:08.27 systemd
    2 root      20   0       0      0      0 S   0.0  0.0   0:00.08 kthreadd
    3 root       0 -20       0      0      0 I   0.0  0.0   0:00.00 rcu_gp
    4 root       0 -20       0      0      0 I   0.0  0.0   0:00.00 rcu_par_gp
    6 root       0 -20       0      0      0 I   0.0  0.0   0:00.00 kworker/0:0H-kb
    8 root       0 -20       0      0      0 I   0.0  0.0   0:00.00 mm_percpu_wq
```

(a) Process monitoring inside host



```
top - 03:14:11 up 12:19,  0 users,  load average: 7.54, 2.24, 0.77
Tasks:  17 total,   9 running,   8 sleeping,   0 stopped,   0 zombie
%Cpu(s): 84.5 us, 14.1 sy,  0.0 ni,  0.0 id,  0.0 wa,  0.0 hi,  1.2 si,  0.1 st
KiB Mem :  5909376 total,   876884 free,  1948924 used,  3083568 buff/cache
KiB Swap:        0 total,        0 free,        0 used.  3668700 avail Mem

  PID USER      PR  NI    VIRT    RES    SHR S  %CPU %MEM     TIME+ COMMAND
  127 roms      20   0  506200 164584  10768 R  99.7  2.8   1:08.62 gcc_nwp_no_acc
  124 roms      20   0  508700 166220  10856 R  99.3  2.8   1:08.55 gcc_nwp_no_acc
  129 roms      20   0  510200 167592  10892 R  99.3  2.8   1:08.56 gcc_nwp_no_acc
  132 roms      20   0  508472 166844  10808 R  99.3  2.8   1:07.52 gcc_nwp_no_acc
  123 roms      20   0  524704 185312  14684 R  99.0  3.1   1:01.79 gcc_nwp_no_acc
  125 roms      20   0  507776 166264  10896 R  98.0  2.8   1:08.26 gcc_nwp_no_acc
  128 roms      20   0  510200 167456  10752 R  97.3  2.8   1:07.96 gcc_nwp_no_acc
  126 roms      20   0  506084 164476  10772 R  96.3  2.8   1:08.28 gcc_nwp_no_acc
  150 roms      20   0   36616   3180   2740 R   0.3  0.1   0:00.07 top
    1 root      20   0    4624     64      0 S   0.0  0.0   0:00.01 sh
    7 root      20   0   72296    752      0 S   0.0  0.0   0:00.00 sshd
    8 root      20   0    4528     60      0 S   0.0  0.0   0:00.00 sleep
   52 root      20   0   18504   1548   1096 S   0.0  0.0   0:00.01 bash
   68 root      20   0   60076    896    444 S   0.0  0.0   0:00.00 su
   69 roms      20   0   18624   2668   2196 S   0.0  0.0   0:00.04 bash
  117 roms      20   0  424288  14124  11448 S   0.0  0.2   0:00.14 mpirun
  122 roms      20   0   45184   1052    384 S   0.0  0.0   0:00.03 ssh
```

(b) Process monitoring inside pod

Figure 3.12 a) Process monitoring inside host and b) process monitoring inside pod

3.4. Results and Discussion

3.4.1. Reproducibility

3.4.1.1. Plotting

The graphs of the model output are useful for intuitively evaluating experimental results. Horizontal distributions of temperature and ocean-surface currents from the container clusters were compared. For this purpose, a control simulation was performed in the Kubernetes cluster deployed to local Ubuntu 18.04 servers. Figure 3.13 shows the surface current and sea-surface temperature (SST) from the control simulation after a month. The simulated Kuroshio current showed speed and warmth along the eastern coasts of Taiwan and Japan. Owing to atmospheric cooling and vertical mixing, relatively cold water appeared along the coast of the Yellow Sea (Seo et al., 2014). Output plots from simulations of container clusters in public and private clouds had the same values as those from the control simulation in every grid.

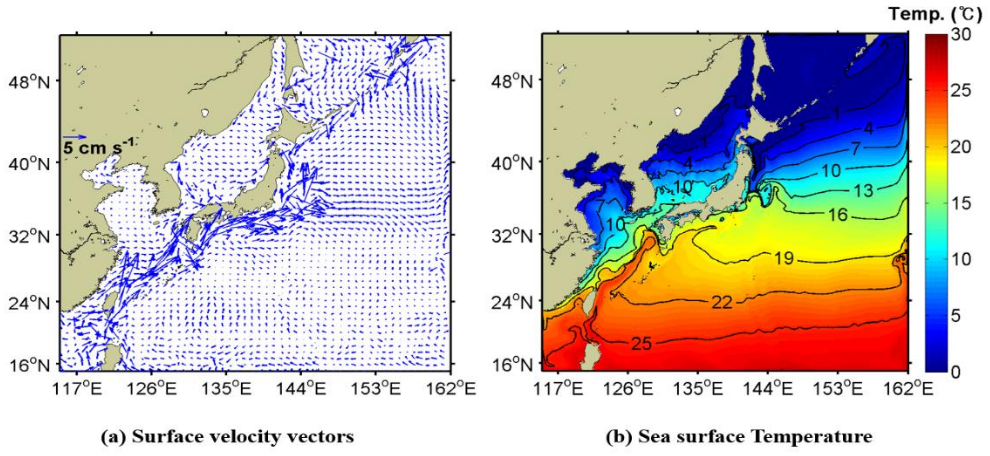


Figure 3.13 a) Surface-velocity vectors and b) sea-surface temperatures from the control simulation of February 1, 2010.

3.4.1.2. Root mean-squared error (RMSE) of containerization for reproducibility

The RMSE was calculated to measure the computational reproducibility of the ROMS modeling. I assumed n observations as y_i and an associated estimator, $\hat{y}_i$. In this study, observations included the ocean-model outputs from the control simulation, and $\hat{y}_i$ captured the outputs of simulations from various container clusters. The RMSE of the SST, vertical temperatures, and u and v vectors between the control simulation result and the simulation results having various cloud clusters were examined. The computational reproducibility was evaluated based on the RMSE. Containerized-ROMS models were deployed to various Linux OSs and associated

HW configurations, and they provided the same results. Tables 3.3 and 3.4 list the configurations of various container clusters. Results were evaluated to analyze whether those of each execution were identical. Comparisons of the control simulation via the simulation of various container clusters in public and private clouds showed that all RMSEs were commonly 0.0 °C for both SST and vertical temperature, and they were 0.0 ms^{-1} along the surface velocity. This result suggests that the container-based architecture might be a suitable computation environment for achieving the needed computational reproducibility for ROMS modeling.

3.4.2. Portability and Performance

3.4.2.1. Portability

Container images, including the OS library, compiler, MPI, NetCDF, and the model binary used for parallel numerical modeling, are downloadable and executable on container clusters. I configured the worker nodes using Kubernetes syntax commands to construct a model cluster. A node registered in the Kubernetes master node was operated as a worker node. Communication between the master and worker nodes was accomplished via the API server and NW topology using an overlay NW (Luksa 2018; Kubernetes Components, 2022).

Table 3.6 Homogeneous and Heterogeneous Cluster Configuration.

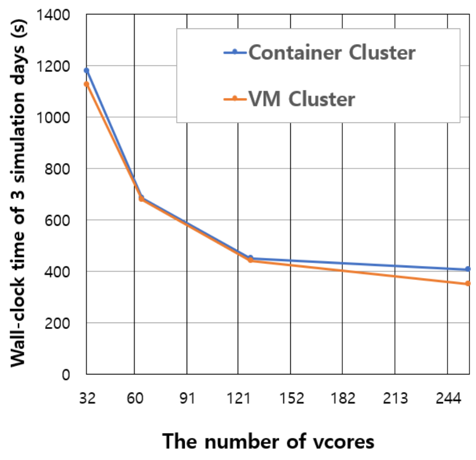
	Homogenous Cluster	Heterogeneous Cluster
HW	Intel Xeon 8124-M CPU 3.00-GHz 36 CPU/72 GB	Intel Xeon 8124-M CPU 3.00-GHz 36 CPU/72 GB, AMD EPYC 7R32/2.8 GHz 32 CPU/32 GB
OS	Ubuntu 18.04	Ubuntu 18.04 + CentOS 7 (7.5)
Container Runtime	Docker v19.03.6	Docker v19.03.6, Docker v19.03.12
Orchestration Tool	Kubernetes v1.18.3	Kubernetes v1.18.3, v1.18.5

HW, SW, and OS configurations of homogenous and heterogeneous container clusters are shown in Table 3.6. After configuring the master node, I installed Kubernetes SW on the worker nodes and joined them to the master node. ROMS was automatically configured for modeling the environment after downloading it from a repository where pre−built images are stored. RMSE was analyzed using the results from the heterogeneous OS worker nodes and those of the homogenous OS clusters. The calculated RMSEs were close to zero. Container−based clusters were suitable for the reproducibility of the model for various OSs. The container cluster enabled us to use various server resources for modeling with little effort.

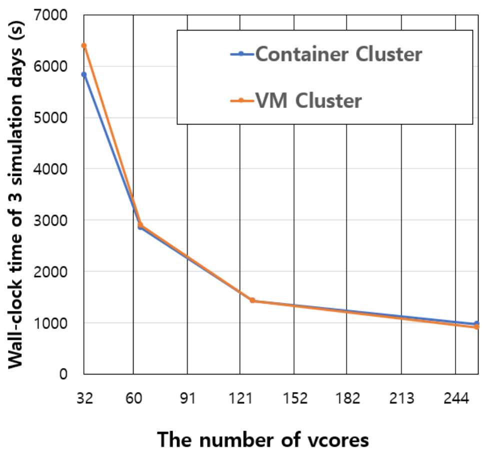
3.4.2.2. Performance

Running time is a clear indicator of performance. Thus, for our performance evaluations, I compared the running times of ROMS models according to grid sizes and cores on the various clusters of the public and private clouds. I also examined the throughput using the STREAM benchmark on various clusters. Notably, when multiple nodes are used, the memory I/O performance among multiple nodes becomes important (McCalpin, 2017).

To compare the performance of the container-based cluster with the VM cluster, I compiled and set up ROMS on the VM clusters using the same HW and SW configurations as the private and public clouds. Compared with the performance of the ROMS on the VM clusters, Figure 3.14 shows that the container-based cluster had a performance penalty between 1 and 9% for four nodes (128 vcores) in the AWS cluster. However, when I reached eight nodes (256 vcores), the performance penalty changed to 7%–14% because of the inter-pod NW latency.



(a) Wall-clock time of medium grid simulation



(b) Wall-clock time of fine grid simulation

Figure 3.14 (a) Wall−clock running time of 3−day simulation of a medium grid, and (b) wall−clock running time of 3−day simulation of a fine−grid according to the vcores on the AWS clusters.

I then measured the latency of the VM and pod using the Ohio State University SW micro benchmark (UL HPC Team, 2021). Figure 3.15 shows the NW latency of the pod NW and the VM in ethernet−based MPI clusters. The NW latency of the pod was slightly larger than that of the VM. There was a small performance penalty in the HPC container cluster. However, considering the preparation time and portability of the cluster, the container−based clusters provided an important and alternative space in which to run numerical models with computational reproducibility and portability in private and public clouds. In virtualization environments that support RDMA, it was possible to reduce NW latency (Zhuang et al., 2020; Beltre et al., 2019).

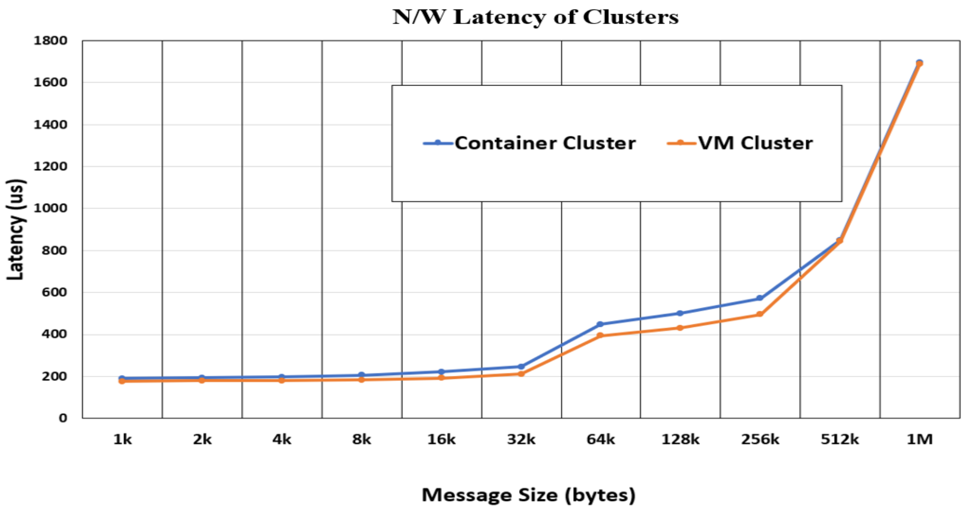


Figure 3.15 Comparison of NW latency of container and VM cluster according to message sizes.

Memory bandwidth can be crucial to speed computations, and numerical ocean modeling requires a large−memory I/O in the MPI environment. I evaluated the memory bandwidth in a VM and container cluster environment using the STREAM benchmark. The memory bandwidth was measured at multiple nodes in parallel with the MPI. The memory bandwidth of the container cluster was slightly less than that of the VM cluster. The performance patterns of the running time were similar in the container and VM clusters. Figure 3.16 shows the memory bandwidth of the container and VM cluster.

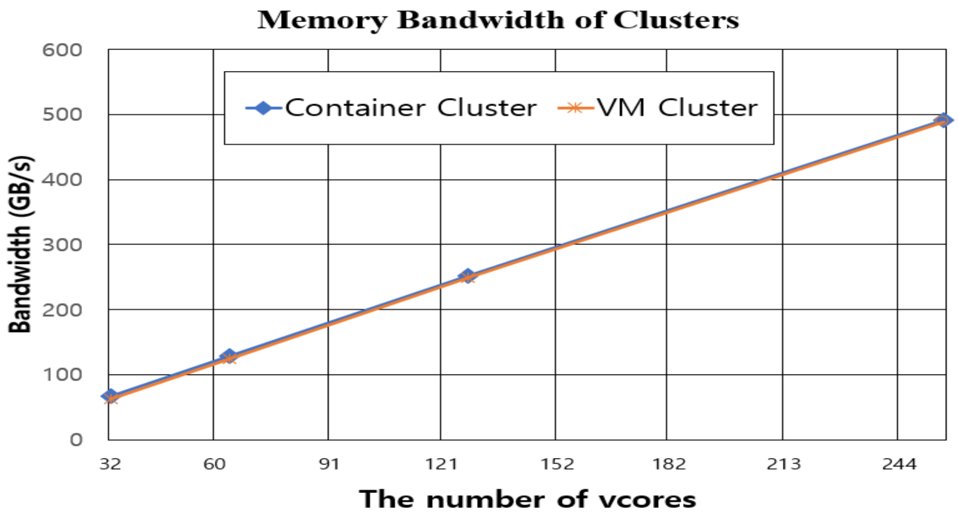


Figure 3.16 Memory bandwidth of the container and VM cluster vs. the number of AWS vcores.

3.5. Conclusions

In this study, an architecture for numerical models was designed and applied based on light-weight container virtualization and container orchestration technology to improve the computational reproducibility and portability of numerical models in various public and private infrastructure environments. The ROMS model container image was registered in the standard repository for user convenience in the container runtime environment. The infrastructure configuration code was executed using Kubernetes by applying the parallel processing of numerical containers in various cloud environments. VMs and the container virtualization environments were managed via codes, owing to the development of cloud-related virtualization techniques. ROMS models having different grid sizes were implemented in cloud clusters having 32, 64, 128, and 256 cores. Model results from various clusters based on containers were the same as those from the control model, regardless of OS and HW environments. The container-based simulation results coincided with those of the control simulation of the SST, vertical temperature profile, and surface velocity. This suggests that a container-based cluster is appropriate for use in the computational reproducibility of the numerical ocean model.

The container-based architecture makes numerical ocean modeling much easier than does VM-based architecture in a variety of private and public cloud environments, because the cloud infrastructure environments are abstracted to make it easier for researchers to share numerical model environments. After executing a simple model on a PC using modifying grid sizes and iteration numbers, large-scaled modeling was carried out in the HPC cluster of private and public clouds by applying the same setup codes and modifications. The procedure for preparing the numerical modeling was coded and shared in the public repository. Additionally, the proposed container-based numerical model-cluster architecture made it easy to overcome the heterogeneity of NW drivers and HW limitations, compared with the traditional architecture.

This container−based architecture can help researchers perform numerical modeling more easily in various public and private cloud environments and improve model reproducibility and portability in geoscientific research. This is especially useful for researchers who lack appropriate IT infrastructures.

4. Generative models for the prediction of ocean temperature profile[③]

4.1. Introduction

It is estimated that the ocean stores 93% of the world's energy, and the redistribution and exchange of the subsurface to the deep sea plays a significant role in global warming (Wang et al., 2021). Subsurface temperature data in the ocean aids in the interpretation of physical properties associated with ocean physical motion, which is also useful for military submarine positioning and obtaining fishery distribution data (Schmidt et al., 2019).

The development of satellite and sensor technology enables us to easily obtain sea surface information, but there are limitations to directly obtaining subsurface information. Subsurface data are sparse, whereas many satellites routinely collect sea surface temperature (SST) and sea surface height (SSH). Recently, many researchers have attempted to estimate the ocean’s vertical temperature profile using statistical or gen techniques (Jiyang et al., 2017; Wang et al., 2021). Some studies have been conducted to estimate the subsurface

[③] The results of the presented work have been submitted into 『*Frontiers in Marine Science*』, (2022).

temperature distribution of sea water vertically using satellite and Argo Float data (Han et al., 2019). Argo floats also have limitations in the precision or measurement of specific areas, such as near coasts and marginal seas, because the floats are not uniformly distributed (Roemmich et al., 2019).

Machine learning techniques such as convolutional neural networks (CNN) and recurrent neural networks (RNN) have recently been used to predict the vertical temperature profile using surface information such as SST and SSH (Han et al., 2019). Prior studies have made significant contributions, but there are some limitations. The depth of data collection may limit the predicted value at a given depth. The measured locations in Argo-float were sparse and not fixed (Roemmich et al., 2019). Stationary data collection has the advantage of obtaining uniform data from the same location. Creating a model suitable for predicting ocean physical properties requires the preparation of sufficient datasets for model training. Although machine learning-based prediction models perform well in open oceans such as the Pacific Ocean, they are limitedly used in marginal seas because of the large spatiotemporal variability in the temperature and current.

The objective of this study was to improve the prediction model's performance using data augmentation for training in a marginal sea with large temperature variations in time and space. I apply generative models such as the generative adversarial network (GAN) method and triplet variational auto-encoder (TVAE) to augment the observational datasets and trained augmented datasets for the subsurface temperature profile. The TVAE and conditional generative adversarial network (CGAN) methods, among others, were used to create artificial datasets that were used to improve the model's performance.

The remainder of this paper is organized as follows. Section 4.2 introduces the study area and methods, such as generative models for the data augmentation and stacking ensemble method for the prediction model, and explains the model architecture for predicting the sea subsurface temperature profile and its implementation in deep neural network environments. Section 4.3 shows that the results of the ensemble prediction model based on the generated dataset are meaningful, and the feasibility of data augmentation using generative models such as TVAE and GAN in ocean science. Finally, in Section 4.4, I summarize the achievements of our ensemble prediction model architecture and the data augmentation architecture based on generative models. I also make recommendations for future improvements to our architecture for geo-scientific applications and extensions.

4.2. Materials and Methods

4.2.1. Model domain and datasets for predicting the subsurface temperature

The Tsushima current (TC) supplies heat and salt to the East/Japan sea (EJS) (Preller and Hogan, 1998). The TC is divided into two branches: one along the Japanese coast and the other along the Korean coast (Figure 4.1) (Cho and Kim, 2000). This flow along the Korean coast is called the East Korean warm current (EKWC) (Cho and Kim, 1996; Kim et al., 2018). The EKWC turns eastward around Ulleung Island, forming the Ulleung warm Eddy (UWE) (Kang and Kang, 1990; Kim et al., 1991; Katoh, 1994). The UWE, with a diameter of approximately 150 km, is located in the Ulleung basin (Figure 1). The size and location of the UWE varies seasonally and interannually (Kang and Kang, 1990; Isoda and Saitoh,1993; Choi, 2004). The UWE plays a key ecological role in supporting a significant phytoplankton biomass (Kim et al., 2012). A station routinely observed by the National Institute of Fisheries Sciences (NIFS), which is located in the UWE, was selected for the generation of the sea subsurface temperature profile at our study point (37.06°N, 130.31°E, red circle in Figure 4.1). The fluctuating characteristics of the temperature profile in the EJS render it suitable for testing the prediction performance of the proposed temperature profile model. Compared to the open sea, it is challenging to predict temperature profiles in marginal seas, such as the EJS, because many complex dynamic processes cause large variations in temperature and current.

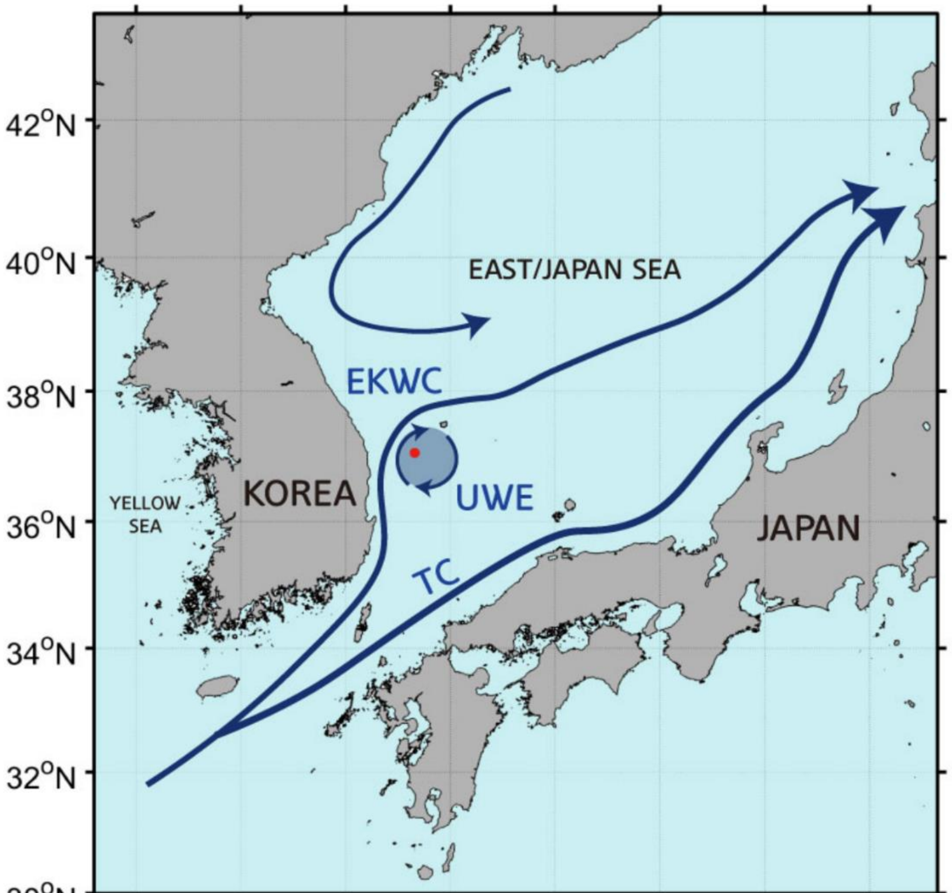


Figure 4.1 Schematic currents in the study area and model domain. The red point (37.06° N, 130.31° E) represents a routine observation station. Selected station for comparing the model and observation temperature profiles. TC, EKWC and UWE stand for Tsushima current, East Korean warm current and Ulleung warm eddy, respectively.

The model has 14 vertical layers. The subsurface temperature profile was predicted using the NIFS’s serial oceanographic observation dataset. The SST dataset for the research domain was extracted from the advanced very high-resolution radiometer (AVHRR) instrument. The AVHRR has a spatial grid resolution of approximately 0.25°, and the temporal resolution is 1day. These datasets were downloaded from the National Center for Environmental Information (NOAA, 2022). The sea surface temperature data among the NIFS observation data sets were also used as auxiliary data to prepare the SST data. I used the Copernicus marine environment monitoring service (CMEMS) gridded dataset for daily sea-level data (Copernicus, 2022). The horizontal resolution was 0.25°. The datasets are sea-level daily gridded data from satellite observations for the global ocean from 1993 to 2017. I used a dataset for the prediction model, which included absolute dynamic topology (ADT) and sea level anomaly (SLA) from daily sea level data. I used data from 1993 to 2017 when both SST and SSH satellite data were available. The dataset was downloaded from the CMEMS climate data store (Copernicus, 2022). Training data were created using datasets from 1993 to 2012 as seed data. The model's performance was measured using test data from 2013 to 2017.

I created a dataset for GAN seed data by combining satellite data on the date when the measured temperature was present for each reference depth of the observation point. The data were removed without artificial interpolation when missing temperature values at the corresponding depths were found. Datasets were generated using only data when all observation data existed at the reference depth on the corresponding date.

4.2.2. Model architecture for predicting the subsurface temperature

In this study, I intend to create a machine learning model that predicts the subsurface temperature profile by combining satellite datasets such as SSH and SST with locally measured in situ temperature profile data. Figure 4.2 shows the conceptual architecture for predicting the sea subsurface temperature profile. To augment the sparse temperature profile dataset, I experimented with some types of generative methods, such as CTGAN, CopulaGAN, and TVAE (Xu et al., 2019). The observed datasets in Earth science are mainly tabular-type datasets, and continuous columns can have multiple modes. Some observed datasets may have non-Gaussian distributions, which are sometimes multimodal. Owing to these characteristics, there may be challenges in tabular data augmentation tasks using GANs (Xu et al., 2019). Much research has been conducted to overcome these challenges, and I applied generative methods such as CTGAN, TVAE, and copular GAN based on related research in this study.

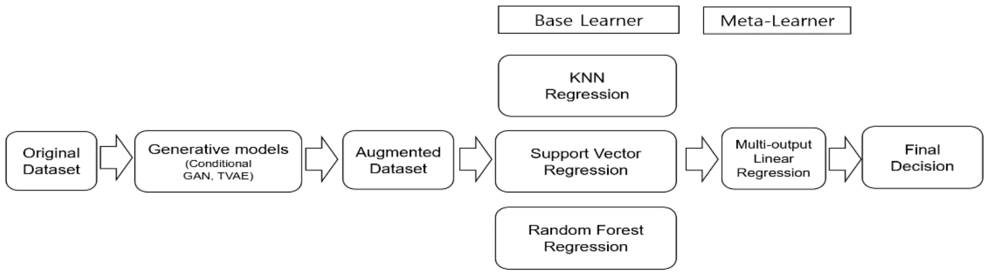


Figure 4.2 Conceptual architecture for predicting subsurface temperature

4.2.3. Neural network generative models

4.2.3.1. TVAE

Triplet-based variational autoencoders (TVAEs) are enhanced types of variational autoencoders (Ishfaq et al., 2018) that can learn latent representations with more fine-grained information.

Figure 4.3 shows an example of latent representation, which is a key feature of the input data. The key features of dogs and cats are their ears and eyes. The latent representation is the sum of the latent features. The autoencoder, the middle layer of this network, contains a simplified representation of the input data and can be used to reconstruct the output.

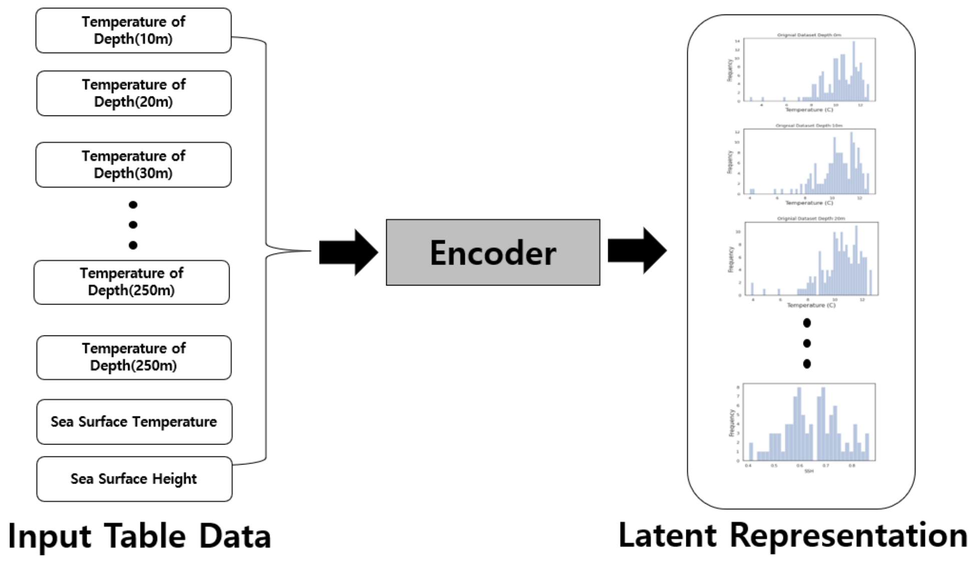

Temperature of Depth(10m)
Temperature of Depth(20m)
Temperature of Depth(30m)
Temperature of Depth(250m)
Temperature of Depth(250m)
Sea Surface Temperature
Sea Surface Height
Encoder
Input Table Data
Latent Representation

Figure 4.3 Conceptional architecture of latent representation

TVAEs can learn an interpretable latent representation that preserves the original dataset's semantic structure by incorporating triplet constraints into the learning process. In each iteration of training, the input triplet is randomly sampled from the training dataset. Then, the triplet of images or data is flown into the encoder network simultaneously to obtain their mean latent embedding (Ishfaq et al., 2018). A loss function over triplets to model the similarity structure over the image or data can be defined, as in Wang et al. (2014). Embedding, a method used to represent discrete variables as continuous vectors, is the process of converting high-dimensional data into low-dimensional data in the form of a vector such that the two are semantically similar (Jeevanandam, 2021).

4.2.3.2. Generative Adversarial Networks

The generative adversarial network (GAN) is a machine−learning method proposed by Ian Goodfellow (Goodfellow et al., 2014). The core idea is that one generator is trained to generate fake data and the other (discriminator) is trained to distinguish between real and fake samples. The goal of training a generative network is to improve the discriminant network's error rate. The generative network generates fake or candidate data, whereas the discriminative network evaluates them. In terms of data distribution, they compete with each other. A conceptual diagram was shown (Figure 4.4). A GAN consists of two networks: a generator (G) and discriminator (D).

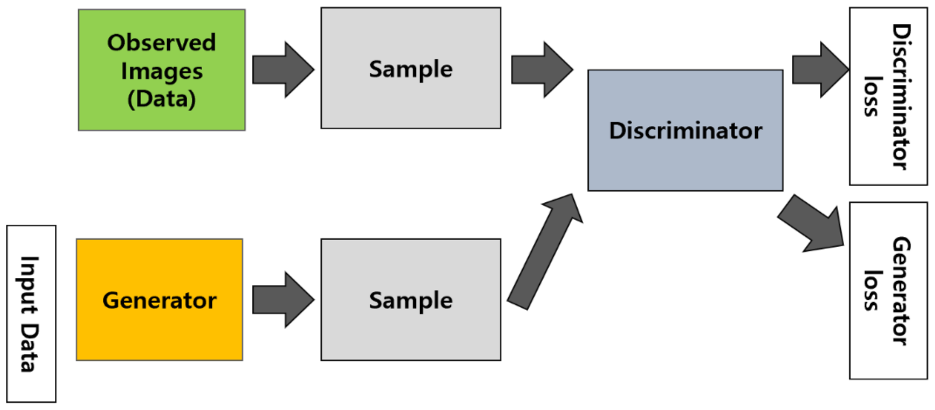


Figure 4.4 Conceptional architecture of generative adversary network

Both networks had their own loss functions. The loss function of GAN is given below, and it is similar to the min−max problem (Goodfellow et al., 2014).

$$minmax\, V(D,G) = \mathbb{E}_{x \sim P_{data}(x)}[\log D(x)] + \mathbb{E}_{z \sim P_z(z)}[\log(1 - D(G(z)))]$$

The definitions of the terms used are follows.

Term	Definition
G	Generator model
D	Discriminator model
z	Random noise
x	Real data
G(z)	Data generated by Generator (synthetic data)
$p_{data}(x)$	Probability distribution of real data
$p_z(z)$	Probability distribution of synthetic data
D(G(z))	Discriminator' s output when the generated data is an input
D(x)	Discriminator' s output when the real data is an input

4.2.3.3. Conditional GAN

If I can determine the type of data to be generated through a GAN, GANs can be used for many scientific applications. When I suppose both the generator and discriminator having a condition of some supplementary or auxiliary information y, GANs can be extended to a conditional model. Furthermore, y could be various types of supplementary information, such as class labels or different types of data. I can perform conditioning by inputting y into both the generator and discriminator as an additional input layer. The joint hidden representation in the generator combines the prior input noise $P_z(z)$ and y, and the adversarial training framework allows considerable flexibility in how this hidden representation is composed. In the discriminator, x and y are presented as inputs to a discriminative function (embodied again by a multilayer perceptron in this case (Mirza and Osindero, 2014). Figure 4.5 shows the conceptual structure of a basic conditional adversarial network. The generator synthesizes a fake sample (G (z, y) = x*|y) using a random noise vector z and label y. Given the label, the fake sample's goal is to resemble the real sample as closely as possible. The discriminator takes a real sample and a label (x, y), as well as a fake sample and the label used to generate it (x*|y, y) (Langr and Bok, 2019). The discriminator learns to distinguish between real data and matching pairs from real sample−label pairs, and how to identify fake data−label pairs from a generator's sample. The discriminator outputs a single probability that the input pair is real data, and computes it using the activation function sigma of the sigmoid.

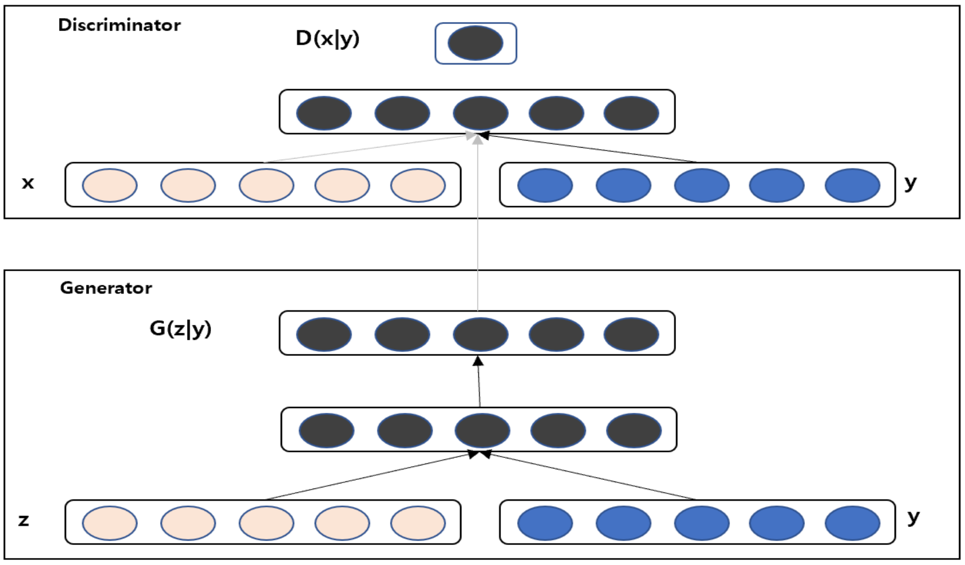


Figure 4.5 Conceptional architecture of a conditional generative adversary network

The CTGAN is a GAN−based method for generating tabular data using the data distribution of a tabular sample dataset (Xu et al., 2019). CopulaGAN is a CTGAN model variant that uses a cumulative distribution function−based transformation (synthetic data vault (SVD), 2022). The dataset of the sub−surface temperature profile is generally tabular data, and the temperature at each depth is not linearly dependent on the observed depth. TVAE with a variational autoencoder was used to generate datasets with high performance and flexibility (Xu et al., 2019). I can prepare training datasets for the prediction model and apply them to train the prediction models, which are then used for the ensemble model, using the proposed augmentation architectures based on generative models. Enhancing model training is possible after artificially augmenting meaningful datasets. The models used in this study were developed in Python using the Tensorflow−based Keras library and pytorch−based SDV libraries (MIT Data To AI Lab, 2022). Generative model codes (that is, some types of generative adversarial networks and TVAE used to generate sample data) were deployed on Python Jupyter notebooks on AMD 10 cores and Nvidia RTX−3090. Augmented datasets were used for several base models to construct an ensemble model for the prediction of sea subsurface temperature profiles. Generally, the ensemble model is more accurate than the single model at predicting values. I designed and implemented ensemble stacking methods using several candidates to improve performance.

4.2.4. Prediction Models

In the stacking method for the ensemble, I chose the K−nearest neighbors regression (KNNR) model, support vector regression (SVR), and random forest regression (RFR) as base learners and the multioutput linear regression (LR) model as the meta−learner in our study. Every base learner generates the predicted values based on their own algorithm, and they are used as datasets for the meta−learner.

4.2.4.1. Stacking Ensemble

In statistics and machine learning, ensemble methods use multiple learning algorithms to obtain better predictive performance than any of the constituent learning algorithms alone (Zhang and Ma, 2012). In ensemble learning, three major methods aim to combine base models or weak learners. Bagging and boosting learn homogeneous base models and combine them using a deterministic strategy or process (Rocca, 2019). Stacking learns heterogeneous weak learners in parallel and combines them by training a metamodel to output a prediction based on different base model predictions. Ensemble stacking, or stacked generalization, involves training a learning algorithm to combine the predictions of several other learning algorithms (Brownlee, 2021; Kadkhodaei, 2020; Rocca, 2019). First, all other algorithms are trained using the available data. And combiner algorithm is trained to make a final prediction with all the prediction outputs of the other algorithms as additional inputs. Stacking typically outperforms any single trained model. Figure 4.6 shows the conceptual architecture of the stacking ensemble. In this study, I will use the ensemble stacking method to combine weak learners that stand out in individual models to build a model with better performance. Regression models, which are traditionally used to estimate numerical values, are chosen as base models, such as the KNNR, SVR, and RFR.

Furthermore, I intend to construct a metamodel using a multioutput linear regression model that performs well in multiple predictions. Multioutput regression is a regression problem that involves predicting two or more numerical values given an input example (Brownlee, 2021). In this study, I use a multioutput regressor to predict multiple subsurface temperatures by depth using SSH and surface temperature.

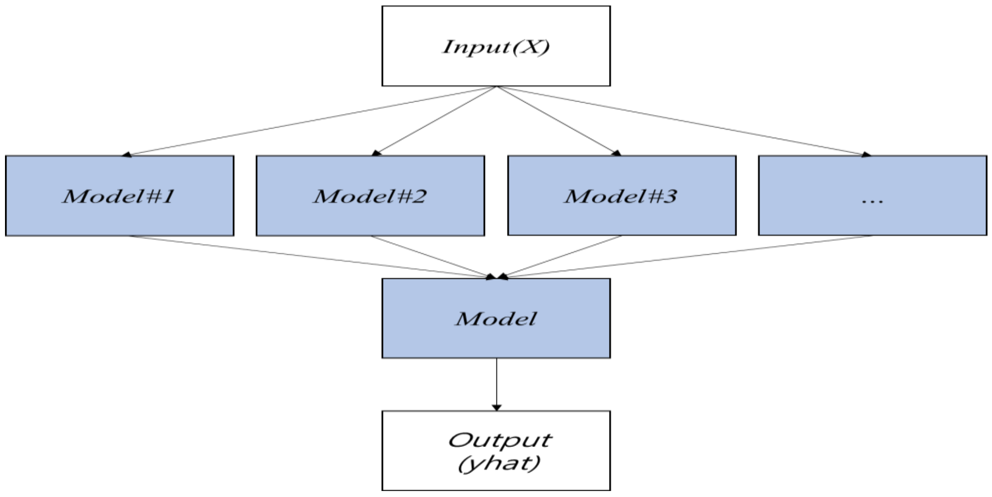


Figure 4.6 Conceptual diagram of the stacking ensemble

4.2.4.2. K−Nearest Neighbors Regression

The k−nearest neighbor (k−NN) algorithm is a non−parametric supervised learning method used for classification and regression (Atteia et al., 2019). The K−NN regression output is the property value for the object, and the value is the average of the values of the k−nearest neighbors. K−NN estimates the association between the input and response variables using feature similarity (Yao et al., 2006). In the k−NN regression, the response variables are approximated by averaging the observations in the nearest neighborhood of the input instance using similarity measures (Ali et al., 2019).

Guo et al. (2018) used KNNR to refine the existing datasets for thermocline research, and Li et al. (2019) used KNNR to develop a method for constructing high−resolution ocean models and found that the proposed KNNR model was used to refine seawater thermocline data and improve the data resolution on their vertical gradient.

I chose KNNR as the base learner because of its approximation performance and recent thermocline research cases using KNNR.

4.2.4.3. Support Vector Regression

SVR is an extended algorithm of the support vector machine (SVM), which is a classic and powerful machine learning algorithm for solving nonlinear regression problems (Brereton and Lloyd, 2010). SVR calculates the loss function based on structural risk minimization, allowing a deviation of ε between the model output and the real value. This differs from the traditional regression model, which is based on the error between the model output and the real output. This can avoid the disadvantages caused by pursuing experiential risk minimization. SVR is a model that uses high−dimensional feature spaces but penalizes the resulting complexity using a penalty term augmented with the error function, making it suitable for fitting high−dimensional data with comparatively fewer samples (Balogun et al., 2021). The basic idea of SVM is to map multi−dimensional data onto a higher−dimensional feature space. And there is a hyperplane that linearly separates the original data while maximizing the margin between different classes (Burges, 1998). Through SVM, the sub−sea surface temperate anomaly in the Indian Ocean could be estimated from satellite measurements of sea surface parameters (SSTA, SSHA, and SSSA as input attributes for SVM) (Hua et al., 2015). Li et al. (2017) evaluated the performance of a support vector machine–complementary ensemble empirical mode decomposition model to estimate SST in the northeast Pacific Ocean. In another study, Jiang et al. (2018) evaluated LR and SVR prediction performance of SST in the Canadian Berkley Canyon. Water depth and coordinate information such as latitude and longitude were used as input variables. These input variables have seldom been used to assess SST in previous studies. The results showed that SVR provided estimates closer to the observed data than LR.

4.2.4.4. Random Forest Regression

In this study, the RFR was chosen as the base learner to create the ensemble model. RFR creates robust estimates using an ensemble of decision trees, frequently without requiring data pre−processing, making it an effective "off the shelf" method (Louppe, 2014).

Decision trees are useful for determining nonlinear relationships between the target variables and input features (Auret et al., 2012). Gregor et al. (2017) used SVR and RFR to estimate CO_2 levels in the Southern Ocean and achieved good prediction performance. A random forest is also a meta-estimator that fits several classifying decision trees on various subsamples of the dataset and uses averaging to improve the predictive accuracy and control overfitting (scikit-learn.org, 2022). I chose the RFR for reasons such as nonlinear relationships and predictive accuracy.

4.2.4.5. Linear Regression

Traditionally, linear regression analysis has been widely used in various Earth science fields. Linear models have been widely used in ocean prediction because they require minimal data input and are relatively simple. Although simple, it is relatively effective in identifying trends and provides important insights for understanding and analyzing overall trends. It is widely used in ocean science to predict water temperature distributions and analyze trends. Many scientists use linear regression models (Morrill et al., 2005; Krider et al., 2013) in ocean sciences. Feng et al. (2020) developed a multiple linear regression algorithm for sea surface temperature retrieval using one−dimensional synthetic−aperture microwave radiometry. The regression method is a strong candidate for determining the relationships among a variety of properties, such as sea surface temperature, sea surface height, and depth. I also need to analyze the correlation between sea surface temperature and sea surface height and depth. Leuliette and MWahr studied coupled pattern analysis of sea surface temperature and TOPEX/Poseidon sea surface height (Leuliette et al., 1999). They showed that the spatial correlation is strong in both the Atlantic and Pacific. The good temporal and spatial agreement between the SSH and SST fields suggests that a robust regression between fields may have some physical significance. With reference to the results of previous studies and the robust of the model, I choose linear regression, one of most common statistical methods as a member for ensemble in many oceanic analyses.

4.2.5. Accuracy

For the performance evaluation of the machine learning models, I applied the commonly used metrics: mean absolute error (MAE) and root mean square error (RMSE). The RMSE evaluates the residual between the observed and predicted values and is particularly sensitive to large errors. The MAE is less sensitive to extreme values than the RMSE (Ait−Amir et al., 2015).

The mathematical formulas are as follows:

$$RMSE = \sqrt{\frac{1}{N}\sum_{i=1}^{N}(y_{o,i} - y_{p,i})^2}$$

$$MAE = \frac{1}{N}\sum_{i=1}^{N}|y_{o,i} - y_{p,i}|$$

where $y_{p,i}$ and $y_{o,i}$ are the predicted and observed values in the dataset, respectively, i is the sample number of the dataset, and N is the length (number of samples in the test set).

4.3. Results and Discussion

4.3.1. Data Generation

High−quality datasets are critical for the prediction model's performance in machine−learning approaches. Real and observed datasets may be costly and challenging to measure and acquire. In this study, generative models were used to generate subsurface temperature datasets, which are difficult to obtain, and the similarity was determined by comparing synthetic and observed data distributions (Figures 4.7 and 4.8).

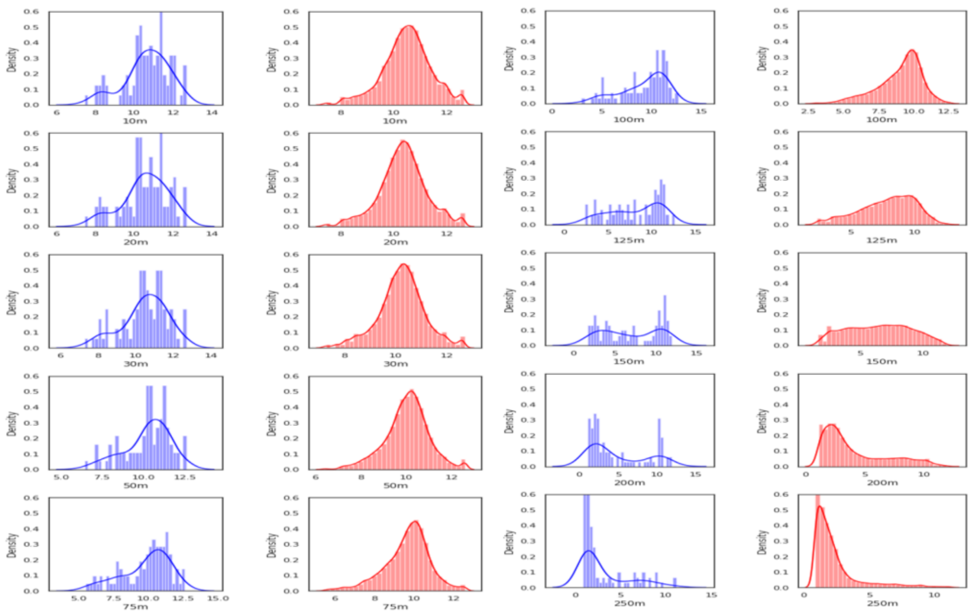


Figure 4.7 Data distributions of observed and synthetic datasets in FEB from 1993 to 2012

In this study, I attempted to generate subsurface profile data using TVAE and GANs and visualized the histogram and difference matrix of observed and synthetic datasets. Figures 4.7 and 4.8 show the density histogram of the synthetic data according to depth, and these histograms provide information on the similarity between synthetic and observed datasets. The difference matrix of the observed and synthetic datasets shows the similarities and differences between them (Figures 4.9 and 4.10).

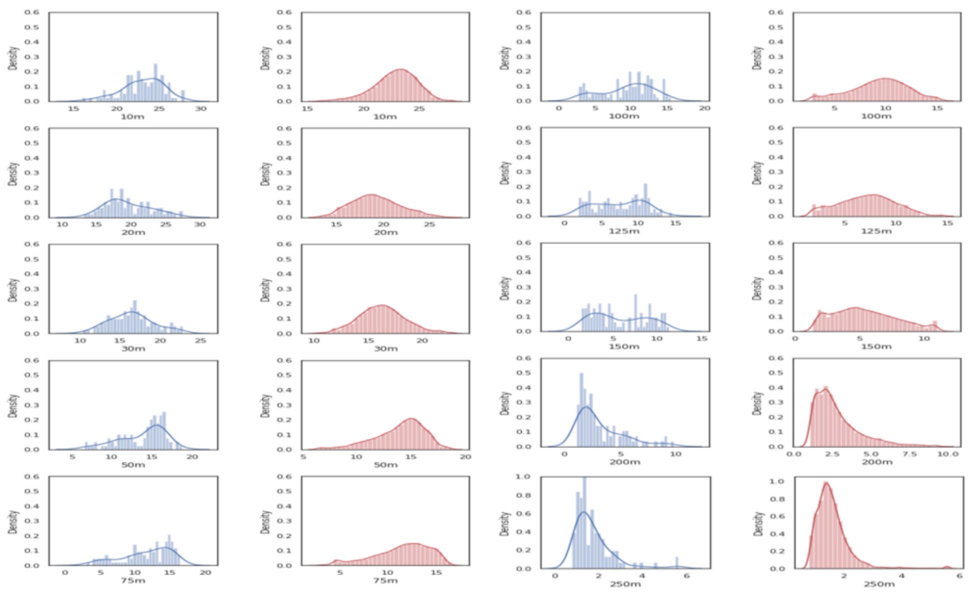


Figure 4.8 Data distributions of observed and synthetic datasets in AUG from 1993 to 2012

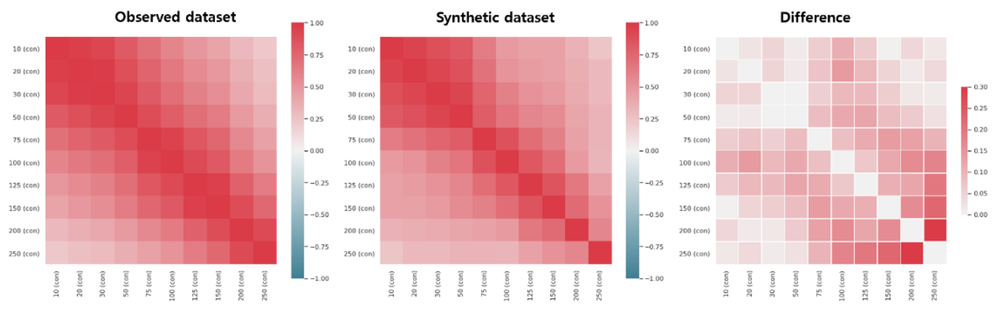


Figure 4.9 Difference matrix of observed and synthetic datasets in FEB from 1993 to 2012

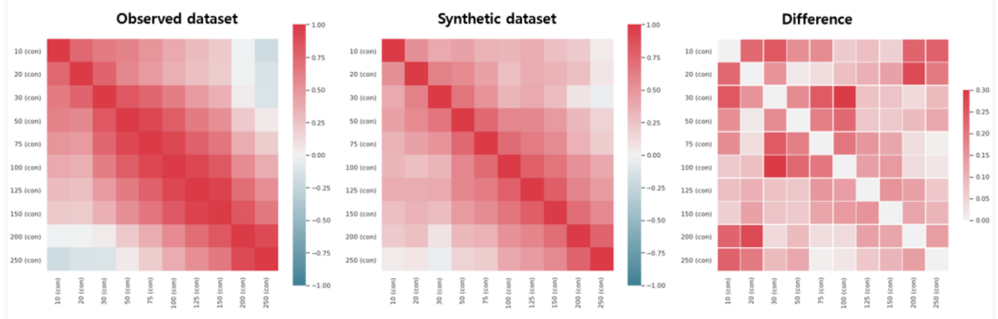


Figure 4.10 Difference matrix of observed and synthetic datasets in AUG from 1993 to 2012

In this study, I compared whether the data were synthesized by inputting summer and winter datasets into a generative model, reflecting the distribution of the changed data. The August data generation also shows a similar distribution between the observed and synthetic data, and the difference matrix indicates that the gap between the observed and synthetic data is small, and they are similar.

In this study, I compared the accuracy metrics of the candidate models, such as K-nearest neighborhood, SVR, random forest, and linear regression for selecting ensemble model members. Several models were used for the ensemble model, and I chose the base learners for the stacking ensemble based on previous studies and accuracy metrics such as the MAE and RMSE values. I chose an observation point close to the UWE and evaluated the MAE and RMSE of the base-learner models using the observed and synthetic datasets.

In Figure 4.11, the MAE and RMSE of regression models such as KNNR, SVR, RFR, and LR using the observed and augmented datasets are shown. The MAEs and RMSEs of the models using the observed dataset were higher than those of the models using the synthetic dataset. This means that the accuracy of the prediction model using the synthetic dataset is better than that of the observed dataset in this study. When I compared the MAE and RMSE of the individual models to those of the stacking ensemble prediction in Table 4.1, the accuracy metrics of the stacking ensemble prediction were better.

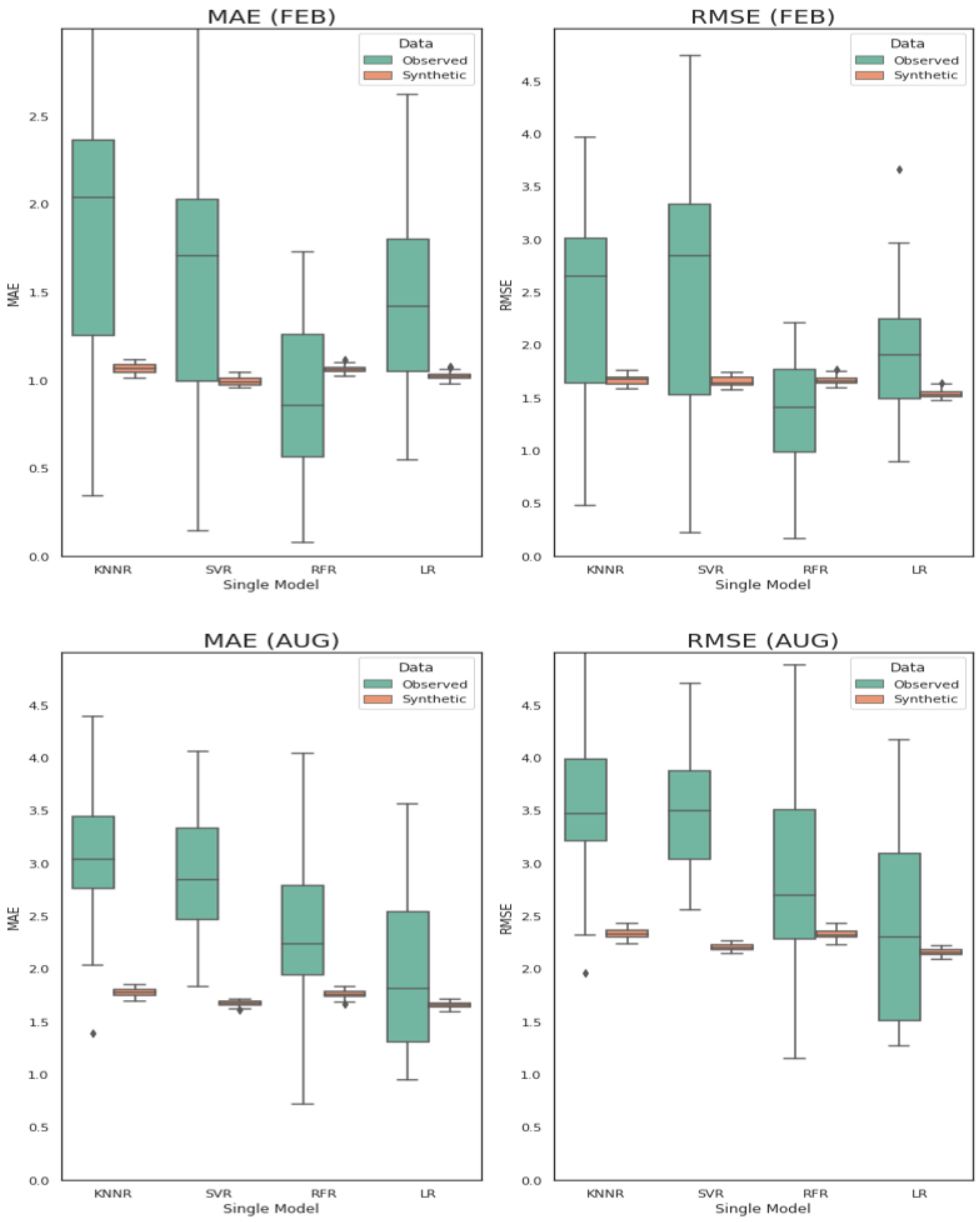


Figure 4.11 MAE and RMSE of regression models (KNNR, SVR, RFR, LR) using FEB and AUG datasets

4.3.2. Ensemble Prediction

Figure 4.12 shows the prediction results of the stacking ensemble model. In this study, data were synthesized using data from a station in the UWE from 1993 to 2012 and used as training data for the model. Then, using the data for five years from 2013 to 2017 as test data, I measured the stacking ensemble's model prediction performance. Data from February for winter and August for summer were used to compare temperature profiles during seasonal changes.

As shown in the observed temperature profiles, the UWE, which can be characterized by homogeneous water from the surface to about 200 m depth, appears in winter. However, the UWE mostly disappeared, and a strong thermocline appeared in summer. The model results accurately predicted the seasonal change in the temperature profile over the entire period, except for August 2015, when the remnant of the UWE was present in the subsurface.

Table 4.1 shows the accuracy metrics for the ensemble prediction results. The synthetic dataset model outperforms the observed dataset model.

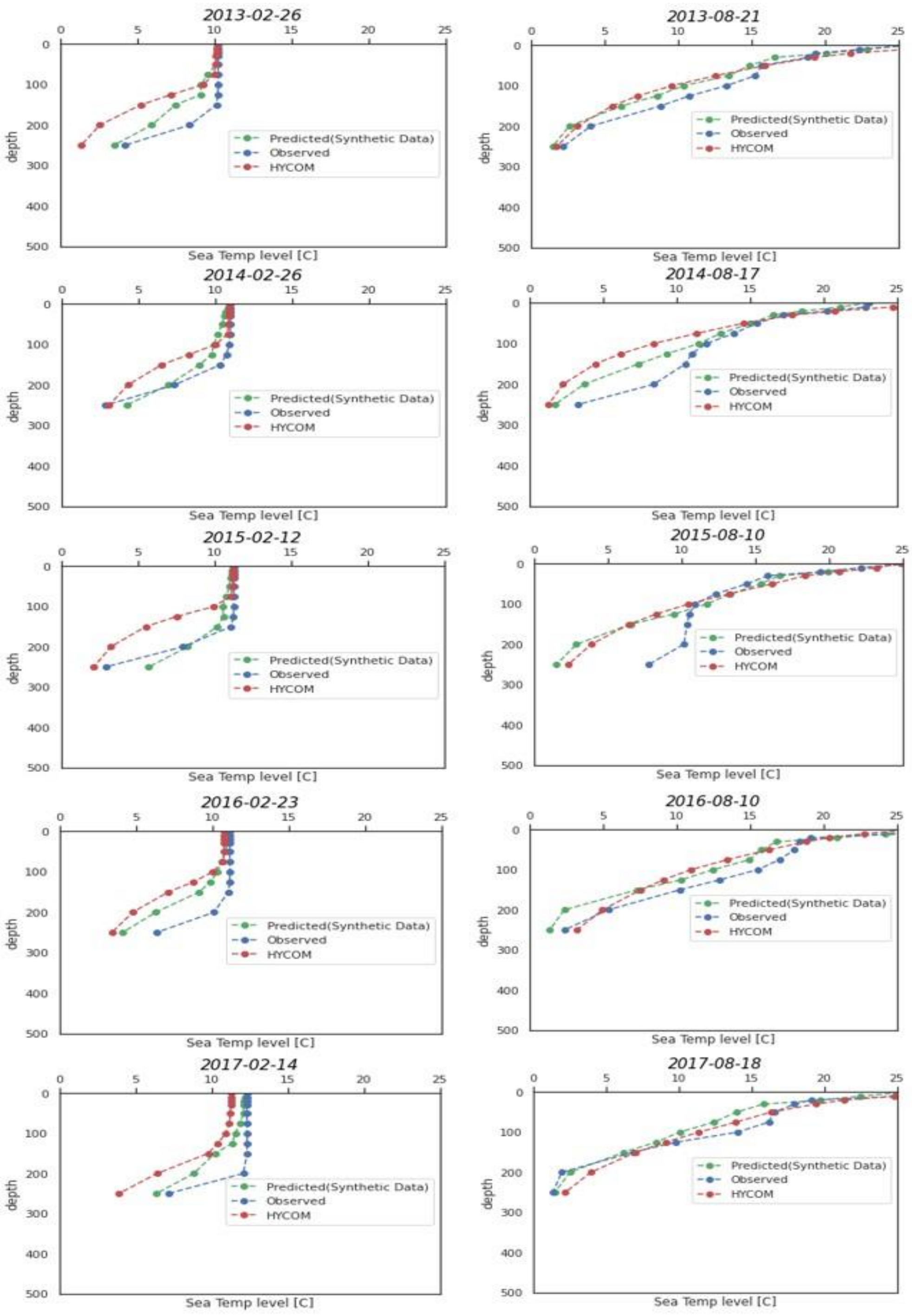


Figure 4.12 Comparison of the predicted temperature profile using the stacking ensemble prediction model with augmented dataset, HYCOM and the observation data in February and August from 2013 to 2017.

Table 4.1 Ensemble prediction accuracy metrics of synthetic dataset (FEB, AUG)

Dataset type	Synthetic Dataset (FEB)		Synthetic Dataset (AUG)	
Metrics	MAE	RMSE	MAE	RMSE
Accuracy	0.96	1.20	1.92	2.45

4.3.3. Limitations of this study and future works

This study was conducted focusing on specific observation point to check the seasonal occurrence of UWE. For the spatial expansion of this study, it is necessary to extend the three-dimensional appearance of UWE through data generation and analysis of multiple observation points where the observation data of NIFS exists. In addition, data synthesis, model training, and analysis were executed after 1993, when satellite data existed. In order to generate SLA data before 1992, it is necessary to consider the development of the generative model and prediction model and whether it is possible to synthesize SLA data of the past by using the temperature profile data as input data and training with SLA data sets as the output.

4.4. Conclusion

In this study, the augmentation architecture was successfully adopted as a generative model for the subsurface temperature profile data in a marginal sea. The GAN can also be a suitable method for tabular and non−Gaussian data distribution datasets. To train a model that predicts the subsurface temperature profile in the marginal sea, the observed dataset from 1993 to 2012 was used to augment and train the data. The accuracy metrics of the prediction model, the MAE was 0.96 and 1.92 and the RMSE was 1.20 and 2.45 in February and August, respectively. The augmented dataset improved model prediction performance. The GAN−based architecture improved and increased the real dataset for the prediction model accuracy, and a candidate served as a data imputation solution for missing values. The copular GAN model, which considers the correlation of variables, and TVAE are suitable for subsurface profile data synthesis.

A stacking ensemble method that combines heterogeneous models with excellent performance in the respective areas achieve a better predictive performance than a single model. The MAE and RMSE of the stacking ensemble had better accuracy metrics than the MAE and RMSE of the individual regression models. To consider the characteristics of spatial−temporal distribution, based on the observation time and station points, datasets were created and trained according to the data distribution of each observed data point for better prediction. In contrast to the previous prediction model

applied to the open ocean, this study can be useful in accurately predicting subsurface temperature profiles in a marginal sea with large spatiotemporal variability in water temperature owing to complex phenomena. When predicting the vertical temperature profile during the strong stratification season, it is crucial to create a predictive model that considers a thin surface mixed layer that is frequently overlooked.

This study devised a method to synthesize data needed to effectively make data-based prediction models for regions with limited observations. A major achievement of this study is the use of machine learning techniques to predict subsurface data that are difficult to measure on satellites.

5. Summary and conclusion

A numerical regional ocean model was successfully simulated in the cloud environment and achieved performance similar to that of the physical server. The numerical ocean model with various grids yielded the same results as the physical server. The cloud-based numerical model experiment environment was provided remotely through the network, and the amount of usage was measured. The computation environment was created without preparing physical equipment in the laboratory.

An MPI-based HPC cluster, which is the execution environment of the numerical ocean model, was constructed. The same numerical ocean model was tested in various environments of cloud vendors, and the performance of the numerical model based on each grid was measured. The CPU resources, memory performance, and cache characteristics of the HPC cluster were classified and measured to estimate the factors influencing the performance of the numerical model in the cloud-based environment. Moreover, it was confirmed that the size and structure of the CPU cache memory were among the various performance factors in improving the computation performance of the numerical model. By comparing and analyzing the performance of the numerical model and the memory performance based on the number of nodes, the degree of performance delay attributed to node expansion was confirmed.

The cloud computing environment also enables researchers to reduce the time and cost of preparing an infrastructure environment for numerical ocean models, secure an environment for collaboration by providing the same environment, and achieve performance using the latest information devices.

In this study, cloud−native, containerization, and orchestration technologies were applied to configure the architecture for the numerical ocean model to achieve the reproducibility of the ocean numerical model and the advantages of preparing the numerical model environment. Numerical ocean modeling was conducted in a container−based environment from a physical server to various public clouds and personal computers, and the model exhibited exact computational reproducibility. Container−based HPC numerical modeling is essential in computational reproducibility and research sharing, even in a homogeneous or heterogeneous environment. The results of the base and container−based models were compared with the RMSE to verify the reproducibility of the numerical ocean model.

In addition to the numerical ocean model, machine learning methods, widely used for data generation and analysis in earth science, were tested in cloud computing. Neural network−based generative models have been applied to synthesize training datasets using ocean observation datasets. Observation datasets in NIFS were applied to the generative model to establish a model for predicting the ocean temperature profile. The synthesized datasets were used as training datasets for the prediction model of the vertical temperature profile in oceans. Because of the insufficient observation datasets for model training, the MAE and RMSE of the model using only observational data are higher than those of the trained model with sufficient data. Based on the accuracy metrics, the performance was evaluated, and the model results for synthetic datasets are better than those of the original datasets.

In this study, the model was focused on the seasonal detection of UWE. Datasets synthesized based on observation data from 1993 to 2012, when SLA data observed through satellite datasets exist, were used as training data. Sea surface temperature and SLA data obtained from the satellite were used to input data (x−values), and the prediction model was trained by combining the temperature profile data as output data (y−values) for each depth from the data observed by NIFS. Data collected from 2013 to 2017 were used as test data for verification. The final predictive model was implemented by combining stand−alone models using the stacking ensemble method to improve the performance of the predictive model.

Synthetic datasets used for model training were generated using GPU in the cloud environment. Predictive results were generated using a numerical ocean model in various cloud–based environments, and training datasets for the machine learning model were generated using generative models. High–resolution numerical and computational machine–learning prediction models, which were difficult to perform in the limited physical infrastructure environment in the past, became possible. In addition, the achievement of computational reproducibility and rapid preparation of the computational environment were achieved in cloud computing. This study demonstrates that the cloud environment can play an essential role in the generation and prediction model of numerical data in earth science.

6. References

Adams, J. K., and Buchwald, V. T. (1969). The generation of continental shelf waves. *J. Fluid Mech.* 35, 815–826. doi:10.1017/S0022112069001455.

Ahmad, H. (2019). Machine learning applications in oceanography. *Int. Aquat. Res.* 2(3), 161−169. doi:10.3153/AR19014.

Ait−Amir, B., Pougnet, and P., Hami, A.E. (2015). 6 − Meta−Model Development. In A. E., Hami & P., Pougnet (Eds). *Embedded Mechatronic Systems 2* (pp 151−179). Elsevier. doi:10.1016/B978−1−78548−014−0.50006−2.

Ali, N., Neagu, D., and Trundle, P. (2019). Evaluation of k−nearest neighbour classifier performance for heterogeneous data sets. *SN Appl. Sci.* 1,1559. doi:10.1007/s42452−019−1356−9.

Amante, C., and Eakins, B. W. (2009). ETOP01 1 arc−minute global relief model: Procedures, data sources and analysis*. NOAA Tech. Memo.* NESDIS NGDC−24*,* 19.

Antonov, J. I., Levitus, S., Boyer, T. P., Conkright, M. E., Brien, T. O., and Stephens, C. (1998). World Ocean Atlas 1998 Vol. 2: Temperature of the Pacific Ocean, NOAA Atlas NESDIS 28, U.S. Government Printing Office, Washington, D.C.

Antonov, J. I., Seidov, D., Boyer, T. P., Locarnini, R. A., Mishonov, A.

V., Garcia, H. E., et al. (2010). *World Ocean Atlas 2009, Vol. 2: Salinity.* S. Levitus, Ed. NOAA Atlas NESDIS 69, U.S. Government Printing Office, Washington, D.C., p. 184.

Atteia, G. E., Mengash, H. A., and Samee, A. (2021). Evaluation of using Parametric and Non-parametric Machine Learning Algorithms for Covid-19 Forecasting. *Int. J. Adv. Comput. Sci. Appl. (IJACSA)*. 12(10). doi: 10.14569/IJACSA.2021.0121071.

Auret, L., and Aldrich, C. (2012). Interpretation of nonlinear relationships between process variables by use of random forests. *Miner. Eng.* 35, 27-42. doi:10.1016/j.mineng.2012.05.008.

AWS. (2016). Elastic Network Adaptor. https://aws.amazon.com/ko/about-aws/whats-new/2016/06/introducing-elastic-network-adapter-ena-the-next-generation-network-interface-for-ec2-instances.

AWS. (2022a). Amazon Elastic Kubernetes Service (EKS). https://aws.amazon.com/eks/?nc1=h_ls.

AWS. (2022b). Amazon Machine Images. http://docs.aws.amazon.com/AWSEC2/latest/UserGuide/AMIs.html.

AWS. (2020). KVM. https://aws.amazon.com/ko/about-aws/whats-new/2020/02/aws-storage-gateway-available-linux-kvm-hypervisor.

AWS. (2022c). Pricing. https://aws.amazon.com/ec2/pricing/on-demand/?nc1=h_ls.

AWS. (2022d). Spot-Instance. https://aws.amazon.com/ec2/spot/?nc1=h_ls/.

AWS. (2022e). Virtual Private Cloud. https://aws.amazon.com/vpc/?nc1=h_ls/.

AWS. (2022f). VPN Tunneling. https://docs.aws.amazon.com/vpn/latest/s2svpn/VPNTunnels.html.

Azure. (2022). Azure Kubernetes Service (AKS). https://azure.microsoft.com/en-us/services/kubernetes-service.

Balogun, A., and Adebisi, N. (2021). Sea level prediction using ARIMA, SVR and LSTM neural network: assessing the impact of ensemble Ocean-Atmospheric processes on models' accuracy. *Geomat. Nat. Hazards Risk*. 12:1, 653-674. doi: 10.1080/19475705.2021.1887372.

Balogun, A., Rezaie, F., Pham, Q. B., Gigović, L., Drobnjak, S., Aina, Y. A., et al. (2021). Spatial prediction of landslide susceptibility in western Serbia using hybrid support vector regression (SVR) with GWO, BAT and COA algorithms. *Geosci. Front.* 12(3). doi:10.1016/j.gsf.2020.10.009.

Beltre, A. M., Saha, P., Govindaraju, M., Younge, A., and Grant, R. E. (2019). Enabling HPC Workloads on Cloud Infrastructure Using Kubernetes Container Orchestration Mechanisms. *2019 IEEE/ACM International Workshop on Containers and New Orchestration Paradigms for Isolated Environments in HPC (CANOPIE-HPC).* 2019, 11-20. doi:10.1109/CANOPIE-HPC49598.2019.00007.

Blackport, R., and Kushner, P. J. (2018). The Role of Extratropical Ocean Warming in the Coupled Climate Response to Arctic Sea Ice Loss. *J. Clim.* 31 (22), 9193–9206. doi:10.1175/JCLI-D-18-0192.1.

Bolton, T. and Zanna, L. (2019). Applications of Deep Learning to Ocean Data Inference and Subgrid Parameterization. *J. Adv. Model. Earth Syst.* 11, 376-399. doi:10.1029/2018MS001472.

Bozzo-Rey, M., Jeanson, M., Nguyen, M., Gauthier, C., Barrette, M., Vachon, P., et al. (2006). Design, Deployment and Bench of a Large Infiniband HPC Cluster. *20th International Symposium on High-Performance Computing in an Advanced Collaborative Environment (HPCS'06).* 8-8. doi:

10.1109/HPCS.2006.18.

Brereton, R. G., and Lloyd, G. R. (2010). Support vector machines for classification and regression. *The Analyst*. 135(2), 230–267. doi:10.1039/B918972F.

Brownlee, J. (2022). *Ensemble Learning Algorithms with Python*. Machine Learning Mastery.

Burges, C. J. (1998). A tutorial on support vector machines for pattern recognition. *Data Min. Knowl. Discov*. 2, 121–167. doi: 10.1023/A:1009715923555.

Canonical. (2022). Microk8s. https://microk8s.io/.

Carton, J. A., and Giese, B. S. (2008). A reanalysis of ocean climate using Simple Ocean Data Assimilation (SODA*). Mon. Weather Rev.* 136(8), 2999–3017. doi: 10.1175/2007MWR1978.1.

Chapman, D. C. (1985). Numerical treatment of cross-shelf open boundaries in a barotropic coastal ocean model. *J. Phys. Oceanogr*. 15(8), 1060–1075. doi:10.1175/1520-0485(1985)015<1060:NTOCSO>2.0.CO;2.

Chen, X., Huang, X., Jiao, C., Flanner, M., Raeker, T., and Palen, B. (2017). Running climate model on a commercial cloud computing environment: A case study using Community Earth System Model (CESM) on Amazon AWS. *Computers & Geo*. 98, 21–25. doi:10.1016/j.cageo.2016.09.014.

Cho, Y. K., and Kim, K. (1996). Seasonal variation of the East Korea Warm Current and its relation with the cold. *water. La Mer*. 34, 172–182.

Choi, B.J., Haidvogel, D.B., and Cho, Y. K. (2004). Nonseasonal sea level variations in the Japan/East Sea from satellite altimeter data. *J. Geophys. Res*. Oceans 109:C12028.

Cho, Y. K., and Kim, K. (2000). Branching mechanism of the Tsushima Current in the Korea Strait. *J. Phys. Oceanogr*. 30(11), 2788–2797.

Copernicus. Data from: Sea level daily gridded data from satellite observations for the global ocean from 1993 to present. Climate Data Store. (2022) https://cds.climate.copernicus.eu/cdsapp#!/dataset/satellite-sea-level-global?tab=overview.

Dee, D. P., Uppala, S. M., Simmons, A. J., Berrisford, P., Poli, P., Kobayashi, S., et al. (2011). The ERA-Interim reanalysis: Configuration and performance of the data assimilation system. *Q. J. R. Meteorol. Soc*. 137(656), 553–597. doi:10.1002/qj.828.

Egbert, G. D., and Erofeeva, S. Y. (2002). Efficient inverse modeling of Barotropic Ocean Tides. *J. Atmos. Oceanic Technol*. 19, 183–204. doi:10.1175/1520-0426(2002)019<0183:EIMOBO>2.0.CO.

Fairall, C. W., Bradley, E. F., Rogers, D. P., Edson, J. B., and Young, G. S. (1996). Bulk parameterization of air–sea fluxes for Tropical Ocean–Global Atmosphere Coupled–Ocean Atmosphere Response Experiment. *J. Geophys. Res.* 101(C2), 3747–3764. doi:10.1029/95JC03205.

Feng, M., Ai, W., Chen, G., Lu, W., and Ma, S. (2020). A Multiple Linear Regression Algorithm for Sea Surface Temperature Retrieval by One−Dimensional Synthetic Aperture Microwave Radiometry. *J. Atmos. Ocean. Technol.* 37(9), 1753−1761. doi: 10.1175/JTECH−D−20−0003.1.

Flather, R. A. (1976). A tidal model of the north−west European continental shelf. *Memoires de la Societe Royale de Sciences de Liege*. (10),141–164.

Gartner: Public Cloud Service. (2018). https://www.gartner.com/en/newsroom/press−releases/2018−08−01−gartner−says−worldwide−iaas−public−cloud−services−market−grew−30−percent−in−2017.

Google. (2022a). Google Kubernetes Engine (GKE). https://cloud.google.com/kubernetes−engine.

Google. (2022b). Hybrid Connectivity. https://cloud.google.com/hybrid−connectivity.

Google. (2022c). Pre−emptible VM Instances. https://cloud.google.com/compute/docs/instances/preemptible.

Goodfellow, I., Pouget−Abadie, J., Mirza, M., Xu, B., Warde−Farley, D., Ozair, S., et al. (2014). Generative adversarial nets. *Adv. Neural Inf. Process. Syst. (NIPS)*. 3(11). doi: 10.1145/3422622.

Gou, Y., Liu, J., and Zhang, T. (2018). KNN regression model−based refinement of thermohaline data. *2018. Proceedings of the Thirteenth ACM International Conference on Underwater Networks & Systems*. 1−8. doi:10.1145/3291940.3291967.

Gregor, L., Kok, S., and Monteiro, P. M. S. (2017). Empirical methods for the estimation of Southern Ocean CO_2: support vector and random forest regression. *Biogeosciences*. 14, 5551–5569. doi: 10.5194/bg−14−5551−2017.

Grüning, B., Chilton, J., Köster, J., Dale, R., Soranzo, N., Beek, M., et al. (2019). Practical Computational Reproducibility in the Life Sciences. *Cell Syst*. 6 (6), 631–635. doi:10.1016/j.cels.2018.03.014.

Gupta, A., Kale, L.V, Gioachin, F., March, V., Suen, C. H., Lee, B., et al. (2013). The who, what, why and how of high performance computing in the cloud. *2013 IEEE 5th International Conference on Cloud Computing Technology and Science*.

2013, 306–314. doi:10.1109/CloudCom.2013.47.

Han, M., Feng, Y., Zhao, X., Sun, C., Hong, F., and Liu, C. (2019). A convolutional neural network using surface data to predict subsurface temperatures in the Pacific Ocean. IEEE Access. 7, 172816–172829. doi: 10.1109/ACCESS.2019.2955957

Hamilton, K., and Ohfuchi, W. (2008). High Resolution Numerical Modelling of the Atmosphere and Ocean, Springer New York, NY. USA.

HPL. (2018). HPL(High-performance Linpack Benchmark). http://www.netlib.org/benchmark/hpl/index.html.

Hu, M. (2021). Integrated Machine Learning and Numerical Modeling for Multiscale Analyses of Coupled Processes in Geosystems. IOP Conf. Ser. Earth Environ. Sci. 861:032055.

Intel. (2019). Xeon Processor Scalable Family Technical Overview. https://software.intel.com/en-us/articles/intel-xeon-processor-scalable-family-technical-overview.

Intel. (2021). Memory Latency Checker v3.6. https://software.intel.com/en-us/articles/intelr-memory-latency-checker.

Ishfaq, H., Hoogi, A. and Rubin, D. (2018). TVAE: Triplet-Based Variational Autoencoder using Metric Learning. https://arxiv.org/abs/1802.04403.

Isoda, Y., and Saitoh, S.−I. (1993). The northward intruding eddy along the coast of Korea. J. Oceanogr. 49, 443−458.

Jeevanandam N. (2021). What does machine learning embedding mean?. Analytics India Magazine. https://analyticsindiamag.com/machine−learning−embedding.

Jiang, Y., Zhang, T., Gou, Y., He, L., Bai, H., and Hu, C. (2018). High−resolution temperature and salinity model analysis using support vector regression. *J. Ambient Intell. Humaniz. Comput.* doi: 10.1007/s12652−018−0896−y.

Jiyang, Y., Gou, Y., Zhang, T., Wang, K., and Hu, C. (2017). A Machine Learning Approach to Argo Data Analysis in a Thermocline. *Sensors*. 17(10), 2225. doi:10.3390/s17102225.

Jung, K., Cho, Y.−K., and Tak, Y.−J. (2017). Performance evaluation of ROMS v3.6 on a commercial cloud system. *Geosci. Model Dev. Discuss.*, https://doi.org/10.5194/gmd−2017−270, 2017.

Jung, K., Cho, Y.−K., and Tak, Y.−J. (2021). Containers and orchestration of numerical ocean model for computational reproducibility and portability in public and private clouds: Application of ROMS 3.6. *Simul. Model. Pract. Theory*. 109, 10235. doi:10.1016/j.simpat.2021.102305.

Kadkhodaei, H. R., Moghadam, A. M., and Dehghan, M. (2020). Hboost:A heterogeneous ensemble classifier based on the

Boosting method and entropy measurement. *Expert Syst. Appl.* 157, 113482. doi:10.1016/j.eswa.2020.113482.

Kang, H. E., and Kang, Y. Q. (1990). Spatio−temporal characteristics of the Ulleung Warm Lens. *Bull. Korean. Fish. Soc.* 203, 407−415.

Katoh, O. (1994). Structure of the Tsushima Current in the southwestern Japan Sea. *J. Oceanogr.* 50, 317−338.

Kim, D., Yang, E. J., Kim, K. H., Shin, C.W., Park, J., Yoo, S., et al. (2012). Impact of an anticyclonic eddy on the summer nutrient and chlorophyll−a distributions in the Ulleung Basin, East Sea (Japan Sea). *ICES J.Mar. Sci.* 69(1), 23−29. doi:10.1093/icesjms/fsr178.

Kim, K., Kim, K. R., Chung, J., Yoo, H., and Park, S., (1991). Characteristics of physical properties in the Ulleung Basin. *J. Oceanol. Soc. Kor.* 26, 83−100.

Kim, Y.−Y., Cho, Y.−K., and Kim, Y. H. (2018). Role of cold water and beta−effect in the formation of the East Korean Warm Current in the East/Japan Sea: a numerical experiment. *Ocean Dyn.* 68, 1013−1023. doi:10.1007/s10236−018−1175−3.

Klemas, V. (2014). Subsurface and deeper ocean remote sensing from satellites: An overview and new results. *Prog. Oceanogr.* 122, 1−9. doi: 10.1016/j.pocean.2013.11.010.

Konkol, M., Kray, C., and Pfeiffer, M. (2019). Computational reproducibility in geoscientific papers: Insights from a series of studies with geoscientists and a reproduction study. *Int. J. Geogr. Inf. Sci*. 33 (2), 408–429.

Krider, L. A., Magner, J. A., Perry, J., Vondracek, B., and Ferrington, L. C. (2013). Air–water temperature relationships in the trout streams of southeastern Minnesota's carbonate–sandstone landscape. *J. Am. Water Resour. Assoc*. 49, 896–907. doi:10.1111/jawr.12046.

Kubernetes Components. (2022). Kubernetes. https://kubernetes.io/docs/concepts/overview/components/.

Kubernetes Container Runtime. (2022). Kubernetes. https://kubernetes.io/docs/setup/production-environment/container-runtimes/.

Kubernetes Controller. (2022). Kubernetes. https://kubernetes.io/docs/concepts/architecture/controller/.

Kubernetes Cluster Networking. (2022). Kubernetes. https://kubernetes.io/docs/concepts/cluster-administration/networking/.

Kubernetes Namespace. (2022). Kubernetes. https://kubernetes.io/docs/concepts/overview/working-with-objects/namespaces/https://kubernetes.io/docs/concepts/overview/working-with-objects/namespaces/.

Kubernetes Overview. (2022). Kubernetes. https://kubernetes.io/docs/concepts/overview/.

Kubernetes Pods. (2022). Kubernetes. https://kubernetes.io/docs/concepts/workloads/pods/.

Kubernetes Volumes. (2022). Kubernetes. https://kubernetes.io/docs/concepts/storage/volumes.

Langr, J., and Bok, V. (2019). GAN in action. Manning Publications Co.

Large, W. G., McWilliams, J. C., and Doney, S. C. (1994). Oceanic vertical mixing: A review and a model with a nonlocal boundary layer parameterization. *Rev. Geophys.* 32, 363–403, doi:10.1029/94RG01872.

Leuliette, E. W., and Wahr, J. M. (1999). Coupled Pattern Analysis of Sea Surface Temperature and TOPEX/Poseidon Sea Surface Height. *J. Phys. Oceanogr.* 29(4), 599-611.

Levin, L. A., Bett, B. J., Gates, A. R., Heimbach, P., Howe, B. M., Janssen, F., et al. (2019). Global Observing Needs in the Deep

Ocean. *Front. Mar. Sci*. 6:241. doi: 10.3389/fmars.2019.00241.

Linux Foundation. (2020). Open Container Initiative. https://opencontainers.org/.

Li, Q.-J., Zhao, Y., Liao, H.-L., and Li, J.-K. (2017). Effective forecast of Northeast Pacific sea surface temperature based on a complementary ensemble empirical mode decomposition–support vector machine method. *Atmos. Ocean. Sci. Lett*. 10:3, 261-267. doi:10.1080/16742834.2017.1305867.

Li, Z., Kaufamn, Y. J., Ichoku, C., Fraser, R., Trishchenko, A. Giglio, L., et al. (2001). A review of AVHRR-based active fire detection algorithms: Principles, limitations and recommendations. In F. J., Ahren, J. G., Goldammer, & C. O., Justice (Eds). *Global and Regional Vegetation Fire Monitoring from Space: Planning a Coordinated International Effort* (pp. 199-225). SPB Academic Publishing.

Locarnini, R. A., Mishonov, A. V., Antonov, J. I., Boyer, T. P., Garcia, H. E., Baranova, O. K., et al. (2010). *World Ocean Atlas 2009, Volume 1: Temperature*. S. Levitus, Ed. NOAA Atlas NESDIS 68, U.S. Government Printing Office, Washington, D.C., p. 184.

Louppe, G. (2014). Understanding Random Forests: From Theory to Practice. [dissertation/Ph.D's thesis]. University of Liège. doi:10.13140/2.1.1570.5928.

Luksa, M. (2018). Kubernetes in Action. Manning Publications CO, NY, USA.

McCalpin, J. D. (1995). Memory bandwidth and machine balance in current high performance computers. *IEEE Computer Society Technical Committee on Computer Architecture (TCCA) Newsletter.*

McCalpin, J. D. (2017). STREAM: Sustainable memory bandwidth in high performance computers, a continually updated technical report (1991-2007). http://www.cs.virginia.edu/stream.

Mell, P., and Grance, T. (2011). The NIST definition of cloud computing recommendations of the National Institute of Standards and Technology, Special Publication 800-145, NIST, Gaithersburg. http://nvlpubs.nist.gov/nistpubs/Legacy/SP/nistspecialpublication800-145.pdf.

Microsoft. (2015). Azure support for Linux RDMA. https://azure.microsoft.com/en-us/updates/azure-support-for-linux-rdma.

Mirza, M., and Osindero, S. (2014). Conditional Generative Adversarial Nets. https://arxiv.org/abs/1411.1784.

MIT Data To AI Lab. (2022). Synthetic Data Vault (SDV). https://sdv.dev/SDV/index.html.

Montes, D., Añel, J. A., Pena, T. F., Uhe, P., and Wallom, D. C. H. (2017). Enabling BOINC in infrastructure as a service cloud system. *Geosci. Model Dev.* 10, 811–826. doi: 10.5194/gmd-10-811-2017.

Morrill, J. C., Bales, R. C., and Conklin, M. H. (2005). Estimating Stream Temperature from Air Temperature: Implications for Future Water Quality. *J. Environ. Eng. ASCE.* 131. doi:10.1061/(ASCE)0733-9372(2005)131:1(139).

NIFS. Data from: NIFS Serial Oceanographic observation. Korea Oceanonic Data Center (2022) https://www.nifs.go.kr/kodc/coo_list.kodc.

NIST. (2022). Hybrid clouds. https://csrc.nist.gov/glossary/term/Hybrid_cloud.

NOAA. Data from: NOAA High-Resolution Sea Surface Temperature (SST) Analysis Products. National Center for Environmental information, (2022) https://www.ncei.noaa.gov/access/metadata/landing-page/bin/iso?id=gov.noaa.ncdc:C00680.

Nüst, D., Konkol, M., Schutzeichel, M., Pebesma, E. C. Kray, C. , Przibytzin, H., et al. (2017). Opening the Publication.

NVIDIA. (2022). PGI: Community Edition.

http://www.pgroup.com/products/community.htm.

Oesterle, F., Ostermann, S., Prodan, R., and Mayr, G. J. (2015). Experiences with distributed computing for meteorological applications: Grid computing and cloud computing. *Geosci. Model Dev.* 8, 2067−2078. doi: 10.5194/gmd−8−2067−2015.

Open Reproducible Research. (2020). https://www.fosteropenscience.eu/.

Perkins, H., Teague, W. J., Jacobs, G. A., Change, K. I., and Suk, M.−S. (2000). Currents in KoreaTsushima Strait during summer 1999. *Geophys.Res. Lett.* 27, 3033−3036.

Preller, R. H., and Hogan, P. J. (1998). Oceanography of the Sea of Okhotsk and the Japan/East Sea. The Sea: The Global Coastal Ocean. In A., Robinson & K., Brink (Eds). *Regional Studies and Syntheses, Vol. 11.* (pp. 429−481). John Wiley and Sons.

Rajan, A., Joshi, B. K., Rawat, A., Jha, R., and Bhachavat, K. (2012). Analysis of process distribution in HPC cluster using HPL. *2012 2nd IEEE International Conference on Parallel, Distributed and Grid Computing*. 2012, 85−88. doi:10.1109/PDGC.2012.6449796.

Rocca J. (2019). Ensemble methods: Bagging, boosting and stacking. https://towardsdatascience.com/ensemble−methods−bagging−boosting−and−stacking−c9214a10a205.

Roemmich, D., Alford, M. H., Claustre H, Johnson, K., King B, Morum, J., Oke, P., Owens, W. B., et al. (2019). On the Future of Argo: A Global, Full–Depth, Multi–Disciplinary Array. *Front. Mar. Sci.* 6. doi:10.3389/fmars.2019.00439.

ROMS. (2022). Regional Ocean Modeling System (ROMS). https://www.myroms.org/.

Schmidt, J. O., Bograd, S. J., Arrizabalaga, H., Azevedo, J. L., Barbeaux, S. J., Barth, J. A., et al. (2019). Future Ocean Observations to Connect Climate, Fisheries and Marine Ecosystems. *Front. Mar. Sci.* 6. doi:10.3389/fmars.2019.00550.

Sci–kit.learn.org. (2022). https://scikit–learn.org/stable.

Seo, G. –H., Cho, Y. –K., Cho, B. –J., Kim, K. –Y., Kim, B. –g., and Tak, Y. –J. (2014). Climate change projection in the Northwest Pacific marginal seas through dynamic downscaling, J. Geophys. Res. 119, 3497–3516. doi:10.1002/2013JC009646.

Shah, J., and Dubaria, D. (2019). Building Modern Clouds: Using Docker, Kubernetes & Google Cloud Platform. *2019 IEEE 9th Annual Computing and Communication Workshop and Conference (CCWC).* pp.184–189. doi:10.1109/CCWC.2019.8666479.

Shchepetkin, A. F., and McWilliams, J. C. (2005). The Regional

Oceanic Modeling System (ROMS): A split-explicit, free-surface, topography-following-coordinate oceanic model. *Ocean Modell.* 9, 347–404. doi: 10.1016/j.ocemod.2004.08.002.

Signell, R. P., and Pothina, D. (2019). Analysis and visualization of coastal ocean model data in the cloud. *J. Mar. Sci. Eng.* 7(4), 110. doi:10.3390/jmse7040110.

Sommer, J., Chassignet, E., and Wallcraft, A. (2018). *Ocean Circulation Modeling for Operational Oceanography: Current Status and Future Challenges*. doi:10.17125/gov2018.ch12.

Su, H., Wu, X., Yan, X. H., and Kidwell, A. (2015). Estimation of subsurface temperature anomaly in the Indian Ocean during recent global surface warming hiatus from satellite measurements: A support vector machine approach. *Remote Sens. Environ.* 160, 63–71. doi: 10.1016/j.rse.2015.01.001.

Sultan, S. Ahmad, I., and Dimitriou, T. (2019). Container Security: Issues, Challenges, and the Road Ahead. *IEEE Access*. 7, 52976–52996. doi:10.1109/ACCESS.2019.2911732.

Than, K. (2017). 21st-century Earth science is computer intensive and data driven. https://earth.stanford.edu/news/21st-century-earth-science-computer-intensive-and-data-driven#gs.1gp5iq.

Tintó, O., Acosta, M., Castrillo, M., Cortés, A., Sanchez, A., Serradell, K., et al. (2017). Optimizing domain decomposition in an ocean model: the case of NEMO. *Procedia Comput. Sci.* 108, 776–785. doi: 10.1016/j.procs.2017.05.257.

UL HPC Team. (2021). UL HPC MPI Tutorial: Building and Running OSU Micro-Benchmarks. https://ulhpc-tutorials.readthedocs.io/en/latest/.

Vance, T. C., Wengren, M., Burger, E., Hernandez, D., Kearns, T., Medina-Lopez, E., et al. (2019). From the oceans to the cloud: opportunities and challenges for data, models, computation, and workflows. *Front. Mar. Sci.* 6:211. doi.org/10.3389/fmars.2019.00211.

Veiga, V. S., Simon, M., Azab, A., Fernandez, C., Muscianisi, G., Fiameni, G., et al. (2019). Evaluation and Benchmarking of Singularity MPI containers on EU Research e-Infrastructure. *2019 IEEE/ACM International Workshop on Containers and New Orchestration Paradigms for Isolated Environments in HPC (CANOPIE-HPC)*. 2019, 1-10. doi: 10.1109/CANOPIE-HPC49598.2019.00006.

Vörösmarty, C., Fekete, B., and Tucker, B. (1996). River discharge database version 1.0 (RivDIS v1.0), Vol. 0–6., A contribution to IHP-V theme 1.

Wang, H., Song, T., Zhu, S., Yang, S., and Feng, L. (2021). Subsurface Temperature Estimation from Sea Surface Data Using Neural Network Models in the Western Pacific Ocean. *Mathematics*. 9, 852. doi:10.3390/math9080852.

Wang, J., Song, Y., Leung, T., Rosenberg, C., Wang, J., Philbin, J., et al. (2014). Learning fine-grained image similarity with deep ranking. *In Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*. 1386–1393.

Wang, Q., Guo, X., and Takeoka, H. (2008). Seasonal variations of the Yellow River plume in the Bohai Sea: A model study. *J. Geophys. Res*. 113:C08046. doi: 10.1029/2007JC004555.

Weller, R. A., Baker, D. J., Glackin, M. M., Roberts, S. J., Schmitt, R. W., Twigg, E. S., et al. (2019). The Challenge of Sustaining Ocean Observations. *Front. Mar. Sci*. 6:105. doi: 10.3389/fmars.2019.00105.

Xu, L., Skoularidou, M., and Cuesta-Infante, A., Veeramachaneni, K. (2019). Modeling tabular data using conditional gan. *Adv. Neural Inf. Process. Syst*. 32.

Yaml.org. (2021). YAML. https://yaml.org.

Yao, Z., and Ruzzo, W. (2006). A Regression-based K neareast neighbor algorithm for gene function prediction from heterogeneous data. *BMC Bioinforma*. 71(7), 1-11. doi:

10.1186/1471-2105-7-S1-S11.

Younge, A. J., Henschel, R., Brown, J. T., Laszwwski, G. V., Qui, J., and Fox, G. C. (2011). Analysis of virtualization technologies for high performance computing environment. *2011 IEEE 4th International Conference on Cloud Computing*. 2011, 9–16. doi:10.1109/Cloud.2011.29.

Zhang, C., and Ma, Y. (2012). *Ensemble Machine Learning: Methods and Applications 2012th Edition*. Springer.

Zhang S. Q., Yang, L., Ma, X. H., Wang, H. N., Zhang, X. F., Xiao-Lin Yu, X. L., et al. (2018). The 'Two oceans and one sea' extended range numerical prediction system with an ultra-high resolution atmosphere-ocean-land regional coupled model. *Atmos. Ocean. Sci. Lett.* 11:4, 364-371. doi:10.1080/16742834.2018.1494498.

Zhuang, J., Jacob, D. J., Lin, H., Lundgren, E. W., Yantosca, R. M., Gaya, J. F., et al. (2020). Enabling high-performance cloud computing for Earth science modeling on over a thousand cores: Application to the GEOS-Chem atmospheric chemistry model. *J. Adv. Model. Earth Syst.* 12(5):e2020MS002064. doi:10.1029/2020MS002064.

7. Abstract (in Korean)

지구의 변화와 현상을 연구하기 위해 많은 과학자들은 수치 모델을 기반으로 한 고해상도 모델 결과를 사용하거나 관측된 데이터로 머신러닝 기반 예측 모델을 개발하고 활용한다. 정보기술이 발전함에 따라 지역 및 전 지구적인 고해상도 수치 모델링과 머신러닝 기반 지구과학 데이터 생성을 위한 실용적인 방법론이 필요하다.

본 연구는 지구과학의 고해상도 수치 모델과 머신러닝 기반 예측 모델을 기반으로 한 데이터 생성 및 처리가 클라우드 환경에서 효과적으로 구현될 수 있음을 제안한다.

클라우드 컴퓨팅에서 고해상도 수치 해양 모델 구현의 재현성과 이식성을 검증하기 위해 북서태평양, 동해, 황해 등 모델 영역의 다양한 해상도에서 수치 해양 모델의 성능을 시뮬레이션하고 분석하였다. 컨테이너화 방식을 통해 다양한 인프라 환경 변화에 대응하고 계산 재현성을 효과적으로 확보할 수 있었다.

머신러닝 기반 데이터 생성의 적용을 검증하기 위해 생성 모델을 이용한 표층 이하 온도 데이터의 데이터 증강을 실행하여 해양의 수직 온도 분포를 예측하는 모델 훈련을 위한 대용량 데이터 세트를 준비했다. 예측모델 훈련을 위해 위성 데이터에 비해 상대적으로 부족한 관측 데이터에 대해서 생성 모델을 사용하여 데이터 증강을 수행하였다. 모델의 예측성능 비교에는 관측 데이터 외에도 HYCOM 데이터 세트를 사용하였으며, 증강 데이터의 데이터 분포는 입력 데이터 분포와 유사함을 확인하였다. 독립형 예측 모델을 결합한 앙상블 방식은 기존

관측 데이터를 기반으로 하는 예측 모델의 성능에 비해 향상되었다. 데이터합성을 위해 많은 양의 계산 자원이 필요했으며, 데이터 합성은 클라우드 기반 GPU 환경에서 수행되었다.

고해상도 수치 해양 모델 시뮬레이션, 예측 모델 개발, 데이터 생성 방법은 해양 과학 분야에서 예측 능력을 향상시킬 수 있다. 본 연구에서 사용된 클라우드 컴퓨팅 기반의 수치 모델링 및 생성 모델은 지구 과학의 다양한 분야에 광범위하게 적용될 수 있다.