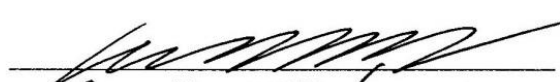
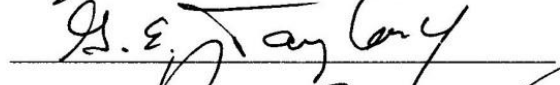
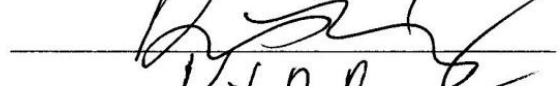

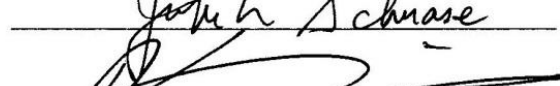

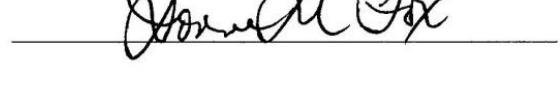
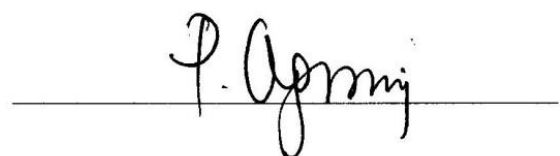


OPTIMIZING GEOSPATIAL CYBERINFRASTRUCTURE TO IMPROVE THE  
COMPUTING CAPABILITY FOR CLIMATE STUDIES

by

Zhenlong Li  
A Dissertation  
Submitted to the  
Graduate Faculty  
of  
George Mason University  
in Partial Fulfillment of  
The Requirements for the Degree  
of  
Doctor of Philosophy  
Earth Systems and Geoinformation Sciences

Committee:

	Dr. Chaowei Yang, Dissertation Director
	Dr. George Taylor, Committee Member
	Dr. Ruixin Yang, Committee Member
	Dr. Kirk Borne, Committee Member
	Dr. John L. Schnase, Committee Member
	Dr. Anthony Stefanidis, Department Chairperson
	Dr. Donna M. Fox, Associate Dean, Office of Student Affairs & Special Programs, College of Science
	Dr. Peggy Agouris, Dean, College of Science

Date: 04-29-2015

Spring Semester 2015  
George Mason University  
Fairfax, VA

Optimizing Geospatial Cyberinfrastructure to Improve the Computing Capability for  
Climate Studies

A Dissertation submitted in partial fulfillment of the requirements for the degree of  
Doctor of Philosophy at George Mason University

by

Zhenlong Li  
Master of Science  
George Mason University, 2010  
Bachelor of Engineering  
Wuhan University, 2006

Director: Chaowei Yang, Professor  
Department of Geography and Geoinformation Science

Spring Semester 2015  
George Mason University  
Fairfax, VA

Copyright: 2015 by Zhenlong Li  
All Rights Reserved

## **DEDICATION**

I dedicated this dissertation to my parents, my beloved wife Weili Xiu, and my wonderful son Mason J. Li.

## **ACKNOWLEDGEMENTS**

First of all, I would like to express my deepest gratitude to my adviser and committee chair, Prof. Chaowei Yang, for his excellence guidance, patient encouragement, and enormous support during the whole process of my graduate study. As my adviser, he teaches me the fundamental skills of how to think scientifically, critically and creatively. Besides the knowledge, he continually and convincingly conveys me the spirit of research, scholarship and teaching which is essential for my academic career. I also sincerely thank my committee members, Prof. George Taylor, Dr. John L. Schnase, Prof. Ruixin Yang, and Prof. Kirk Borne, for their expertise and time in commenting, advising, and encouraging throughout the entire process. Finally, I thank all my colleagues and friends at CISC/GMU for the pleasant time we have spent together: Huayi Wu, Wenwen Li, Qunying Huang, Jing Li, Kai Liu, Jizhe Xia, Min Sun, Lizhi Miao, Chen Xu, Yunfeng Jiang, Jibo Xie, Haijun Zhu and others.

## TABLE OF CONTENTS

	Page
LIST OF TABLES .....	viii
LIST OF FIGURES .....	ix
LIST OF EQUATIONS .....	xi
ABSTRACT .....	xii
CHAPTER 1 INTRODUCTION.....	1
1.1 A Study Case: ModelE Sensitivity Analysis.....	3
1.2 Computational Challenges .....	4
1.3 Geospatial Cyberinfrastructure and Cloud Computing.....	8
1.4 Objective and Summary of Contribution .....	10
CHAPTER 2 MODEL AS A SERVICE.....	15
2.1 Introduction .....	15
2.2 Literature Review .....	16
2.2.1 Model Web.....	16
2.2.2 Computing technologies for computing intensive model simulations.....	17
2.3 Methodologies .....	19
2.3.1 MaaS in the cloud .....	19
2.3.2 MaaS architecture .....	21
2.3.3 Model I/O.....	23
2.3.4 Mechanism for publishing new models .....	25
2.3.5 Mechanism for parallelizing ensemble model run.....	26
2.3.6 Mechanism for handling model output .....	29
2.4 Evaluation.....	31
2.4.1 Experiment setup .....	31
2.4.2 Prototype .....	32
2.4.3 Result .....	34

2.4.4	Key features of MaaS.....	38
2.4.5	Advantages of MaaS .....	40
2.4.6	User roles potentially benefited from MaaS .....	42
2.5	Chapter Summary.....	43
CHAPTER 3 A SCALABLE BIG CLIMATE DATA ANALYTICS FRAMEWORK		44
3.1	Introduction .....	44
3.2	Literature Review .....	45
3.2.1	Database technologies for managing big geospatial data .....	45
3.2.2	Parallelization technologies to process big geospatial data .....	47
3.2.3	Auto-scaling Hadoop cluster in cloud.....	48
3.3	Big Climate Data Processing with MapReduce .....	51
3.3.1	Spatiotemporal Decomposition Mechanism .....	51
3.3.2	MapReduce-enabled Framework for Processing Big Climate Data .....	53
3.3.3	Evaluation of big climate data processing .....	56
3.4	Auto-scaling Mechanism.....	59
3.4.1	Auto-Scaling Framework.....	60
3.4.2	CoveringHDFS .....	62
3.4.3	Auto-Scaling algorithm.....	63
3.4.4	Evaluation of auto-scaling mechanism .....	68
3.5	Chapter Summary.....	75
CHAPTER 4 SERVICE-ORIENTED CLOUD-BASED SCIENTIFIC WORKFLOW		76
4.1	Introduction .....	76
4.2	Literature Review .....	76
4.3	Methodologies .....	78
4.3.1	Framework .....	78
4.3.2	Unified service model .....	79
4.3.3	Loosely-coupled Service I/O Mechanism.....	82
4.3.4	Cloud-based Workflow Execution Environment.....	85
4.4	Implementation Architecture.....	87

4.5	Chapter Summary.....	90
CHAPTER 5 INTEGRATION AND VALIDATION: MODELE SENSITIVITY ANALYSIS 91		
5.1	Introduction.....	91
5.2	Services for the Study Case .....	91
5.3	Executable Workflow of the Study Case .....	92
5.4	Chapter Summary .....	96
CHAPTER 6 CONCLUSIONS AND FUTURE RESEARCH..... 97		
6.1	Conclusions.....	97
6.2	Future Research .....	98
REFERENCES .....		101



## LIST OF TABLES

Table	Page
Table 1.1 Seven tested atmospheric parameters in the experiment .....	3
Table 2.1 Comparison of time spent on setting up ModelE environment with/without MaaS (MaaS usage does not include the time for registering ModelE into MaaS).....	35
Table 2.2 Geospatial Modeling Challenges addressed by key features of MaaS .....	40
Table 2.3 Supportability of the features by the three platforms (HPC, IaaS, MaaS) for key features of MaaS .....	42
Table 3.1 Three Hadoop clusters .....	70

## LIST OF FIGURES

Figure	Page
Figure 2.1 General framework of the cloud-enabled MaaS.....	21
Figure 2.2 Overall architecture of MaaS.....	23
Figure 2.3 Model I/O data transfer between model VM, MaaS server and data server ...	24
Figure 2.4 Mechanism for parallelizing ensemble model run (model VMs are provisioned based on the same Model VM Image but running with different configurations).....	28
Figure 2.5 Mechanism for concurrent user requests (model VMs provisioned based on the different Model VM Images and running different models configurations) .....	29
Figure 2.6 Workflows for simulating and analyzing 14 different scenarios using ModelE: A) without using MaaS and B) using MaaS .....	32
Figure 2.7 Physical architecture for the MaaS prototype .....	33
Figure 2.8 (A) The Web Portal GUI showing the running status of the submitted tasks; (B) Model output progressively transmitted to data server to ensure data availability.....	34
Figure 2.9 Timeline for provisioning and running the 14 model runs with a 10-year simulation period. X-axis is time spent on the three stages, while y-axis is the 14 model runs (for better visualization, the timeline is not evenly scaled) .....	37
Figure 3.1 Hierarchical structure of the multi-dimensional climate data .....	52
Figure 3.2 MapReduce-based framework for processing big climate data .....	55
Figure 3.3 Algorithm for computing annual global mean of multiple datasets based on the framework .....	56
Figure 3.4 Performance evaluation result for the MapReduce-enabled big climate data processing .....	58
Figure 3.5 Auto-scaling Framework .....	61
Figure 3.6 Structure of the Hadoop cluster with CoveringHDFS.....	63
Figure 3.7 An example status of the jth loop of Equation 3.4 .....	66
Figure 3.8 100 workloads submitted to the cluster in a 10-hour period with a time spacing of 6 minutes. A total of 224 jobs were submitted (each dot representing each workload)69	
Figure 3.9 Time consumed by each workload (100 workloads in total) for the three clusters in the 10-hour period. The small diagram on the top-right corner is the zoomed-in view for the first 1.5 hour .....	71

Figure 3.10 (A) The size change of the auto-scaling cluster in the 10-hour period (the number of slaves monitored every 30 seconds). The 7-slave and 14-slave clusters are represented as two parallel lines; (B) Number of idle slaves in the 10-hour period.....	72
Figure 3.11 (A) Time consumed for finishing all workloads for each cluster. Time calculated by summing the 224 job finish times; (B) Cluster Idle Slave Time (CIST) for each cluster in the 10-hour period.....	73
Figure 4.1 Framework architecture.....	79
Figure 4.2 Service definition metadata example (Infrastructure Service: ProvisionVM)	82
Figure 4.3 Loosely-coupled Service I/O mechanism.....	83
Figure 4.4 A typical workflow with four types of services .....	85
Figure 4.5 Prototype implementation architecture.....	88
Figure 4.6 GUI of the web-based workflow builder.....	89
Figure 5.1 Executable workflow for the study case built in the prototype .....	93
Figure 5.2 Workflow output published in a web accessible folder (A), presented as correlation statistics in CSV format (B), and plotted output climate variables highly affected by the seven model input parameters ( $R^2 > 0.6$ , 9 of 57 pairs) .....	94
Figure 5.3 Monitor the status of model-run in a web-based application .....	95
Figure 5.4 Yearly mean “net thermal radiation” for three simulation scenarios for a select study area .....	96

## LIST OF EQUATIONS

Equation	Page
Equation 3.1 .....	51
Equation 3.2 .....	53
Equation 3.3 .....	64
Equation 3.4 .....	65
Equation 3.5 .....	67

## **ABSTRACT**

### **OPTIMIZING GEOSPATIAL CYBERINFRASTRUCTURE TO IMPROVE THE COMPUTING CAPABILITY FOR CLIMATE STUDIES**

Zhenlong Li, Ph.D.

George Mason University, 2015

Dissertation Director: Dr. Chaowei Yang

Climate simulation has significant uncertainties due to our current limited understanding of the processes and interactions between different components of the Earth. Model sensitivity analysis, which tests the sensitivity of model output to the input parameter values, is a standard practice for determining the model uncertainties and improving model accuracy. A common approach for climate model sensitivity analysis is to run a model many times by sweeping a large number of adjustable parameters. However, this approach is hampered by three computational challenges: computing intensity, data intensity, and procedure complexity. This dissertation proposes three optimization methodologies to address these challenges respectively, including 1) tackling the computing intensity challenge posed by climate simulation using Model as a Service, a new service model in the context of cloud computing; 2) managing and processing the big model output – “data intensity” – using a scalable big spatiotemporal data analytics

framework; 3) solving the procedure complexity issue using a service-oriented cloud-based scientific workflow framework.

The feasibility and efficiency of these approaches is demonstrated by a case study – ModelE sensitivity analysis. Experiment result shows that 1) Model as a Service reduced the time spent on 300 model-runs (with forty machines) from 38 days to 5 days; 2) the data analytic framework archived 6 times speedup when processing 100 model-run outputs with 6 machines; and 3) the workflow framework transformed the complex analysis by dragging and connecting steps in a visually intuitive diagram. By integrating the three optimization methodologies seamlessly, the time spent on ModelE sensitivity analysis (including setting up model, running model and analyzing model output) is reduced from approximate two months to five days.

This research offers a computational solution to efficiently sweep a large number of adjustable climate model input parameters for identifying model uncertainty. It helps scientists to find answers in a more efficient way to the questions like “which model parameter is more sensitive to simulated climate changes?”. This research also provides a valuable guideline on bridging the computing infrastructure and computing requirements for climate studies. Since the challenges of computing intensity, data intensity and procedure complexity are quite common in geosciences, the proposed optimization methodologies can be broaden from the climate science to broader geoscience domains. In addition, this research provides a potential solution to the uncertainty quantification (UQ) problems for general geoscience applications.

## **CHAPTER 1 INTRODUCTION**

Global climate change has become one of the biggest concerns for human kind in the 21st century due to its broad impacts on society and ecosystems worldwide and across a variety of economic sectors. According to the latest Fifth Assessment Report (AR5) of the Intergovernmental Panel on Climate Change (IPCC), atmospheric temperature measurements show an estimated warming of 0.85 degrees Celsius (1.5 degrees Fahrenheit) since 1880 with the fastest rate of temperature increase in the Arctic, and the global surface temperature will keep rising beyond year 2100 in all scenarios except the lowest emission scenarios. It is extremely like that (a greater than 95 percent chance) human activities of releasing large amount of greenhouse gases such as carbon dioxide are a major contribution to this change (IPCC, 2013).

To advance the understanding, and to limit and adapt the impacts of global climate change, it is essential to explore the factors that cause climate change, and to identify the indicators associated with the change. Climate modeling is a fundamental methodology to reconstruct past climate conditions, understand the dynamics of climate system and project future climate changes by quantifying the interactions among atmosphere, land, ocean, and sea ice with numerical description (Skamarock and Klemp 2008). Comprehensive climate models are the only tools that can be used to determine future climate change at both a global and regional level (Murphy et al 2004). However, a

major problem the decision makers are facing is that different climate models produce different projected climate changes due to unknown uncertainties and mixed model qualities. Quantifying these model uncertainties associated is a critical priority for climate research.

Testing the sensitivity of input parameters is a standard climate modeling practice for determining the model uncertainties (Murphy et al. 2004). However, due to the complexity of climate models, it is a challenging task to fully explore the possible future climate forcing, climate model parameterization values, and climate observations. Therefore, most model sensitivity studies explored only a limited set of parameter-combinations. So an important question still remains open: how to search through this multi-dimensional parameter space **in an efficient way** to identify which parameter is more sensitive to simulated climate changes? This question raises three computational challenges given current limited computing resources and infrastructure: computing intensity, data intensity, and procedure complexity. Aiming to improve computing infrastructure for climate studies, this dissertation optimizes geospatial cyberinfrastructure in a cloud computing environment to tackle the three computational challenges. To clearly identify the challenge and illustrate my solution, a practical example of climate model sensitivity analysis is used as an example.



## 1.1 A Study Case: ModelE Sensitivity Analysis

Climate@Home<sup>1</sup> is a project initiated by NASA to advance climate modeling studies (Sun et al. 2012, Li et al. 2013). In this project, to study the sensitivity of ModelE<sup>2</sup> (global climate model developed by NASA, Schmidt et al. 2014), 300 ensemble model-runs are conducted for each experiment, sweeping seven atmospheric parameters (Table 1.1). The simulation period is from December 1949 to January 1961 with a  $4^0 \times 5^0$  spatial resolution and a monthly temporal resolution. Each model-run generates ~10 gigabytes data in four dimensions (3D space and 1D time) with 336 climatic variables. The experiment produced three terabytes of data in total.

Table 1.1 Seven tested atmospheric parameters in the experiment

Parameter	Definition	Range	Default
funio_denom	Affects fraction of time that clouds in a mixed-phase regime over ocean are ice or water	5 – 25	22
autoconv_mult	Multiplies rate of auto conversion of cloud condensate	0.5 - 2	1
radius_mult	Multiplies effective radius of cloud droplets	0.5 - 2	1
ent_conf1	Entrainment coefficient for convective plume	0.1 – 0.9	0.3
ent_conf2	Entrainment coefficient for secondary convective plume	0.1 - - 0.9	0.6
U00a	Relative humidity threshold for stratus cloud formation	0.4 – 0.8	0.6

<sup>1</sup> [http://www.nasa.gov/offices/ocio/ittalk/08-2010\\_climate.html#.VT6AsTHF-BI](http://www.nasa.gov/offices/ocio/ittalk/08-2010_climate.html#.VT6AsTHF-BI)

<sup>2</sup> <http://www.giss.nasa.gov/tools/modelE/>

To identify which of the 336 output variables are sensitive to the seven input parameters, the three terabytes model output is analyzed. Specifically, the following steps are taken:

- S1.** Prepare machines: prepare and configure 40 machines;
- S2.** Setup Model: install, configure and compile ModelE on 40 machines;
- S3.** Run Model: run ModelE 300 times sweeping seven input parameters;
- S4.** Preprocess: convert model output (monthly .acc files) into NetCDF files, and combine monthly data to reduce the file numbers;
- S5.** Manage data: store the NetCDF files in a file system or database;
- S6.** Process: for each of the 336 variables in each of the 300 model-runs, calculate the global annual mean and 10-year mean;
- S7.** Analysis: conduct linear regression analysis for each Parameter-Variable (P, V) pair (totally  $336 \times 7$  pairs) using the 300 model-runs; and
- S8.** Visualization: identify and plot the variables most affected by the parameters.

## **1.2 Computational Challenges**

To finish this experiment, it took around 2 months for a Ph.D. student from GMU collaborated with a NASA scientist. For the 2 months, ~12 days were spent on setting up

model environment, ~38 days on running the model, and ~8 days on collecting/managing/analyzing model output. Two-month is an unacceptable long time for finishing one experiment. Furthermore, such experiment often needs to be conducted many times to reach a sound conclusion. This study case demonstrates clearly the three challenges of computing intensity, data intensity and procedure complexity.

- **Computing Intensity**

Climate models divide the Earth into a three dimensional grid, simulate the interactions of atmosphere, ocean, and land surface for each grid point by conducting complex mathematical calculations, and evaluate the interactions between each grid and its neighboring grids. This intrinsic nature makes running climate models an extremely computing intensive process. In the third step of the study case (S3), each model-run requires ~5 days for a 10-year simulation. In fact, it is normal for global warming predictions to span decades or centuries with a number of human induced key parameters, such as CO<sub>2</sub> and other greenhouse gases (Keenlyside et al. 2008). Conducting such a long term climate predictions requires large amounts of computing resources to be ready for use in a short time.

Climate model sensitivity analysis requires hundreds or even thousands model-runs, which demands a hundreds- or thousands- fold increase in computing resources (Ferraro et al. 2003) compared to a single model-run. As illustrated in the study case, traditional computing infrastructure cannot finish the 300 model-runs with reasonable effort and time. Even with 40 machines, it still took ~38 days to finish the 300 model-

runs. In addition, scientists need quick access to large amounts of computing resources for a specific time period for conducting modeling experiment, but continuously accessibility of these large computing resources is not required (U.S. Department of Energy, 2011). Such disruptive computing requirements calls for a scalable infrastructure backed by a large computing pool, which provides the ability of provisioning ready-to-go model environment including both hardware (e.g. network, computing and storage) and software (e.g. OS, model code, dependent libraries, and data manipulation tools).

- **Data Intensity**

Climate simulation is a data intensive process which generates vast amounts of multi-dimensional climate data. The data volume is rapidly accumulated with the increasing number of model-runs in climate model sensitivity analysis. In the study case, 300 model-runs generate approximate 3terabytes climate data in just one experiment. Making sensing of these big model output is critical for scientists to answer key questions such as how the change of model input parameters impact the model simulation result. In the study case, step S4, S5, S6, and S7 are data intensive procedures. How to efficiently analyze vast volumes of model output within an acceptable time period poses several challenging issues.

First, storing and managing these model output is challenging in that highly scalable storage is required for the rapid accumulated data when conducting large number of model-runs. There the traditional approach of storing the data in a centralized repository is no longer desirable and a scalable data management framework is critical

for managing these datasets. Second, efficiently analyzing these data is not an easy task. For example, it takes about 6 hours to read 3 terabytes data using one computer (suppose the read speed is 150M/s). Therefore, distributed parallel computing is desirable to process the big climate data in an acceptable time frame. Third, climate data analytics need to deal with heterogeneous data formats (e.g., array-based data, text files, and images), access distributed data sources, and share the result. Different data access protocols (e.g., FTP, HTTP) and data service standards (e.g., WCS, WFS, and OpenDAP) are normally involved in each step's input/output. Hence, a mechanism to encapsulate these data heterogeneities is essential.

- **Procedure Complexity**

Climate models are quite complex, and setting up the model is onerous and time consuming (Harrop et al., 2008; Nefedova et al., 2006), often having to be repeated many times. Before carrying out the simulation, a lot of works need to be done, such as preparing the computer that hosts the model, installing model and its dependency libraries, compiling the model and running data preprocess codes (step S1). The onerous task of setting up the model is further exacerbated by the need of installing the model on different machines to run multiple simulations at the same time (Nefedova et al. 2006) as required by conducting ensemble runs (step S2). As a result, scientists often become overwhelmed by the model setup procedure. How to provide a plug-and-play mechanism with simple and intuitive interface that enables scientists to easily setup and run the model is a challenging and yet urgent demand.

In addition, the experiment normally requires complex steps with a specific sequence. For example, the study case needs eight steps (S1 to S8) from experiment environment setup, to climate simulation, to data visualization. A workflow platform tailored for handling these procedures is critical for managing, conducting and reusing the complex processes. In addition, conducting each step requires different tools, libraries and external processing services. To accomplish an analytical task, scientists normally need to discover appropriate tools/libraries, write their own programs/scripts and deal with Linux command lines. For example, S4 requires data format conversion tools, and S6 requires specific tools using libraries (e.g., NetCDF-Java<sup>3</sup>). And, for S7 and S8, scientists need to program using R script or other languages. A mechanism to integrate these heterogeneous tools and libraries is essential.

### **1.3 Geospatial Cyberinfrastructure and Cloud Computing**

A geospatial cyberinfrastructure (GCI) combines geospatial data/information, computing platforms, computational services and network protocols to enable scientist perform GIScience discovery (Yang et al. 2010). GCI is an inter-disciplinary research effort which is advanced by geospatial information science, computer science, and data science. In the data-rich and information-driven era, GCI is becoming increasingly important to facilitate addressing fundamental scientific questions, which is well evidenced by the recent studies in GCI (Liang and Huang 2013, Zhang and Tsou 2009, Yang et al. 2011b, Li et al. 2013, NSF 2011). Particularly, EarthCube (NSF 2011) is a

---

<sup>3</sup> <http://www.unidata.ucar.edu/software/thredds/current/netcdf-java>

National Science Foundation's (NSF) strategic movement for building a community-driven cyberinfrastructure to integrate information and data across the geosciences.

As illustrated in the study case, research problems and applications in geosciences are commonly complex, data- and computing- intensive, requiring large amounts of computational power. Elastically providing and scheduling computational resources through services in GCI is a significant element to handle these challenges. Cloud computing, characterized by on-demand self-service, availability, scalability and measured cost, is a new computing paradigm that has been matured and widely used in the past few years. More specifically, according to the National Institute of Standards and Technology (NIST), cloud computing is "a model for enabling convenient, on-demand network access to a shared pool of configurable computing resources (e.g., networks, servers, storage, applications and services) that can be rapidly provisioned and released with minimal management effort or service provider interaction" (Mell and Grance, 2009). Public cloud services provide an alternative approach for scientists to lease computing resources from commercial cloud computing providers. Cloud computing technologies can also be used to build private cloud platforms to provide scalable and flexible computing environment. The types of cloud service models primarily include Infrastructure as a Service (IaaS), Platform as a service (PaaS), Software as a Service (SaaS) and Data as a Service (DaaS). One of the most remarkable characteristics of cloud computing provided by IaaS is that users can provision computing resources to run computing tasks for maximum efficiency and scalability in minutes, instead of spending time waiting in queue with the limited computing pool in the traditional computing

paradigm. Therefore, cloud computing is a powerful and affordable alternative to run large-scale simulation and computation that are computationally intensive (Huang et al., 2013b). However, few studies have explored leveraging cloud computing to support responsive, massive, on-demand models, especially for supporting the computing- and data- intensive climate simulations.

#### **1.4 Objective and Summary of Contribution**

Driven by the aforementioned question, the objective of this dissertation is to optimize geospatial cyberinfrastructure by tackling the three computational challenges facing climate science through three researches. Cloud computing is adapted as a base computing infrastructure, and the concept of decomposition and parallelization is seamlessly integrated across the entire research. The optimized geospatial cyberinfrastructure enables scientists efficiently conduct model sensitivity analysis by 1) conducting hundreds or thousands of model-runs in parallel, 2) analyzing terabytes of model output in parallel, and 3) conducting complex experiment in a simple drag-and-drop style.

The theoretical and technological contributions of the dissertation are detailed in Chapter 2, 3 and 4 followed by experiment and result discussion in each chapter to verify the feasibility/performance of proposed methodologies. More specifically, the contributions are summarized as follows:

**Model as a Service for computing intensity:**



In Chapter 2, Model as a Service, a new service model in the context of cloud computing, is proposed to tackle the computing intensity challenge posed by climate simulation. MaaS automatically invokes ready-to-go modeling environment backed by adequate and scalable computing resources for model simulation, and automatically release the occupied computing resources once the simulations are finished. Specifically, with MaaS:

- Climate models are published as services, and these services can be accessed through a simple web interface.
- Climate simulation parallelization are achieved in both ensemble level and concurrent requests level
- Computing resources for climate simulation are rapidly provisioned and released as needed in cloud.
- MaaS provides a collaborating environment for developing, testing and sharing models.

The proposed MaaS bridges the computing infrastructure and computing requirements of climate modelling, and this can be easily broaden from the climate sciences to other GIScience domains. MaaS provides new insights and guidance for seeking solutions to address the computing demands for model simulations in an innovative way and taking advantage of cloud computing.

**Scalable spatiotemporal data analytics for data intensity:**

In Chapter 3, a scalable big spatiotemporal data analytics framework is proposed to efficiently manage and process the big model output. Compared to other existing solutions, the proposed framework has the following significant advantages:

- The spatiotemporal decomposition mechanism enables the management of multidimensional climate data in a distributed environment.
- The MapReduce-enabled parallelization framework supports on-demand spatiotemporal querying and parallel processing of big climate data.
- The auto-scaling algorithm intelligently adjusts the computing power of the analytics framework to maximize the processing performance while minimizing the resource utilization.

By exploring the decomposition mechanism and parallelization technique for general spatiotemporal data, this framework employs MapReduce, No-SQL database and cloud computing to efficiently process big climate data in an auto-scalable computing environment. Such a framework can be applied to tackle big data challenges in broader GIScience domains.

#### **Service-oriented cloud-based scientific workflow for procedure complexity:**

In Chapter 4, the service-oriented cloud-based scientific workflow framework enables scientists to conduct the complex experiment (study case experiment) by dragging and connecting steps in a simple visually intuitive diagram. Compared to other existing workflow solutions, the proposed workflow framework has the significant contributions:

- The unified service model defines four types of basic services involved in a scientific experiment: process service, data service, model service, and infrastructure service, and enables these services to be chained together in a unified manner.
- By incorporating cloud computing, the workflow framework enables on-demand provisioning computing resources during the workflow execution.
- The service-oriented mechanism opens the framework, allowing scientists to collaborate by publishing their own services and workflows in a web-based environment.

This workflow framework tackles the challenge of procedure complexity. In addition, this framework enables MaaS (as model service) and scalable climate data analytics (as process service) to be seamlessly integrated to build an executable workflow, which further relieve scientist from computing issues and facilitate scientific discoveries.

The contributions of this dissertation optimize the commutability of geospatial cyberinfrastructure by tackling the three computational challenges facing in climate studies. With the optimized geospatial cyberinfrastructure, the complex experiment of the study case is transformed into an executable workflow, which reduces the experiment time from 2 months to 5 days with a few clicks. This research also provides a valuable guideline on bridging the computing infrastructure and computing requirements for climate studies. Since the challenges of computing intensity, data intensity and procedure

complexity are quite common in geosciences, the proposed optimization methodologies can be broaden from the climate science to broader geoscience domains.

## CHAPTER 2 MODEL AS A SERVICE

### 2.1 Introduction

To address the computing intensity challenge of climate modeling, I propose Model as a Service (MaaS) by leveraging the latest advancement of cloud computing.

*Definition: MaaS is a service model in the context of cloud computing, aiming to facilitate model simulation by publishing modeling related processes as web services in the cloud environment.*

The “Service(S)” in MaaS refers to three aspects: first, it refers to web service, because the model simulation related processes are published as web services. Second, it refers to cloud service, because the model simulation required computing resources are obtained from the cloud services. Third, it refers to Everything as a Service (XaaS), serving as a new service model in the context of cloud computing.

MaaS is applicable to broader GIScience domains in that it enables various geospatial models (e.g. climate model) to be published as services, and these services can be accessed through a simple web interface. MaaS automates the processes of configuring machines, setting up and running models, and managing model outputs. The computing resources are automatically provisioned by MaaS in a cloud environment. This chapter details the design and implementation of MaaS. Section 2.1 reviews relevant research whereas Section 2.2 details the methodologies. A proof-of-concept prototype is

presented in Section 2.3 with experimental results demonstrating the feasibility of the MaaS. Finally, Section 2.4 draws conclusions and discusses future relevant research.

## **2.2 Literature Review**

### **2.2.1 Model Web**

The concept of model access through web services, Model Web, has been proposed to facilitate model accessibility and interoperability. Geller and Turner (2007) first defined Model Web as an open-ended system of interoperable computer models and databases, with machine and end-user Internet access *via* web services. They envisioned that through Model Web geoscience communities would work as a system of independent but interactive models in three phases: 1) data interoperability for handling data heterogeneity; 2) ontology to address the domain heterogeneity; and 3) automation with the web portals for model accessibility and integration. Notable efforts have made progress toward this vision. For example, the climate dynamics community has developed modeling systems reflecting the interaction of Earth subsystems, such as ocean and atmosphere (e.g. the METAFOR project<sup>4</sup>, Nativi et al., 2013). Geller and Melton (2008) applied an ecological web model to assess the impacts of climate change. Nativi et al. (2013) further introduced the GEO Model Web (GMW) initiative to increase environmental model access and sharing. Model interfaces have been designed and tested as web services (Goodall et al. 2011) to improve the accessibility of model output (e.g. PCMDI: Program for Climate Model Diagnosis and Intercomparison<sup>5</sup>). Roman et al.

---

<sup>4</sup> <http://metaforclimate.eu/>

<sup>5</sup> <http://www2-pcmdi.llnl.gov/>

(2009) described the MaaS concept as the evolution of Model Web without detailing how such a concept could be implemented with the latest information and computing techniques. Addressing the requirements to make models and their outputs configurable, accessible and interoperable, Model Web has gained recognition by the geoscience community as a useful tool to build geoscience models, combining individual components in complex workflows (Bastin et al., 2013).

The three-phase vision of the Model Web has been largely fulfilled by previous research. However, those visions primarily focused on model integration and interoperability across various disciplines. In addition, previous researches (Geller and Turner, 2007; Geller and Melton, 2008; Roman et al., 2009; Goodall et al., 2011; Nativi et al., 2013) utilized open-standards to achieve model interoperability and have a common limitation of avoiding computational issues. In fact, the underlying computing infrastructure for running these models are barely mentioned, leaving computational challenges such as computing intensity, data intensity and disruptive computing requirement unexplored.

### 2.2.2 Computing technologies for computing intensive model simulations

Traditionally, supercomputers or large scale Grids, such as TeraGrid (Beckman, 2005) and Open Science Grid (Pordes et al., 2007), have been used to address computational challenges for geoscience models (Bernholdt et al., 2005; Chang et al., 2008b; Fernández et al., 2011; Yang et al., 2011b). However, these computing facilities require dedicated hardware and software with significant upfront investment (e.g., years to assemble). Only a few large projects have access to these supercomputers. An

alternative approach is to build a loosely coupled cluster from computing resources of volunteers using BOINC (Anderson, 2004), such as SETI@home (Anderson, 2002). While such a volunteer-based computing environment works well with computing intensive projects, it is less functional for projects with Big Data (e.g., global climate simulation) and a finite deadline due to limited bandwidth resources. As a result, the availability and accessibility of computing resources for traditional computing paradigms hamper the advancements of geoscience and geospatial technologies.

Cloud computing offers a powerful and affordable alternative to run large-scale simulations that are computationally intensive (Huang et al., 2013c). Many studies have been conducted to explore the feasibility of utilizing cloud computing for geoscience models and how to best adapt to this new paradigm (Vecchiola et al., 2009; Huang et al., 2010; Ostermann et al., 2010; Yang et al., 2011b; Yang et al., 2013, Yang et al., 2014). Evangelinos and Hill (2008) concluded that cloud computing could provide a potential solution to support atmosphere-ocean climate models, and Huang et al. (2013) utilized cloud computing to simulate dust storms. However, few studies have explored leveraging cloud computing to support responsive, massive, on-demand models, especially for supporting the computing- and data- intensive geospatial science simulations.

Focusing on cloud computing services, this chapter extends the traditional Model Web concept by integrating the latest cloud computing services (e.g., on demand and elasticity) to handle the model configuration, computing intensive and data intensive challenges. Different from the previous Model Web concept, which focuses on model integration and interoperability, the proposed MaaS is associated to Everything as a



Service (XaaS) in the context of cloud computing. Through incorporating cloud computing, we move Model Web to a new phase to address the computability challenges over the Internet for climate science and general GIScience communities.

## **2.3 Methodologies**

### **2.3.1 MaaS in the cloud**

The concept of MaaS is a viable mechanism to easily access, interact and run complex climate models and to manipulate model output through simple web interfaces. However, it is challenging to build MaaS with traditional computing infrastructure such as grid computing or HPC. First as a web service, it is normal to expect concurrent model runs from many end users; with traditional computing infrastructure, the underlying computing resources need to be provisioned with a large fixed number of physical machines to accommodate peak requests, which are not feasible or cost-effective. Second, many geoscience models run in a sequential mode and utilize only part of a machine's resources. Under such circumstances, high-end computers could not accelerate the simulation; instead, a low-end computer (e.g. one-core CPU) is more cost effective for such models. Third with traditional computing infrastructure, models are tightly coupled with the physical computing infrastructure, and the whole model execution and data manipulation are directly deployed on the machines. Removing existing models from or adding new models to such modeling environment is complex and time consuming.

Cloud computing (especially IaaS) is a promising technology to address this challenge for several reasons. First, IaaS provides scalable computing resources to handle

concurrent and ensemble runs. Second, the specifications of a virtual machine (VM) (e.g., such as the number of CPU cores and the size of RAM) can be efficiently tailored for a specific model run based on its resource consumption characteristics. Third, IaaS provides an image-based mechanism, allowing the model environment to be “burnt” to a VM image (i.e., snapshot of a computer software system), which serves as the foundation of MaaS. PaaS is not suitable for building MaaS because PaaS does not allow users to manage or control the VM’s operating system or storage/network (Mell and Grance, 2009). Based on these considerations, we propose to build MaaS on IaaS, serving as a new service model of cloud computing.

The general framework of MaaS consists of four layers: cloud computing platform, geoscience model image repository, MaaS middleware and MaaS users (Figure 2.1). As one of the most popular and matured IaaS platforms, Amazon EC2<sup>6</sup> is used herein to demonstrate the idea. Eucalyptus<sup>7</sup> is an open source cloud platform compatible with Amazon EC2. Therefore, the methodology presented in this research can be applied to building MaaS either on EC2 or Eucalyptus or other EC2-compatible platforms.

---

<sup>6</sup> <http://aws.amazon.com/ec2>

<sup>7</sup> <https://www.eucalyptus.com>

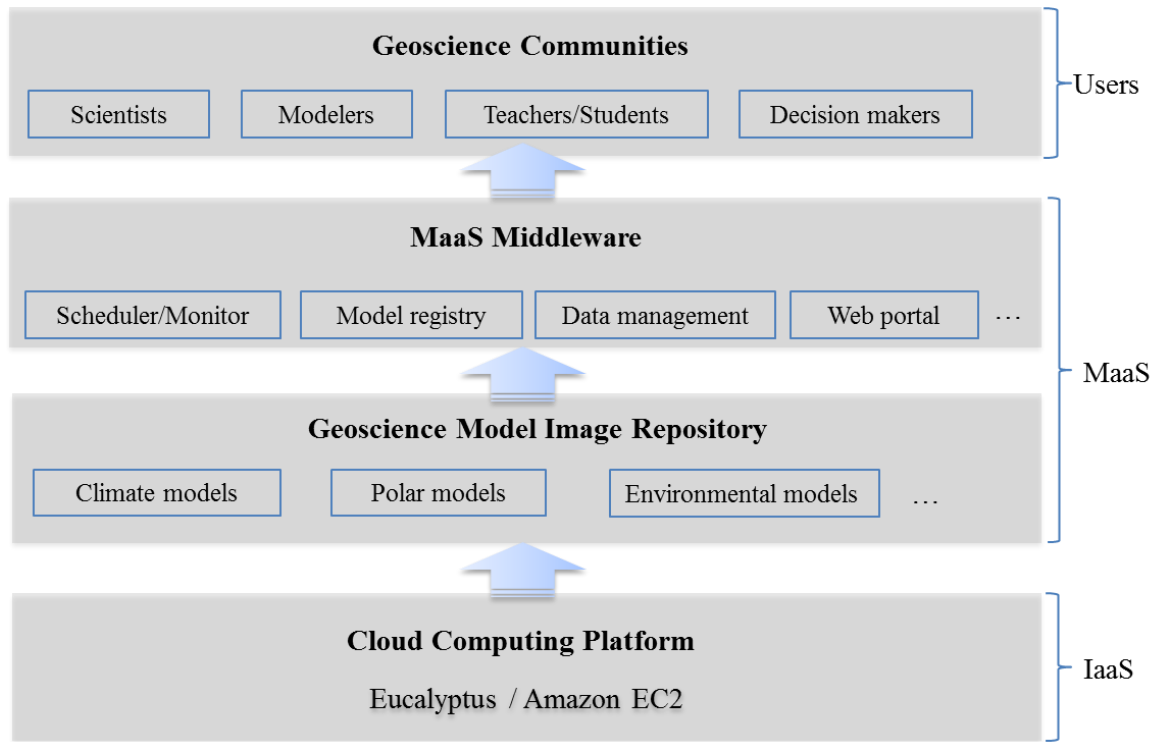


Figure 2.1 General framework of the cloud-enabled MaaS

### 2.3.2 MaaS architecture

The overall architecture of MaaS enables users to submit a model run request, monitor the model run status and finally get the model run result through a web interface (Figure 2.2). The tasks of provisioning machines and setting up and running the model are automated by MaaS.

*MaaS Engine*, powered by cloud computing, compiles and runs models on the *Model VMs*. A *Model VM* is a ready-to-go environment for a specific model, including the model code, dependent software libraries and other configurations. The *Model VM* is provisioned based on the *Model VM Image*, which is a system snapshot containing the

whole required software environment to run a model except for the model configuration file and input data uploaded by the user for flexibility. All images are managed by MaaS in the cloud platform and are downloaded upon request. Once a *Model VM* is provisioned, the following commands will be executed: upload the model configuration file and input data, compile the model, run the model, preprocess model output, and upload the output to the data server. These tasks are controlled by *MaaS Server*.

*MaaS Server* is a virtual agent that dispatches and monitors the model runs as well as manages the underlying computing resources automatically. It is responsible for the following: 1) interpreting model run requests using the *Request Interpreter* and dispatching the tasks using *Task Controller*; 2) controlling the life cycle of *Model VM* using cloud API based on the model execution status; and 3) monitoring task execution status using the *Task Monitor*, a background program which periodically communicates with all *Model VMs* to fetch the latest model execution status (e.g., model running time and size of data produced).

*Data Server* archives and manages all model outputs and related metadata (e.g., model run parameters, output data format) in a centralized database. These data can be accessed by other researchers either by downloading *via* ftp or visualizing online. The data uploading process is controlled by *Task Monitor*, and the data uploading strategy is discussed in Section 2.3.6. *Web Portal* provides the graphic user interface (GUI) for MaaS, allowing users to register/login to MaaS, configure and submit model runs, check the model run status and perform other tasks. By integrating existing online visual analytic systems with *Data Server* and leveraging spatial web portal technologies (Yang

et al., 2006; Li et al., 2011), the *Web Portal* further supports model output visualization and analysis .

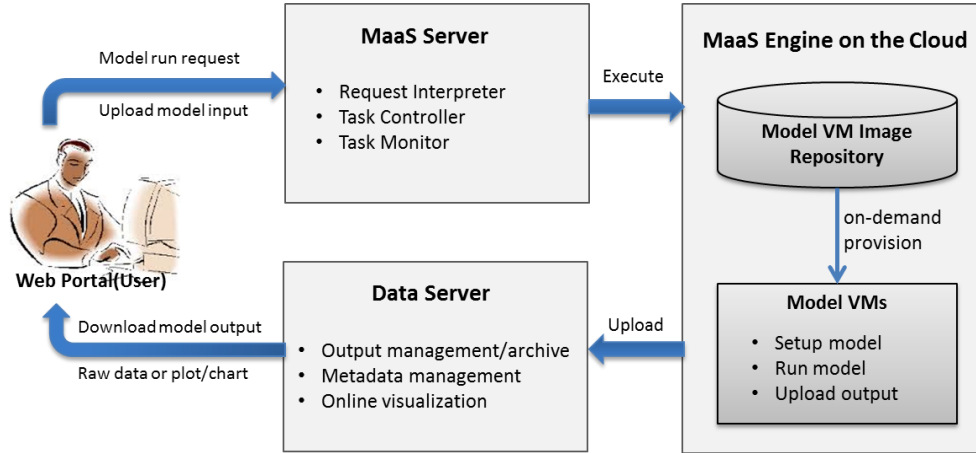


Figure 2.2 Overall architecture of MaaS

### 2.3.3 Model I/O

Each geoscience model is a processing unit producing output with specific input. Handling model input and output (I/O) could be very complex for MaaS because the input types and formats are heterogeneous for different models. For instance, some models take one configuration file as input such as ModelE, while others may require multiple configuration files and data files such as the NMM-dust model (Xie et al., 2010). In order to handle the heterogeneity, we abstract the model input as two general types: configuration input (e.g., geographic region, spatial resolution, and time period) and data input (e.g., observation data and initial condition data). Model outputs are (all data files produced by the model) in different formats as denoted in Expression 2.1.

$$Model(ConfigInput, DataInput) \xrightarrow{run} DataOutput \quad \text{Expression 2.1}$$

This abstraction enables us to define three standardized “channels” (*ConfigInput channel*, *DataInput channel* and *DataOutput channel*) for each model VM to receive the model input from *MaaS Server* and to upload model output to *Data Server* (Figure 2.3). A “channel” refers to a specific *Model VM* location (directory) where the model input files or output files are placed. The three “channels” are specified when creating the *Model VM Image* and registered in *MaaS Server*. HTTP protocol is used for transferring data between model VM, MaaS server and data server. This model I/O mechanism addresses the heterogeneity problem at the framework level by encapsulating the data details (types and formats) to a traditional level that can be dealt with by, for example, OGC web processing services. The interoperability achieved by this I/O mechanism enables MaaS to support various geoscience models transparency in a unified framework. New models can be added to MaaS as long as the channels are properly defined. Section 3.4 details the mechanism for publishing new models.

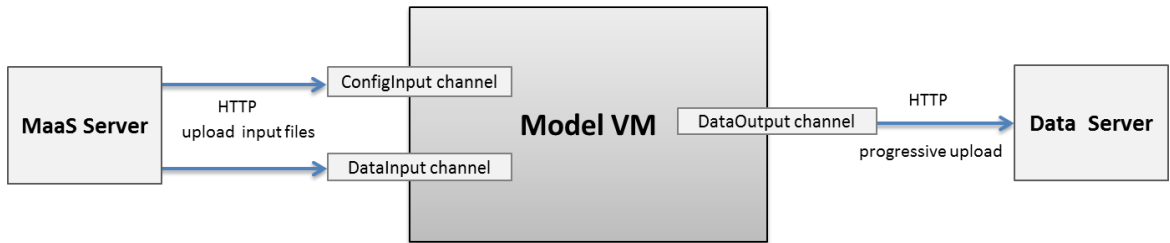


Figure 2.3 Model I/O data transfer between model VM, MaaS server and data server

#### 2.3.4 Mechanism for publishing new models

Supporting an array of geoscience models is an essential capability of MaaS. Even though different models may have different model run environments and model input and output, the procedure to setup and run different models is similar, starting from selecting hardware, to configuring the model run environment and handling model output (Liu et. al 2013b). The MaaS architecture provides a pluggable mechanism to enable new models to be easily plugged into the existing MaaS system with two general steps: 1) creating a model VM image for this new model; and 2) registering the new model VM image into MaaS.

To create a model VM image, software and OS requirements are analyzed (e.g., CentOS, Ubuntu) after which a “bare-metal” VM is launched with supported OS in a cloud platform (any clouds that are compatible with Amazon EC2/Eucalyptus) using a basic VM image. The image is available from the cloud provider (e.g., Amazon) or the cloud platform community (e.g., Eucalyptus). Following VM launch, the model and dependent software libraries are installed following the same procedure as on a physical machine. The next step configures the model (configuration file and input data) and conducts a test run. If the run is successful, four steps are initiated: extract model configuration file(s) and input data required for a model run and delete them from current VM; identify the three channels for the model VM; and create the model VM image based on current VM using the built-in cloud APIs.

Three steps are needed to register the new model VM image in MaaS. First, the new model VM image is uploaded to the MaaS cloud platform. Second, the three channels are registered in MaaS server to indicate where model input and output files should be put on the model VM. Third a new *Request Interpreter* is configured and plugged into MaaS server to parse the model run request for this new model.

The model VM image can be published as a model in MaaS using the built-in cloud APIs (Eucalyptus or Amazon EC2). Registering the new model (VM image) into MaaS is conducted by the MaaS provider. Collaborations between the modelers/researchers and MaaS providers are critical for publishing a new model into MaaS as the process is complex. However, once the model is published, these complex steps are “recorded” and a ready-to-go model environment could be provisioned in minutes.

#### 2.3.5 Mechanism for parallelizing ensemble model run

Ensemble run of a model is a normal practice in geoscience modeling for testing the model sensitivity to input parameters. For example, to test a climate model’s sensitivity to a set of parameters, ensemble runs of the model are conducted hundreds of times with different parameter combinations for which we have observational data. With the traditional model infrastructure such as HPC, an ensemble run can be conducted either via installing the model on one machine and conducting the ensemble runs sequentially, or installing the model on many machines and conducting the runs in parallel. Neither approach is effective.



MaaS can conduct an ensemble run in parallel with a single request (Figure 2.4). The model configuration and input data for each model run are uploaded by users using the web interface through the “channels”. For example, users zip all model configuration files (.R file) in a single package and upload to MaaS. Upon receiving the ensemble run request, MaaS provisions a model VM for each model run concurrently. The uploaded configuration files and input data are distributed to the VMs. After the model runs start on each model VM, model outputs are post-processed and uploaded to the data server for download or visualization. For each finished model run, the model VM is terminated to minimize the resource consumption.

With this parallel mechanism, a large ensemble run can be parallelized by provisioning the same number of model VMs as the number of model runs if the computing pool is large enough (e.g., Amazon EC2). If the computing pool is limited (e.g., private cloud), MaaS enables users to specify how many model VMs to be provisioned. In this case, MaaS (task controller) maintains a wait list for the pending model runs, and a new run is started once a model VM finishes. This parallel mechanism is cost- and performance- effective because conducting 100 model runs on one machine for 100 hours equals the same on 100 machines for 1 hour assuming a 1 hour run per machine.

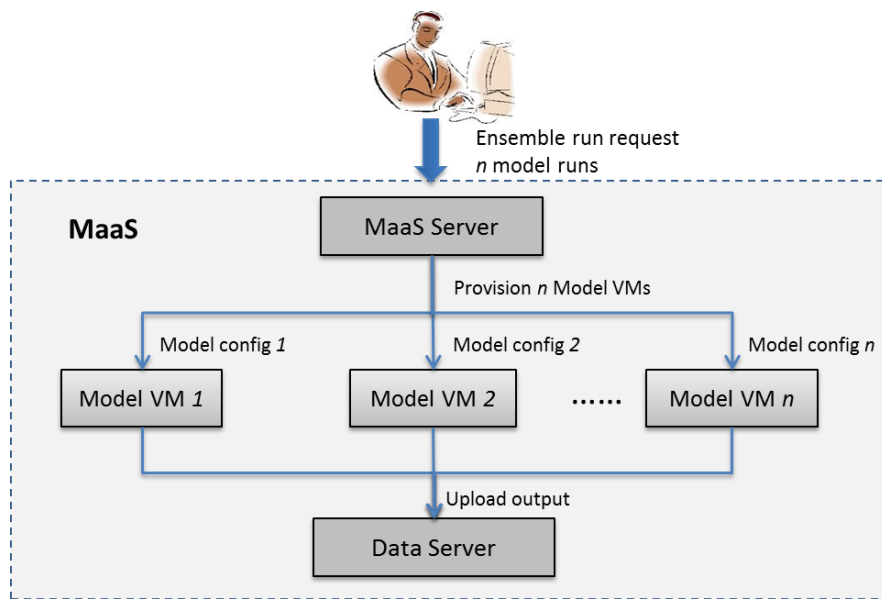


Figure 2.4 Mechanism for parallelizing ensemble model run (model VMs are provisioned based on the same Model VM Image but running with different configurations)

The parallelization for ensemble model run occurs at the experiment-level, which allows conducting many model runs concurrently to accelerate the ensemble experiment. The code-level parallelization of enabling a model to run in parallel using the parallelization technologies such as MPICH2<sup>8</sup> is well studied and beyond the scope of this paper. However, if a model is already MPI-enabled, such as the dust model NMM-dust (Xie et al. 2010), such a model can be incorporated into MaaS, and a cloud-based HPC cluster will be provisioned to support running the model. This is discussed in the future research section.

<sup>8</sup> <http://www.mpich.org/>

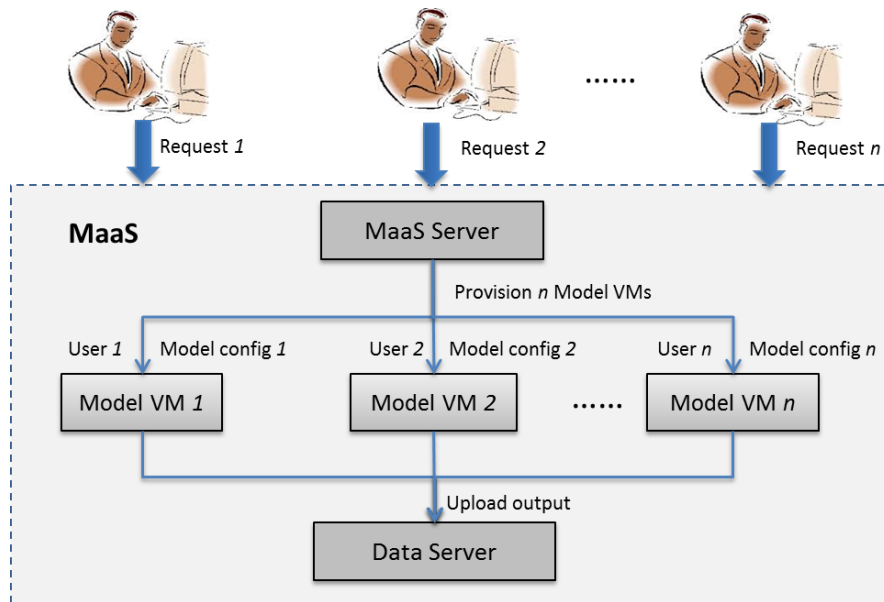


Figure 2.5 Mechanism for concurrent user requests (model VMs provisioned based on the different Model VM Images and running different models configurations)

In order to support concurrent requests, MaaS uses similar mechanism (Figure 2.5). If  $n$  users run a model concurrently, MaaS starts  $n$  model VMs (if the computing pool is large enough) with each VM running these requests concurrently. When the computing pool is not large enough to provision more model VMs, MaaS manages a waiting list. If a model VM has finished a current model run, MaaS terminates this VM and starts a pending model VM using a first-come first-serve policy. It should be noted that each user request can also be an ensemble run request; in this case the parallelization is achieved at the levels of both the user request and the ensemble run.

### 2.3.6 Mechanism for handling model output

Each model run generates gigabytes of data, and this volume increases to terabytes and petabytes levels in many model runs. Often, model output needs to be post-processed (e.g., format conversion, aggregation) before being managed and accessed. In the distributed simulation environment, each model output also needs to be uploaded to the data server for integration. This poses at least three challenges: post-processing is computing intensive; uploading to the data server is communication intensive; and the storage for the *Model VM* needs to be large enough to hold the output.

Even though a model simulation produces large volumes of data, these data are not routinely produced and stored on the disk together. Instead, data are generated progressively and written to a disk individually (dataset). For different models running on multiple VMs, this procedure occurs in a distributed manner. We introduce a progressive mechanism to accelerate data post-processing and uploading: post-processing data distributed on the model VMs where data are generated; post-processing progressively as data are produced; and uploading data progressively once the process is complete. Thus, when a model run is finished, most of the model output has been post-processed and uploaded to the *Data Server*.

This progressive mechanism has several benefits. First, the wait time for post-processing and uploading is significantly reduced. Second, the computing resource consumption is minimized by terminating the VMs almost at the same time when the model run is finished. And third, the VM storage requirement for storing model output is significantly reduced.

## 2.4 Evaluation

A proof-of-concept MaaS prototype is developed to evaluate its feasibility and performance.

### 2.4.1 Experiment setup

The following is a model run scenario demonstrating how ModelE is used by climatologists. A researcher examines how climate responds to small changes in the Earth's absorption of solar radiation by simulating 14 different scenarios with different levels of chlorine in the atmosphere (Figure 2.6A). In this workflow, completing the first three steps is a cumbersome task. In step 4, running models sequentially requires longer run time, while parallel model runs need more efforts to set up the model. Furthermore, the model takes hours to days to finish a single simulation, assuming unbridled access. Once the simulations are finished (Step 6), all model outputs need to be integrated.

The following sections introduce the MaaS prototype for ModelE and analyze the effectiveness of this prototype for this scenario.

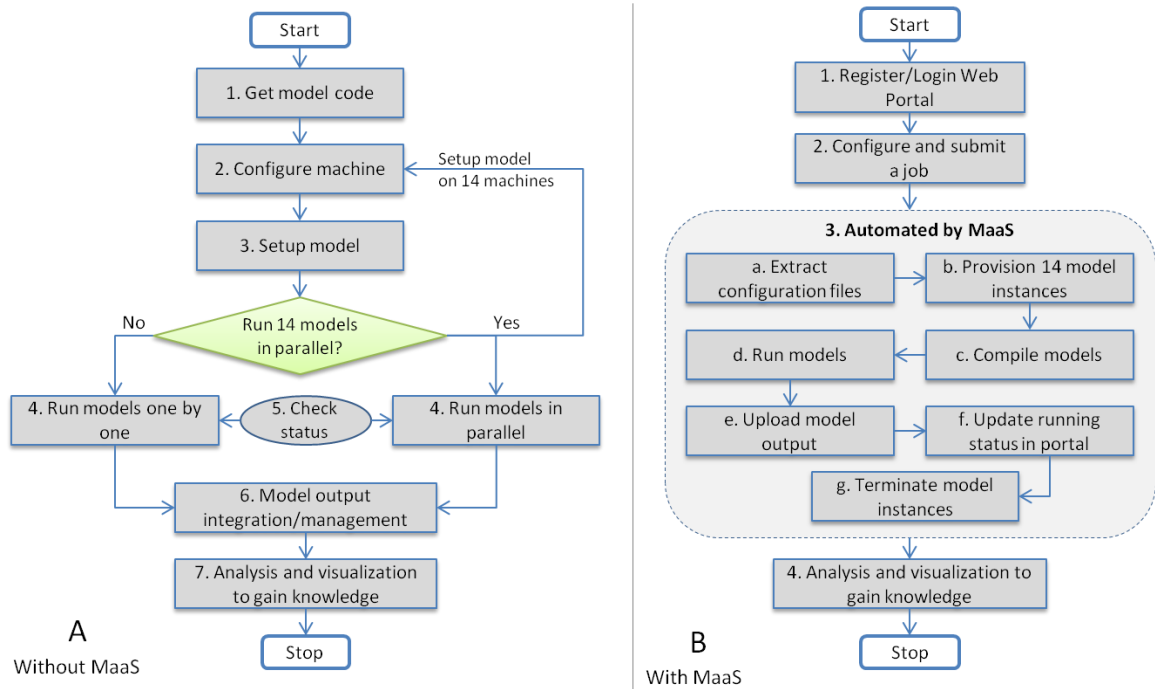


Figure 2.6 Workflows for simulating and analyzing 14 different scenarios using ModelE: A) without using MaaS and B) using MaaS

## 2.4.2 Prototype

A private cloud platform is established on Eucalyptus version 2.0<sup>9</sup>, serving as the cloud environment for the prototype. The underlying hardware for this private cloud are six physical machines (8-core CPU running at 2.35 GHz, 16 GB of RAM) connected with 1 Gigabit Ethernet (Gbps). Totally 40 VMs (1 core CPU running at 1 GHz and 2G of RAM) can be provisioned in the cloud. The ModelE VM image is built based on the Linux Ubuntu 4.3 using Eucalyptus API (Figure 2.7).

<sup>9</sup> <http://www.eucalyptus.com/>

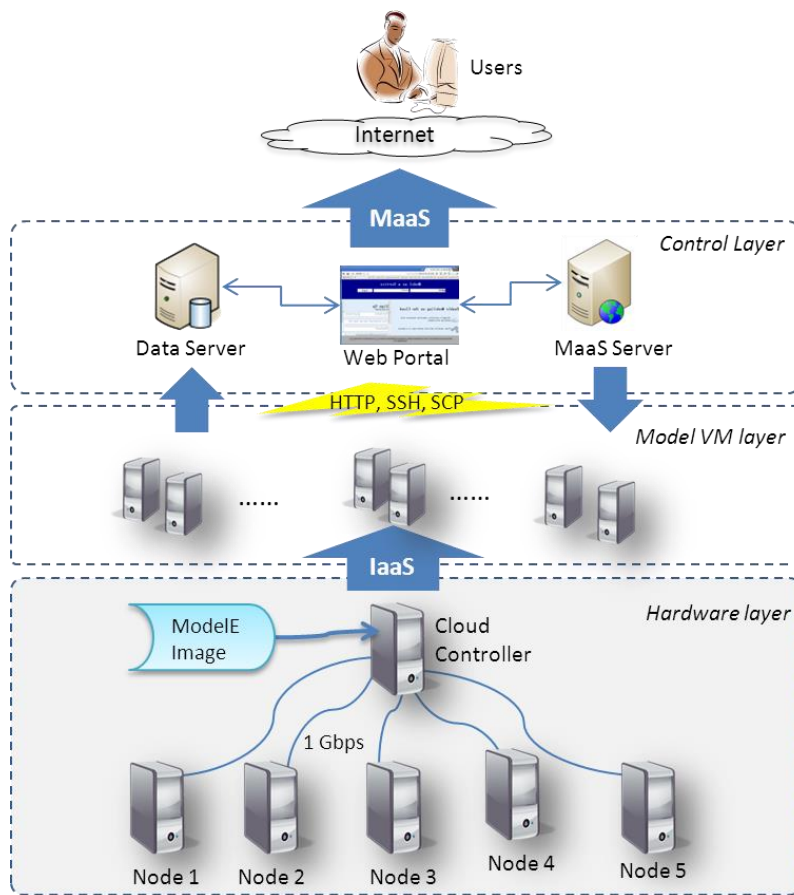


Figure 2.7 Physical architecture for the MaaS prototype

Figure 2.8A shows the *Web Portal* GUI once a user logged in. The running status for the submitted tasks of this user is detailed in a dynamic table in an interactive fashion (e.g., terminating a task, viewing model-run configurations and exploring model output). An existing web-based visual analytic system optimized by various performance-improving techniques (Sun et al., 2012; Li et al., 2013) is incorporated into the *Data Server* and integrated with the *Web Portal*, enabling users to visualize the model output

directly on the web without downloading the data. The original model output can also be browsed and downloaded from *Data Server* (Figure 2.8B).

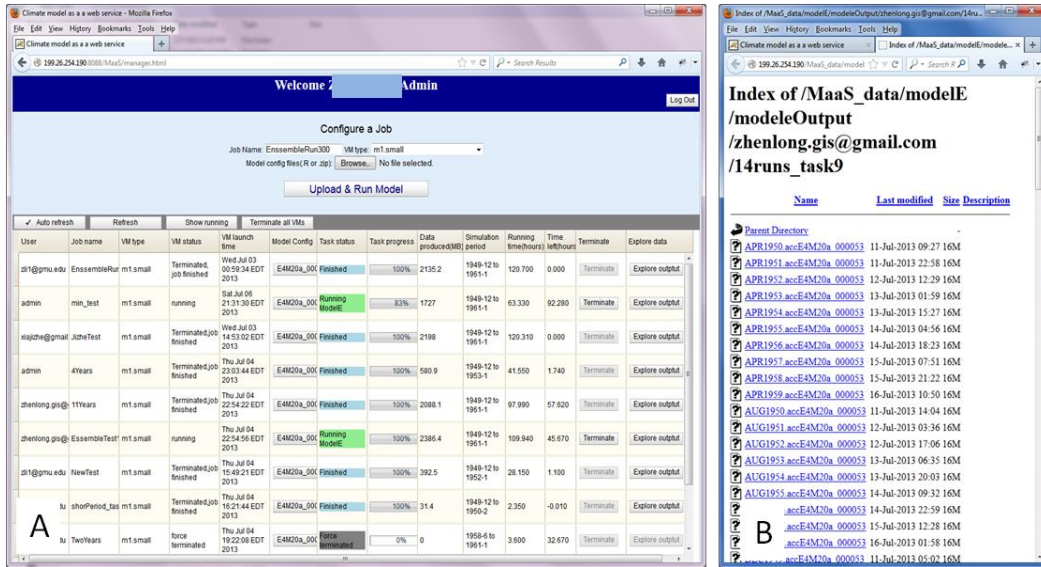


Figure 2.8 (A) The Web Portal GUI showing the running status of the submitted tasks; (B) Model output progressively transmitted to data server to ensure data availability

## 2.4.3 Result

This section evaluates the efficiency of MaaS by illustrating how MaaS helps scientists from three aspects: model setup, model run, and model output handling.

### 2.4.3.1 Evaluation of model setup

An experiment is conducted to evaluate the value of MaaS by comparing the setup of ModelE with and without MaaS. Ten users with different backgrounds were selected (denoted *User1*, *User2*, ..., *User10*). The users were provided with the same MaaS



user manual and ModelE configuration file (.r file), and each setup and ran ModelE with the configuration file using MaaS. Of the ten, only *User1* and *User2* manually set up ModelE because this task is complex and only is needed when the model is first setup in the MaaS. Both *User1* and *User2* have a geoscience background, and *User1* and *User 2* have high-level and limited-level Linux administrative skills respectively. The two users are provided with the same installation guide for ModelE and the same Linux machine.

The time spent for setting up the model by each user (Table 2.1) shows that *User1* and User 2 spent 1.2 and 2.3 hours respectively. Conversely, with MaaS, all ten users required < 4 minutes for configuring and submitting the job. Once the job was submitted, MaaS provisioned the model environment (starting an instance, transferring input files, and compiling and starting) in < 4 minutes. Thus, MaaS not only reduced the user interaction time from hours to minutes for setting up the model, but also allowed those with little system administrative experience to proceed expeditiously.

Table 2.1 Comparison of time spent on setting up ModelE environment with/without MaaS (MaaS usage does not include the time for registering ModelE into MaaS)

<b>Manual setup</b>		<b>MaaS</b>	
	User interaction time (h)	User interaction time (min)	Auto provision time (min)
<i>User1</i>	~ 1.2	1.3	4.3
<i>User2</i>	~ 2.3	1.5	4.2
<i>User3</i>	--	1.9	4.2
<i>User4</i>	--	1.5	3.9

<i>User5</i>	--	2.8	4.3
<i>User6</i>	--	1.9	4.2
<i>User7</i>	--	1.6	4.1
<i>User8</i>	--	3.4	4.5
<i>User9</i>	--	1.5	4.2
<i>User10</i>	--	2.3	4.0

#### 2.4.3.2 Evaluation of model run and model output handling

- Many-model-run performance

The capability and reliability of provisioning multiple model instances to support many-model-run in the usage scenario was evaluated in another experiment by provisioning 14 model runs with 14 different input configurations. These model input configurations came from 300 ensemble runs provided by NASA scientists. The simulation period was from Dec., 1949 to Jan., 1961 with a spatial resolution of  $4^0 \times 5^0$  and a time resolution of monthly. Figure 9 illustrates the timeline for the 14 model runs, and Figure 10 shows the running status in the *Web Portal*. All model instances were successfully provisioned (m1.small instance type) and run with ModelE within 6 min (Figure 2.9). The 14 model runs were finished in~ 5 days, and model instances automatically terminated < 10 seconds after completion of the model run.

- Model output handling

Each run generated 2088 Megabytes (MB) of data, and a total of 28.55 GB data were produced and uploaded. Monthly data were generated by ModelE and the size of each dataset was ~ 16 MB. The progressive data uploading mechanism ensures that when a model run is finished, an instance needs to wait only for uploading the last month dataset (16MB) before termination, ~ 4 seconds in the testing environment. Once model runs are finished, users analyze, visualize and compare model outputs directly through the integrated online visual analytic system without downloading and managing the data.

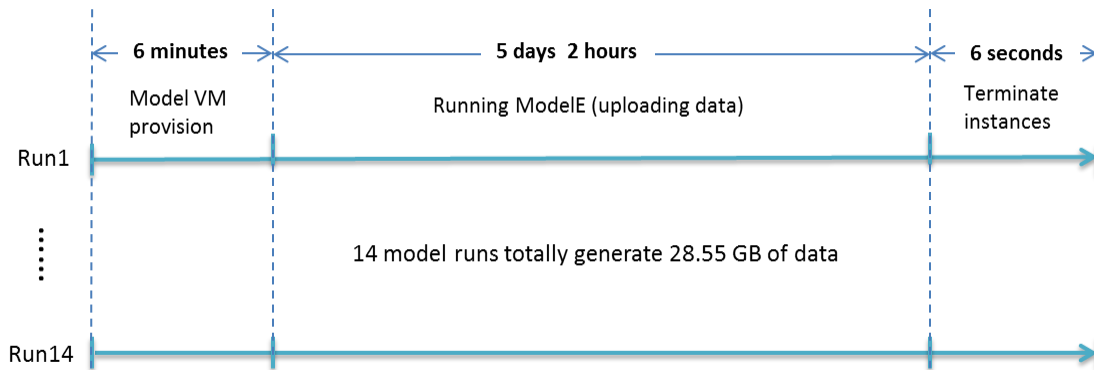


Figure 2.9 Timeline for provisioning and running the 14 model runs with a 10-year simulation period. X-axis is time spent on the three stages, while y-axis is the 14 model runs (for better visualization, the timeline is not evenly scaled)

As showed in Figure 2.6B, with MaaS, the researcher manually conducts Steps 1 and 2, which take a few minutes. The complex and time consuming tasks such as configuring machines, installing and compiling models are automated by MaaS (Step 3). The large computing resources required by model runs are rapidly provisioned by MaaS. Furthermore, all model outputs are managed by MaaS and the researcher directly

analyzes the simulated data using the online visual analytical system. Based on the above experiments, it is concluded that MaaS expedites significantly model setup, model run and model output management.

#### 2.4.4 Key features of MaaS

The key features of MaaS are summarized. The challenges addressed by the key features are summarized in Table 2.2.

- (F1). New geoscience models are published to MaaS by creating new model VM images and registration in MaaS. This feature is enabled by the pluggable approach in the MaaS design and the image-based mechanism offered by cloud computing.
- (F2). The model configuration files and model input data are prepared outside of MaaS and then uploaded to the model VM through MaaS using three standard “channels”. This flexibility enables researchers to run a model many times with different configurations without rebuilding the model environment.
- (F3). Once a model VM image is created, if a modeler needs to modify the model code, she/he can easily create a new version of the VM image containing the modified model based on the existed model VM image. Once the model VM image is updated, anyone who requests to run this model will view the latest model environment. This feature enables modelers to easily test and propagate their models.

- (F4). The specifications of the model VM (e.g., CPU, RAM and disk) are dynamically specified by the user when submitting the model run request based on the resource consumption characteristics. This flexibility ensures that the model VMs are tailored for specific models, which maximizes the load on each model VM while ensuring the computational efficiency. For example, if a model can only utilize one CPU core, a small VM instance with one core CPU is used.
- (F5). The model VM image is downloaded directly from MaaS, allowing users to quickly run the model in other cloud environments, such as their own private cloud platform (needs to be EC2 compatible) or Amazon EC2. This feature enables the models to be “installed one time, run many times on any compatible platforms”.
- (F6). Ensemble model runs are effectively handled by provisioning many model VMs simultaneously and conducting these model runs in parallel. Users prepare the model configuration files, upload the input data and submit the request; MaaS handles the remaining tasks.
- (F7). Concurrent model run requests are handled by provisioning many model VMs simultaneously with each VM dedicated to one user (if the computing pool is large enough). If the computing pool is not large enough to handle all requests concurrently, MaaS handles the requests following a “first-come, first-serve” policy.

- (F8). Progressive data transmitting mechanism enables the model output to be “pushed” to the user as soon as they are available and also dramatically reduces the waiting time for data uploading to data server.
- (F9). The model run request together with model configuration and input data are submitted directly from a web interface. The progress for all model runs is periodically updated in the web interface. Finally the model outputs are downloaded from the web interface. This feature enables users to conduct scientific research in a web-based environment.

Table 2.2 Geospatial Modeling Challenges addressed by key features of MaaS

<b>Challenges\MaaS Features</b>	<b>F1</b>	<b>F2</b>	<b>F3</b>	<b>F4</b>	<b>F5</b>	<b>F6</b>	<b>F7</b>	<b>F8</b>	<b>F9</b>
Model setup complexity	x	x	x		x	x			x
Computing intensive			x	x		x	x		
Scalable requirement				x		x	x		
Data intensive						x		x	x
On demand research	x	x	x		x				x

#### 2.4.5 Advantages of MaaS

The traditional computing infrastructures (e.g., HPC server) have limited availability, accessibility, and scalability. Traditionally, to run a model, researchers need to start from purchasing the infrastructure, or waiting for account approval to access

public HPC resources (e.g., SGI<sup>10</sup>). The modeling configuration, data input, preprocessing and post-processing have to be done each time the HPC is allocated.

Compared to the traditional HPC server, MaaS offers several distinct advantages. First, users access the computing resources and run the model on demand without waiting. Second, MaaS enables users to publish a new geoscience models to MaaS with a pluggable approach which is leverage-able in the future whenever the model is reconfigured or run again. Third, the HPC server is not elastically provisioned for computing resources to satisfy on-demand computing needs. If a researcher conducts a large ensemble experiment by running a model hundreds of times, a long wait time is required as the runs are sequential. Conversely with MaaS, hundreds of virtual machines are provisioned in a few minutes to simulate runs in parallel.

The IaaS serves as the fundamental platform to build MaaS, similar to IaaS serving as the foundation for PaaS and SaaS. The IaaS provides an image-based mechanism, essential for MaaS to create a ready-to-go model environment. The on-demand, elastic and scalable computing resources offered by IaaS also enable MaaS to accommodate the computing intensive challenge posed by the geoscience community. However, IaaS serves only as the underlying computing infrastructure (similar to bare-metal machines) for MaaS, and the aforementioned MaaS features are not by default provided by IaaS (except feature#4 and #5). The mechanisms of scheduling, monitoring, model registering and input/output handling are provided by MaaS.

---

<sup>10</sup> <http://www.sgi.com>

The supportability of the features by the three platforms of HPC, IaaS and MaaS are compared in Table 2.3.

Table 2.3 Supportability of the features by the three platforms (HPC, IaaS, MaaS) for key features of MaaS

<b>Platforms\MaaS Features</b>	<b>F1</b>	<b>F2</b>	<b>F3</b>	<b>F4</b>	<b>F5</b>	<b>F6</b>	<b>F7</b>	<b>F8</b>	<b>F9</b>
MaaS	x	x	x	x	x	x	x	x	x
IaaS		x		x	x			x	x
HPC		x						x	x

#### 2.4.6 User roles potentially benefited from MaaS

The proposed MaaS benefits a wide range of users across the geoscience community and other domains. For geoscientists to run a model registered in MaaS, one only needs to log onto the MaaS *Web Portal*, configure the model input, submit the model run request and monitor the running status in the *Web Portal*. The model output can be directly analyzed in the integrated online visual analytic application and shared with other team members. Hence, MaaS enables model users to set up and effectively run a model in minutes without dealing with complex model setup procedures or the required computing resources.

Geoscience modelers can publish their models to MaaS by creating and publishing VM images to MaaS with a pluggable approach. The published models are accessible by other users through MaaS. Thus, MaaS provides a new mechanism for



modelers to share their computational framework. In addition, MaaS can expedite model development in effectively validating and calibrating models by easily reusing models already installed and running model many times with different configurations and comparing the model output directly on the Web.

Finally, models help students understand important scientific processes and provide a platform to explore relevant processes (Grosslight et al., 1991; Jacobson and Wilensky, 2006). MaaS can be used as a tool in the classroom, facilitating teaching and learning. For example, by logging on the MaaS through a browser, students can easily run a model with different configurations and compare the outputs.

## **2.5 Chapter Summary**

This chapter proposes a MaaS framework to address the modeling challenges by :

- 1) publishing geoscience models as web services to hide the complexity of model setup;
- 2) providing an on-demand ready-to-go model environment, including hardware and software resources (e.g., computing, storage, and operating system (OS) with models and dependent libraries);
- 3) automatically provisioning computing resources to execute multiple model runs in parallel to support many-model-run scenarios and concurrent user accesses; and
- 4) effective handling model output for online visualization.

Methodologies for designing and implementing MaaS are presented. To test the feasibility of the framework, a prototype MaaS system is developed. Experimental results show that the proposed MaaS significantly streamlines the geoscientists' modeling activities in three steps: setting up the model, running the model and handling the model output.

## **CHAPTER 3 A SCALABLE BIG CLIMATE DATA ANALYTICS FRAMEWORK**

### **3.1 Introduction**

In climate studies, large volumes of spatiotemporal data are generated to describe and study the complex Earth's climate system. This is well demonstrated in the study case with the vast amounts of model output. It is predicted that the climate simulation and observational data hold by NASA will reach nearly 350 Petabytes by 2030 (Skytland 2012). In fact, climate science is a typical domain that represents the Big Data shift in general geospatial scientific domains (Schnase et al. 2014, Edwards 2010). Big climate data is playing a critical role in climate studies for enabling us better understand how the complex climate system works and thus predicting the future climate. Crunching and making sense of vast amounts of climate data in an efficient manner enables scientists answer key questions in climate research, e.g. to identify the ModelE sensitivity as discussed in Chapter 1.

To tackle the data intensity challenge, a scalable big spatiotemporal data analytics framework is developed by leveraging the contemporary technologies of NoSQL database, map/reduce programming model and cloud computing. Specifically, 1) a spatiotemporal decomposition mechanism is proposed to manage multidimensional climate data in a distributed environment; 2) a MapReduce-enabled parallelization framework supports on-demand spatiotemporal querying and parallel processing of big

climate data; and 3) the underlying computing resources are automatically adjusted based on the Hadoop cluster auto-scaling mechanism. This auto-scaling mechanism further enhances the data processing performance while minimizing the resource utilization.

## **3.2 Literature Review**

### **3.2.1 Database technologies for managing big geospatial data**

Over the past decades, relational databases management systems (RDBMS) (e.g., Oracle) have been used to manage a variety of scientific data including that of the climate data (Porter 2000). With RDBMS metadata are normally managed in a relational database while the actual data are stored in file systems. The data can be accessed by querying the database to find the reference (file location). While this approach takes advantage of the matured relational database technology, it is limited in terms of scalability and reliability since the data are normally archived in raw files. In fact the evolution of geospatial data has exceeded the capability of existing infrastructure for data access, archiving, analysis and mining (Wright and Wang 2011, Dongarra 2011).

To overcome the drawbacks of the traditional RDBMS, an emerging group of projects are addressing the multi-dimensional geoscience data utilizing distributed data management (e.g., Integrated Rule-Oriented Data Systems<sup>11</sup>, Climate-G testbed (Fiore et al. 2012), the Earth System Grid Federation (Williams et al. 2009)). These projects provide a grid-based framework to manage big geoscience data in a distributed environment. However, they do not draw support from cloud computing (Cinquini et al. 2012), so the resources and services can neither be initiated on demand nor meet the

---

<sup>11</sup> <http://irods.org/>

requirements of high scalability, availability and elastic of computing processes. In addition, these systems are normally complicated and bulky, making them hard to be adopted for other scientific research and applications.

NoSQL databases (Stonebraker 2010) provide a potential solution to the traditional RDBMS problems while offering flexibility to be tailored for various requirements. Over the past several years NoSQL databases have been used to store and manage big data in a distributed environment. Compared to traditional RDBMS, NoSQL database has the characteristics of schema-free, default replication support and simple API (Liu et al. 2013a). The most prevalent NoSQL databases such as HBase (Khetrapal and Ganesh 2006) and Cassandra (Lakshman and Malik 2010) are based on a BigTable (Chang et al. 2008) schema. HBase, an open source distributed database running on top of Hadoop Distributed File System (HDFS), provides high scalability and reliability by storing data across a cluster of commodity hardware with automatic failover support. Studies to harness the power of HBase to manage big geoscience data include that of Liu et al. 2013, who proposed a method to store massive imagery data in HBase by introducing two specific tables (“HRasterTable” and “HRasterDataTable”), and Chen et al. 2013 who proposed a mechanism to effectively search and manage remote sensing images stored in HBase. Unfortunately, less research attention has been focused on leveraging HBase to handle big array-based climate data (e.g., NetCDF or HDF).

To address this shortcoming, a data decomposition mechanism is proposed to manage multidimensional geoscience data with HBase in a scalable cloud computing environment.

### 3.2.2 Parallelization technologies to process big geospatial data

Data intensive geospatial analytics are becoming prevalent. To improve scalability and performance, parallelization technologies are essential (Zhang et al. 2007). Traditionally, most parallel applications achieve fine grained parallelism using message passing infrastructures such as PVM (Geist 1994) and MPI (Gropp et al. 1999) executed on computer clusters, super computers, or grid infrastructure (Foster et al. 2001). While these infrastructures are efficient in performing computing intensive parallel applications, when the volumes of data increase, the overall performance decreases due to the inevitable data movement. This hampers the usage of MPI-based infrastructure in processing big geoscience data. In addition, these infrastructures normally have poor scalability and allocating resources is constrained by computational infrastructure.

MapReduce (Dean and Ghemawat 2008), a parallelization model initiated by Google, is a potential solution to address the big data challenges as it adopts a more data-centered approach to parallelize runtimes, moving computation to the data instead of the converse. This avoids the movement of large volume data across the network which impacts performance. Hadoop<sup>12</sup> is an open source implementation of MapReduce and

---

<sup>12</sup> <http://hadoop.apache.org/>

has been adopted in the geoscience research community (Rizvandi et 2011, Chen et al. 2011, Dean and Ghemawat 2008 ).

Since Hadoop is designed to process unstructured data (e.g., texts, documents, and web pages), the array-based, multi-dimensional geoscience data cannot be digested by Hadoop. Studies have explored processing geoscience data in Hadoop. For example, Zhao et al. 2010 converted NetCDF data into text-based CDL<sup>13</sup> files to allow parallel access of massive NetCDF data using MapReduce. Although straightforward, this approach poses two issues: the transformation sacrifices the integrity and portability of the NetCDF data as well as increases the data management complexity; and the transformed data volume may increase by several times from its original volume. Duffy et al. 2012 leveraged Hadoop MapReduce to process climate data by converting the dataset into Hadoop Sequence Files, eliminating the issues that occurred in the first approach. However, all records must be fully traversed to match records since no index or query is supported by Sequence File, reducing the performance as the number of records increases.

To address this problem, a spatiotemporal decomposition mechanism is proposed to store big geoscience data in HBase. Based on the decomposition mechanism, a MapReduce-enabled framework is introduced to support on-demand accessing and processing in parallel of big geoscience data.

### 3.2.3 Auto-scaling Hadoop cluster in cloud

---

<sup>13</sup> <http://www.unidata.ucar.edu/software/netcdf/docs/netcdf/CDL-Syntax.html>

Previous studies (e.g. Gao et al. 2014, Lin et al. 2013, Krishnan et al. 2010, and Li et al. 2015) provide valuable experience and guidelines on how to adapt Hadoop to support processing geospatial data. However, such a fix-sized cluster has obvious drawbacks. First, it is neither energy- nor cost-effective since the computing resources are not fully utilized when the workload is low. Second, when the workload exceeds the computing capacity, performance is degraded. To address the first issue, Leverich et al. (2010) proposed a strategy to allow the cluster to scale down when the workload is low to improve energy-efficiency. Another approach for scaling down a Hadoop cluster is GreenHDFS (Kaushik et al., 2010), which features a multi-zone layout of hot and cold zones. The above approaches focus on removing cluster nodes, but the automation for the scaling operations is barely noted. To address the auto-scaling issue, Maheshwari et al (2012) proposed a dynamic data placement and cluster reconfiguration algorithm to turn off or on cluster nodes based on the average cluster utilization rate. However, it requires data blocks stored on one node to be transferred to other nodes before node removal. Such a process is not only time intensive but also consumes considerable amount of network resources, especially when the data to be transferred are large. In addition, the second issue is not tackled partly due to the limitation of physical resources (purchase and add a physical computer to the cluster normally requires days and weeks).

Cloud computing offers a potential solution for addressing the size-fixed cluster challenges in that a virtual Hadoop cluster can be promptly provisioned in a few minutes. Public cloud providers typically provide Hadoop cluster as web services allowing users to configure, provision and terminate the cluster through a web-based interface. For

example, Amazon Elastic MapReduce (EMR)<sup>14</sup> and Windows Azure HDInsight<sup>15</sup>, are popular cloud-based Hadoop services which enable users to quickly provision a Hadoop cluster. Once a job is finished, the users terminate the cluster and launch another cluster for additional jobs. However, a critical problem exists for these Hadoop services: the size of the cluster cannot be automatically adjusted based on the dynamic workload (e.g., many jobs submitted to the same cluster in a short time period). Even though some services (e.g., EMR) allow users to change cluster size, this is done manually; and the burden of deciding how many machines to be removed or added is on the non-administrative expert (Herodotou et al. 2011).

Actually, there is little research to study dynamically scaling Hadoop cluster in cloud environment. Römer (2010) proposed threshold-based scaling to automatically add more nodes to the cluster based on the black box performance metrics (CPU and RAM) and predefined threshold values. This scaling provisions new virtual machines when the average cluster load exceeds a threshold and then automatically configures these machines to the cluster. However, the scaling up is triggered based on the current workload. This is problematic as scaling up takes time to provision new virtual machines and configure them to the cluster (minutes to hours depending on the cloud platform). For example, it takes 20 minutes to provision a VM on Amazon EC2 and add this VM to a cluster. It is likely that the workload has changed during this time period (e.g., newly added VMs are no longer needed or more VMs are required). In addition, for scaling

---

<sup>14</sup> <http://aws.amazon.com/elasticmapreduce/>

<sup>15</sup> <http://www.windowsazure.com/en-us/manage/services/hdinsight/introduction-hdinsight/>



down Römer suggested manually terminating the nodes individually using Hadoop's built-in decommissioning mechanism.

To address these problems, an auto-scaling framework that supports automatically scaling up/down computing resources of a Hadoop cluster in the cloud is proposed. Specifically, CoveringHDFS is introduced to enable timely remove computing resources without incurring extra data transfer, and a predictive auto-scaling algorithm is developed to more accurately calculate the amount of computing resources to add by considering the time taken to scale up.

### **3.3 Big Climate Data Processing with MapReduce**

#### **3.3.1 Spatiotemporal Decomposition Mechanism**

This section details the mechanism to decompose the array-based data files and store them in HBase.

Normally, climate data are five dimensional: space (latitude, longitude, and altitude), time and variable. For the array-based data models, data are stored in individual files, regarded as a dataset. The dataset is located by the dataset id (e.g., file URI). The array-based data model is expressed as Equation 3.1, and each dataset id refers to a dataset containing five dimensions (X, Y, Z, T and V).

Equation 3.1

$$f(D) = DS(X, Y, Z, T, V)$$

Where  $DS=Dataset$ ,  $V = Variable$ ,  $T= Time$ ,  $X = Longitude$ ,  $Y= Latitude$ ,  $Z = Altitude$ , and  $D=Dataset Id$

In HBase a straightforward way to store the array-based data is using *Dataset Id* as the row key and *Dataset* as row value. While this works for storing data, the parallelization of data processing is problematic because one dataset may reach gigabytes.

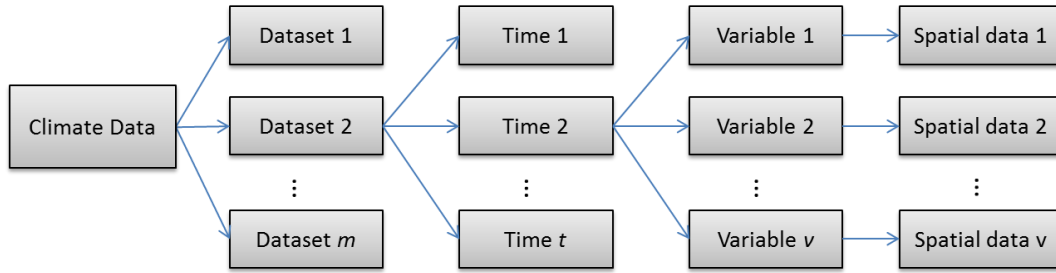


Figure 3.1 Hierarchical structure of the multi-dimensional climate data

Based on the array-based data model, climate data is decomposed hierarchically (Figure 3.1). Each dataset contains one or multiple timestamps, and at each timestamp there are multiple variables; each variable refers to a 2D or 3D data grid. Assuming each data grid as an *AtomDataset*, the decomposed data model is expressed as Equation 3.2.

Equation 3.2

$$f(D,T,V) = ADS(X,Y,Z)$$

Where  $ADS=AtomDataset$ ,  $V = Variable$ ,  $T= Time$ ,  $X = Longitude$ ,  $Y= Latitude$ ,  $Z = Altitude$ , and  $D=Dataset Id$ .

Compared to Equation 3.1, the decomposed data model moves two dimensions T and V from the right to the left side. This triggers two changes: 5D dataset (X, Y, Z, T, V) is degraded to 3D *AtomDataset* (X, Y, Z) and single dataset id (D) becomes composite id(D, T, V). With this decomposition, large volumes of geoscience data are managed in a Bigtable style (Chang et al. 2008), where the (D, T, V) are stored as the composite row key and the *AtomDataset* as the row value in HBase.

Besides the scalability and reliability of HBase, this decomposition has three advantages. First, the D, T, and V are stored in HBase as columns in series enabling flexible search against the time, variable and dataset. Once data are loaded into HBase, the *AtomDataset* queries and accesses data through various filters. Second, new data can be seamlessly appended and integrated to the database without breaking current data structure. And third, parallelization with MapReduce algorithm is achieved in a finer granularity by decomposing the data from 5 to 3 dimensions.

### 3.3.2 MapReduce-enabled Framework for Processing Big Climate Data

Based on the above data decomposition mechanism, we introduce a MapReduce-enabled framework to process big climate data. The back end of the framework is a Hadoop cluster deployed in the cloud environment that provides distributed storage and computing power. The framework contains the following components: *Geo-HBase*, *Controller*, and *Pluggable MR Operator* (Figure 3.2).

- *Geo-HBase* stores the decomposed data. *Geo-HBase* supports flexible queries to the data repository based on dataset id, time, and variable, so a subset of interested data is effectively extracted and processed.
- *Pluggable MR Operator* is a MapReduce program conducting a processing task against the data stored in HBase (e.g., calculating an annual mean for selected variables, sub-setting the data based on user specified regions).
- *Controller* is the user interface allowing users to interact with the framework, such as starting a processing job with specified parameters.

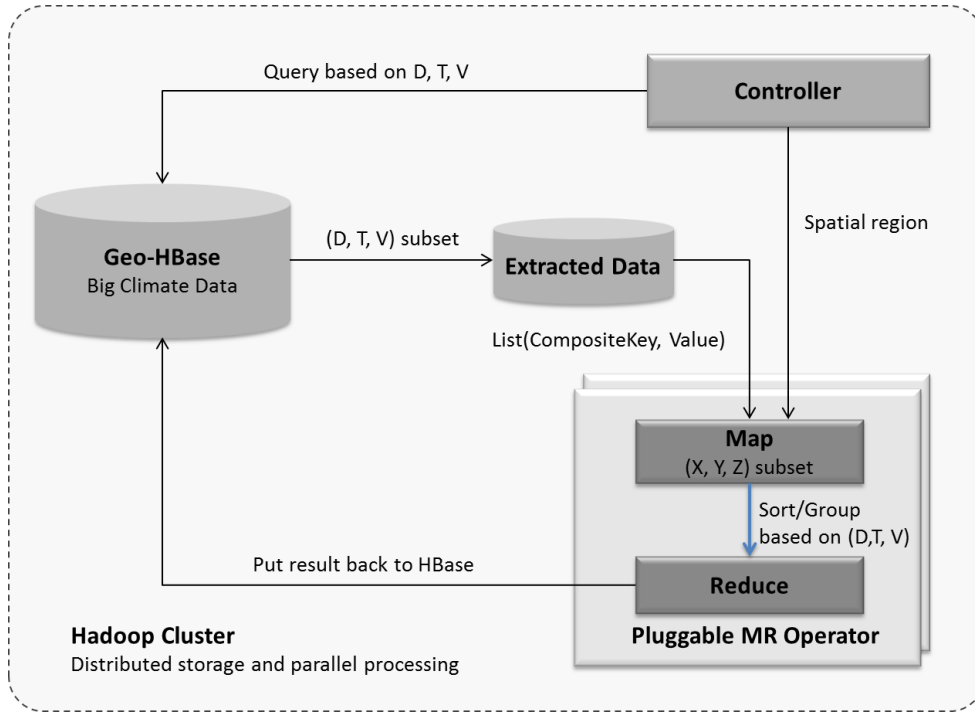


Figure 3.2 MapReduce-based framework for processing big climate data

A typical workflow for processing data with this framework is the following sequence: *Controller* sends processing request with the query parameters (dataset ids, time period, and variables) and spatial region; *Geo-HBase* extracts the required data based on the dataset id (D), time (T), and variable (V); the extracted data are loaded to *MR Operator* as a list of key-value pairs; the *Map* first conducts spatial (X, Y, Z) subsetting based on the specified spatial region. The composite key sorts and groups the emitted intermediate data from *Map* based on the composition of (D, T, V) by *MR Operator*; and finally the result is written back to HBase.

Scientists can develop different MapReduce algorithms to process the data stored in Geo-HBase as *Pluggable MR Operators*. Furthermore, these *Pluggable MR Operators*

are published as *Processing Services* that are used to build the workflow (Chapter 4).

Figure 3.3 is an example MapReduce algorithm for calculating annual global mean of a subset of climate data.

```

Data: monthly global data in Hbase with many datasets
Input: Datasets  $D = \{d_1, d_2, \dots, d_r\}$ 
          Variables  $V = \{v_1, v_2, \dots, v_m\}$ 
          Times  $T = \{t_1, t_2, \dots, t_n\}$ 
          SpatialRegion BBOX = {minLat, minLon, maxLat, maxLon}
Output: annual mean for each selected dataset and variable based on the
           spatiotemporal constraints

/*****Subset the needed data*****/
List<Row> hbaseRows = Query(D, V, T);
/*****Map procedure*****/
/*****The CompositeKey is defined as <D, V, T_year>*****/
Map:
    foreach row in HbaseRows , do:
        NetCDFDoc ncDoc= BuildNetCDFDocFromRow(row);
        Double monthlyMean = ComputeSpatialMean(ncDoc, BBOX);
        CompositeKey comKey = GetCompositeKeyFromRow(row);
        emit(comKey , monthlyMean);
    end

/***** Reduce procedure*****/
/*****CompositeKey is grouped based on <D, V, T_year>*****/
Reduce(CompositeKey comKey, List<Double> values):
    Double sum=0.0;
    foreach value in values do
        sum+=value;
    end
    Double annualMean = sum/12;
    WriteToHbase(comKey, annualMean );

```

Figure 3.3 Algorithm for computing annual global mean of multiple datasets based on the framework

### 3.3.3 Evaluation of big climate data processing

To evaluate the performance of the big climate data processing strategy, I calculated the global monthly mean for 100 model outputs using a 6-node Hadoop cluster (1 master node and 5 slave nodes). Each node is a virtual machine with 8-core CPU/2.35 GHz with 16 GB RAM, and the 100 model outputs are preprocessed and loaded to HBase deployed on the Hadoop cluster. Another virtual machine with the same configuration processes the same data with the traditional serial method. Two sets of tests are conducted. The first keeps the number of cluster nodes the same and processes different numbers of model outputs from 1 to 100 (Figure 3.4A). The second keeps the 100 model output unchanged but changes the number of cluster nodes from 1 to 5 (Figure 3.4B).

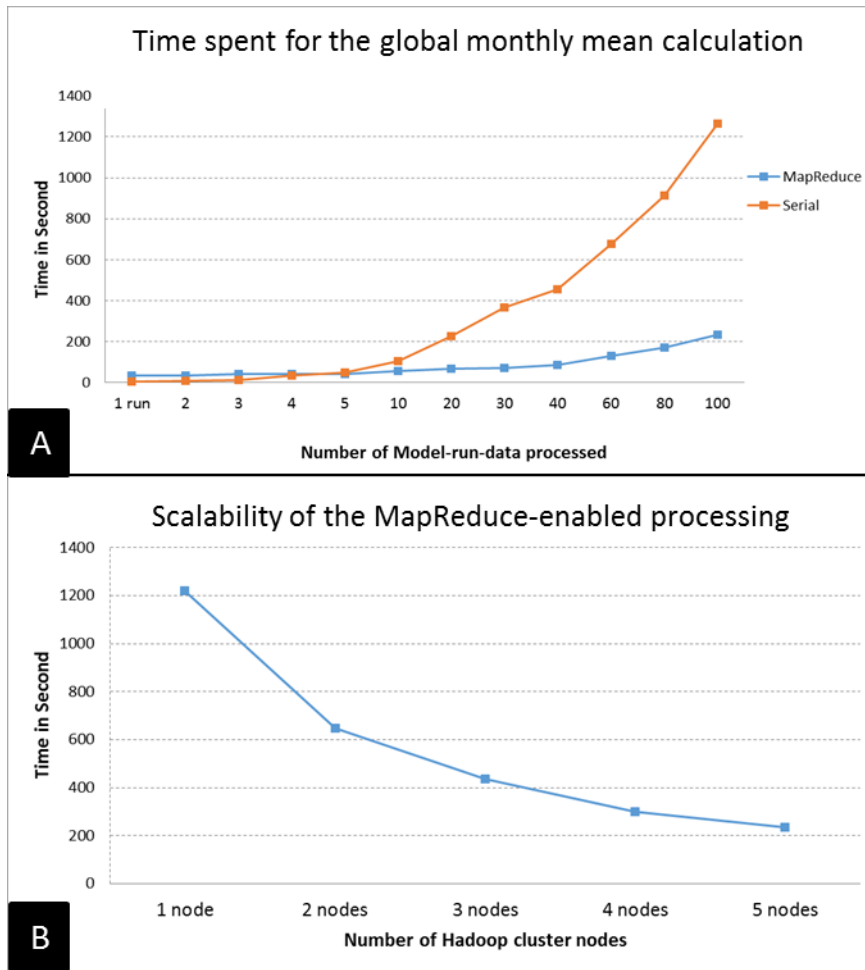


Figure 3.4 Performance evaluation result for the MapReduce-enabled big climate data processing

For the first set of tests and as model output number increases, the time consumed for the serial method increases dramatically for 5 model outputs, whereas the time for MapReduce approach only increases marginally (give a percentage) (Figure 3.4A). With 100 outputs, the serial process takes > 20 minutes, while the MapReduce approach takes ~3.5 minutes. It should be noted that if the number of model output is < 5, the time for the MapReduce approach is more than that of the serial approach due to the overhead of



the Hadoop framework. For the second set of tests and with increasing node number, the consumed time decreases significantly (Figure 3.4B), which indicates efficient scalability of the proposed big geoscience data processing strategy. Scalability is important in cloud environment because new nodes are quickly provisioned and added to the cluster as needed to improve performance.

### **3.4 Auto-scaling Mechanism**

For the aforementioned big climate data processing framework, if the processing workload exceeds the cluster capacity, the performance will be degraded. One traditional solution is to provision “enough” computing resources in advance to handle the peak processing workload. However, the actual workload is rather dynamic than static, and the peak workload is often unpredictable. For example, more computing resources are required when the 300 model-runs are just finished to process the model output. 600 model-runs require even more computing power to ensure data are processed in a reasonable time. Therefore, it is hard if not impossible to estimate how many computing resources will be “enough”. From another aspect, even though we can provide enough computing resources to handle the peak workload; this will be a huge resource waste in that the peak workload may last only in a short time period and most of the time the workload is on a relatively low level. Therefore, how to automatically adjust the computing resources so data processing can be timely finished while minimizing resource usage (cost)?

To address this problem, I propose a novel framework to automatically scale Hadoop cluster in the cloud environment.

### 3.4.1 Auto-Scaling Framework

The goal for the framework is to dynamically adjust computing resources (cluster size) based on the workload to maximize cluster performance (e.g. to support disaster response) while minimizing resource consumption (during the low workload time period). Figure 3.6 illustrates the architecture of the framework, containing the following components: Cloud computing platform, CoveringHDFS-enabled Hadoop cluster, Auto-Scaler and Cluster Monitor.

*Cloud computing platform* provides on-demand computing resources (virtual machines) to the framework. The cloud platform can be both public cloud (e.g. Amazon EC2 and Windows Azure) and private cloud platform (e.g. Eucalyptus cloud and OpenStack). In our experimental prototype, a private Eucalyptus cloud is used.

*CoveringHDFS-enabled Hadoop* cluster is a virtual computing cluster consisting of a number of virtual machines provisioned by the cloud platform. This cluster is responsible for storing and processing geospatial data. CoveringHDFS enables the cluster to be promptly scaled down as needed.

*Auto-Scaler* dynamically adds or removes computing resources based on the current geo-processing workload provided by the *Cluster Monitor*, which continuously collects real-time workload information from the cluster, including running/pending jobs, running map/reduce tasks, and pending map/reduce tasks. If the current workload

exceeds the processing power of the cluster, more computing resources (virtual machines) are added to the cluster. When the workload is relatively low, idle virtual machines are terminated. The core of *Auto-Scalor* is a predictive auto-scaling algorithm that determines when to trigger the scaling operation and estimates how many computing resources need to be scaled.

Such an auto-scaling cluster is able to support various GIScience applications that involve big geospatial data processing. For example, support massive and dynamic concurrent user requests to a terrain-visualization web application by providing right amount of computing resources to interpolate DEM. \

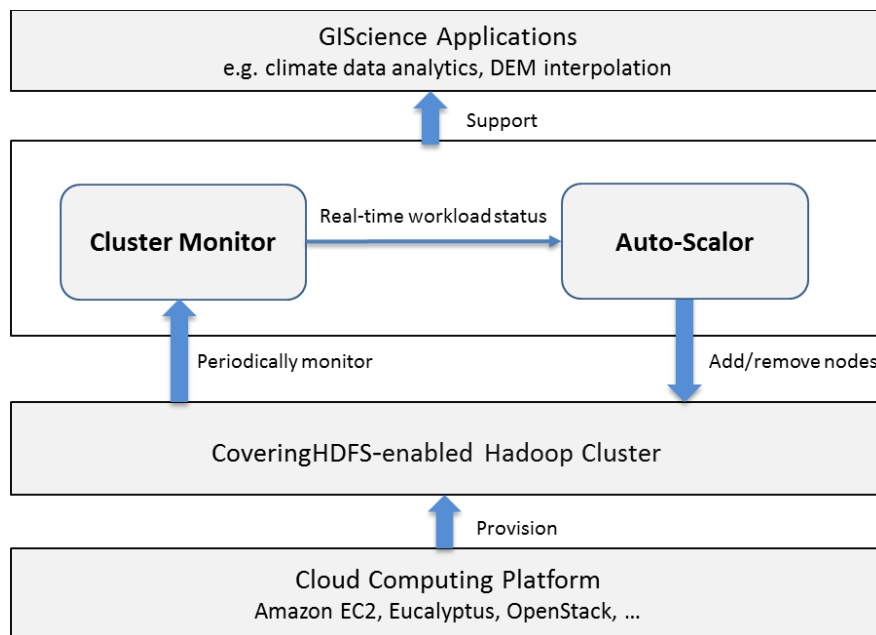


Figure 3.5 Auto-scaling Framework

### 3.4.2 CoveringHDFS

By default, a Hadoop cluster uses Hadoop Distributed File System (HDFS) to store data. Data on HDFS are distributed across all data nodes (nodes that provide both storage capacity and computation) to enable data locality computation. While this storage strategy minimizes data transferring among the cluster nodes, scaling down clusters is difficult since shutting down a node incurs the risk of losing data. Hadoop provides a built-in functionality called ‘decommission’ to safely remove a node from HDFS by first transferring all data blocks from this node to the other nodes before removing it from the cluster. The time taken in decommissioning depends on the data size to be transferred (i.e., minutes, hours, days), which is intolerable for an efficient auto-scaling framework designed to timely capture and respond to a dynamic workload.

To overcome this problem, inspired by the covering subset idea (Leverich and Kozyrakis, 2010), we introduce the *CoveringHDFS* mechanism to scale down the cluster safely and timely without losing data. Different from Leverich and Kozyrakis (2010), instead of modifying the underlying Hadoop software, we store the data directly on a subset of the slave nodes on which the HDFS is deployed. The node consisting of *CoveringHDFS* are called core-slaves, and other nodes are compute-slaves. Core-slaves provide both storage and computing power, whereas compute-slaves only provide computing power. Thus, a cluster is comprised of master node, core-slaves and compute-slaves (Figure 3.6). By incorporating the *CoveringHDFS* in a cluster, idle compute-slaves can be removed instantly when the workload is low (a scaling down operation is triggered). A CoveringHDFS-enabled cluster can have three modes: *FullMode*,

containing core-slaves, compute-slaves and master; *CoreMode*, containing core-slaves and master; and *TerminateMode*, the whole cluster is terminated.

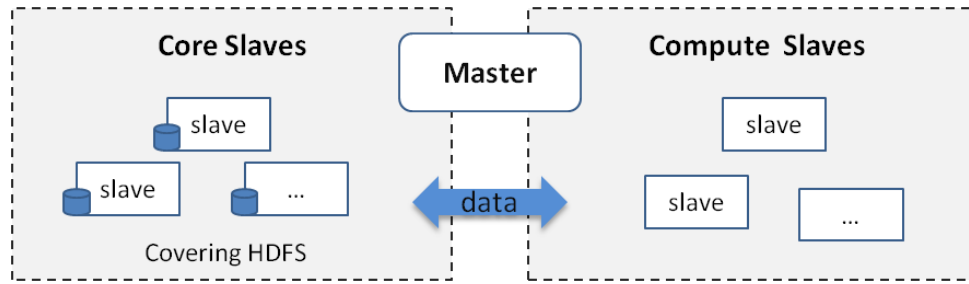


Figure 3.6 Structure of the Hadoop cluster with CoveringHDFS

### 3.4.3 Auto-Scaling algorithm

The Auto-Scaling algorithm adds compute-slaves when the workload exceeds cluster's computational power (e.g., jobs are waiting to be executed) and releases idle compute-slaves when workload is low.

- **Scaling up**

A MapReduce job normally contains many map tasks and one or several reduce tasks. Each slave node in the Hadoop cluster has a maximum capacity of processing map/reduce tasks in parallel which is normally determined by the slave's number of CPU cores and memory size. Suppose each slave at the same time can process  $n$  map tasks and  $m$  reduce tasks, we can think that each slave has  $n$  map slots and  $m$  reduce slots. If the amount of map tasks (workload) exceeds that of map slots (cluster capacity), the

exceeded map tasks need to wait until free map slots are available so the performance is impaired. The key for scaling up mechanism is to estimate how many extra computing resources (compute slaves) are required when the workload exceeds the cluster's capacity.

Suppose that there are 10 total pending map tasks at a moment  $T_o$ , and that each new added slave increases 2 map slots. One simple solution is to add 5 slaves so each pending task is handled by a map slot. However, adding 5 slaves cannot be done instantly. For example, provisioning a medium Amazon EC2 instance with Linux OS takes approximates 10 minutes. In this case, when the 5 slaves are added, there may be no pending maps, so newly added slaves have no functionality if no other jobs are submitted. Therefore, the time spent on the scaling-up process ( $T_N$ , time needed to add N slaves) must be considered to estimate the right number of slaves to add. To this end, we introduce a predictive scaling up algorithm to estimates the workload at the time  $T_o+T_N$  when the scaling up process is finished. The number of slaves to be added (denoted as  $N$ ) is calculated using Equation 3.3

Equation 3.3

$$N = \left\lceil \frac{N_{pending} - N_{finished}}{n} \right\rceil$$

where,  $N_{pending}$  is the number of pending tasks at  $T_o$ ,  $N_{finished}$  is the number of pending map tasks that will be finished or become running when the scaling-up process is done,  $n$  is the number of map slots for each added slave.

Suppose scaling up  $N$  nodes to the cluster takes  $T_N$  minutes. The  $N_{finished}$  is estimated as follows: for a list of processing slots ( $N_{slot}$ ) and task queue with  $N_{map}$  tasks, each slot has the same processing power, but each task may require different time (e.g., job types). At the moment  $T_o$ , each slot processes a task with the progress  $p$ . The waiting tasks are processed following the principle of first-come-first-serve. Once a slot has finished its current task, a new task is fetched from the task waiting list. This process is repeated until the time spent on each slot is  $T_N$ .

Based on this model, the number of pending tasks that are finished or become running during the time period  $T_N$  is equal to the number of map tasks that are finished on each slot in  $T_N$ , which can be calculated as follows:

Equation 3.4

$$N_{finished} = \sum_{i=1}^{N_{slot}} \sum_{j=1}^{T_{remain}^j > 0} \min\left(\frac{T_{remain}^{j-1}}{T_{task}}, 1\right)$$

$$\text{when} \begin{cases} j = 1, T_{remain}^0 = T_N \text{ and } T_{task} = (1 - p_i) * T_{i1} \\ j > 1, T_{remain}^j = T_{remain}^{j-1} - T_{task} \text{ and } T_{task} = T_{ij} \end{cases}$$

where,  $N_{slot}$  is the number of map slots for the current cluster,  $T_{remain}^j$  is the time left in time period  $T_N$  when a node has finished  $j$  tasks (counted from time moment  $T_o$ ),  $T_{ij}$  is the time required for finishing the  $j$ th map task on the  $i$ th slot ( $T_{ij}$  differs for different job types),  $p_i$  is the progress of the running map task on the  $i$ th slot at  $T_o$  ( $0 \leq p_i \leq 1$ ),  $T_N$  is the time spent on scaling up  $N$  nodes, which depends on the cloud platform and the number of node to be scaled.

Figure 3.7 is an example of a specific time point (i.e.,  $j$ th loop) of Equation 3.4. At the  $j$ th loop, the first three slots still have capability remained to accept new map tasks during the time period  $T_N$ , while the remaining slots cannot accept new tasks. After each loop, slots are sorted based on the  $T_{remain}$  in the descending order.

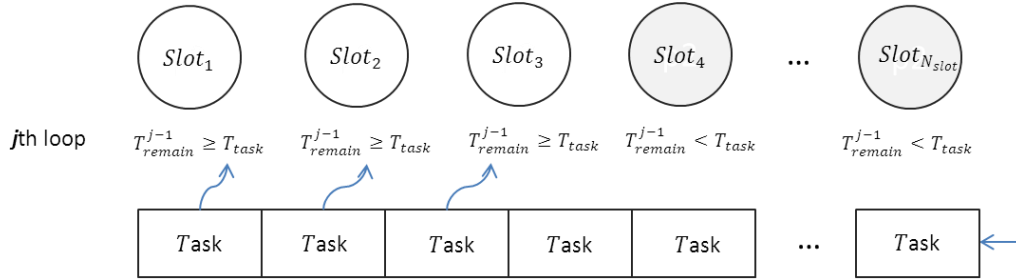


Figure 3.7 An example status of the  $j$ th loop of Equation 3.4

In Equation 3.3 and 3.4,  $N_{slot}$ ,  $N_{pending}$  and  $p_i$  are collected from the Hadoop cluster by *Cluster Monitor* in real time;  $T_{remain}^j$  is calculated dynamically;  $T_N$  is



represented as a function of  $N$ :  $T_N = f(N)$ . The definition of  $f(N)$  depends on the cloud platform on which the cluster is deployed. For example, for our Eucalyptus cloud platform, the function is  $f(N) = 0.2 * N + 1.67$  (unit: minute). Hence, Equation 3.4 is represented as a function of  $N$ , as  $N_{finished} = g(T_N) = g(f(N))$ . Finally, the number of slaves to be scaled is derived as:

Equation 3.5

$$N = \left\lceil \frac{N_{pending} - g(f(N))}{n} \right\rceil$$

If  $N > N_{max} - N_{current}$ , then  $N = N_{max} - N_{current}$

where,  $N_{max}$  is the maximum number of slaves allowed in the cluster, and  $N_{current}$  is the number of slaves for the current cluster.

- **Scaling down**

Removing the compute-slaves is straightforward. When the idle time (no running map tasks or reduce tasks) for a compute-slave exceeds a user specified threshold  $T_{nodeIdle}$  (e.g., 5 minutes), this slave is terminated. In this way, when the workload is low enough to be handled by only core slaves, the cluster will degrade from the *FullMode* to the *CoreMode*. If the cluster under the *CoreMode* is idle (no running or pending jobs) for a specified time period  $T_{clusterIdle}$ , the cluster can either be terminated

(entering the *TermiateState*) or remain idle based on user's configuration. With *CoveringHDFS*, compute-slaves do not store data, removing the slave is completed in a few seconds, which avoids extra resource consumption during the scaling down process.

#### 3.4.4 Evaluation of auto-scaling mechanism

- Prototype

The prototype uses the same private cloud described in Chapter 2 (2.4.2). A Hadoop (version 1.0.4) virtual machine image (built based on CentOS 6.0) is used to provision Hadoop clusters. In this prototype, time spent scaling up  $N$  nodes is represented as  $T_N = 0.2 * N + 1.67$  (*minute*).

- Experiment design

#### **Workload**

To simulate the workload patterns, a different number of data processing jobs were submitted to the cluster every 6 minutes in a 10 hour period (Figure 3.8). Each submission can be considered as a workload for the cluster, and totaled 100 workloads consisting of 224 jobs were submitted within the 10 hour period. The rational for using this workload pattern design is that it represents typical workload patterns of increasing (0 to 100 minute), decreasing (130 to 180 minute), on-off and periodical busting (200 to 420 minute), and light or idle workload (420 to 600 minute).

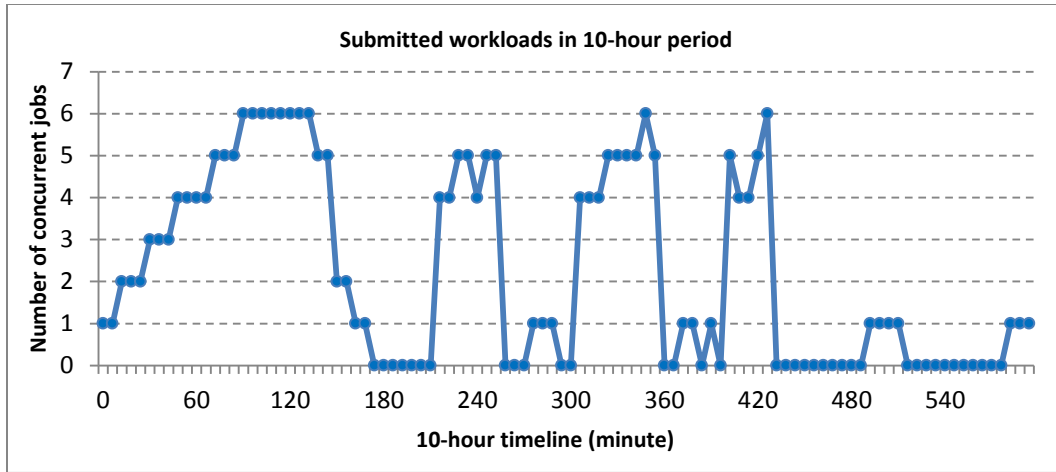


Figure 3.8 100 workloads submitted to the cluster in a 10-hour period with a time spacing of 6 minutes. A total of 224 jobs were submitted (each dot representing each workload)

## Hadoop clusters setup

Three Hadoop clusters in our private cloud environment were used for the sake of comparison: 1) auto-scaling cluster based on the proposed framework, 2) static cluster with 7 slave nodes, and 3) another static cluster with 14 slave nodes. The detailed configuration for the three clusters is shown in Table 3.1. All three clusters used one medium instance as the master node. The auto-scaling cluster started with 3 core slaves, serving as the covering HDFS and capable of scaling up to 15 compute slaves during the peak workload. The two static clusters were used as baselines to compare with the auto-scaling cluster. In comparison to the workloads discussed previously, the static cluster with 7 slaves served as a cluster with less capability, while the static cluster with 14 slaves served as a full-powered cluster.

Table 3.1 Three Hadoop clusters

Cluster type	Master	Slaves	HDFS
Auto-scaling cluster	1 medium instance	Dynamic, start with 3 core slaves with medium instances, can scale up 12 compute slaves with small instances <sup>2</sup>	Covering HDFS, starting with 3 core slaves
7-slave cluster	1 medium instance	Static, 7 slaves with 3 medium instances and 4 small instances	Traditional HDFS with 7 slaves
14-slave cluster	1 medium instance	Static, 14 slaves with 3 medium instances and 11 small instances	Traditional HDFS with 14 slaves

Notes: Medium instance: 2-core CPU, 2 Gb RAM, 10 Gb storage; and small instance: 1-core CPU, 1Gb RAM, 8Gb storage. For all clusters, each slave node configured with 2 map slots and 1 reduce slot.

- Result and discussion

For the three clusters, the time to finish each job for the 100 submitted workloads are recorded (Figure 3.9). By aggregating the times spent in each workload, the times consumed by each workload were derived.

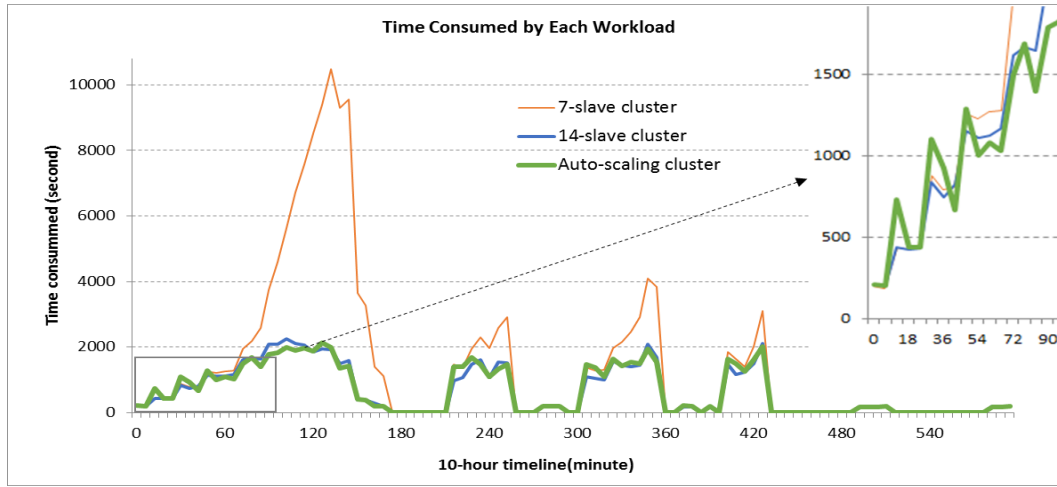


Figure 3.9 Time consumed by each workload (100 workloads in total) for the three clusters in the 10-hour period. The small diagram on the top-right corner is the zoomed-in view for the first 1.5 hour

## Performance

Generally, the workload finish time for each cluster has a similar pattern to that of the workloads pattern (Figure 3.9). However, the 7-slave cluster responded more sensitively to the burst workloads, especially when the workloads were maintained at a high level over a relatively long time as illustrated at the 90 to 150 minute. Thus, when the workloads exceeded the capacity of the cluster, the job requests were queued and could only be processed when previous jobs were finished. The queue became longer with the increase of workload, which in turn lengthened the time to finish. Therefore the cluster with a fixed number of slaves had very limited capacity to handle the dynamic workload unless being provisioned for the peak workload.

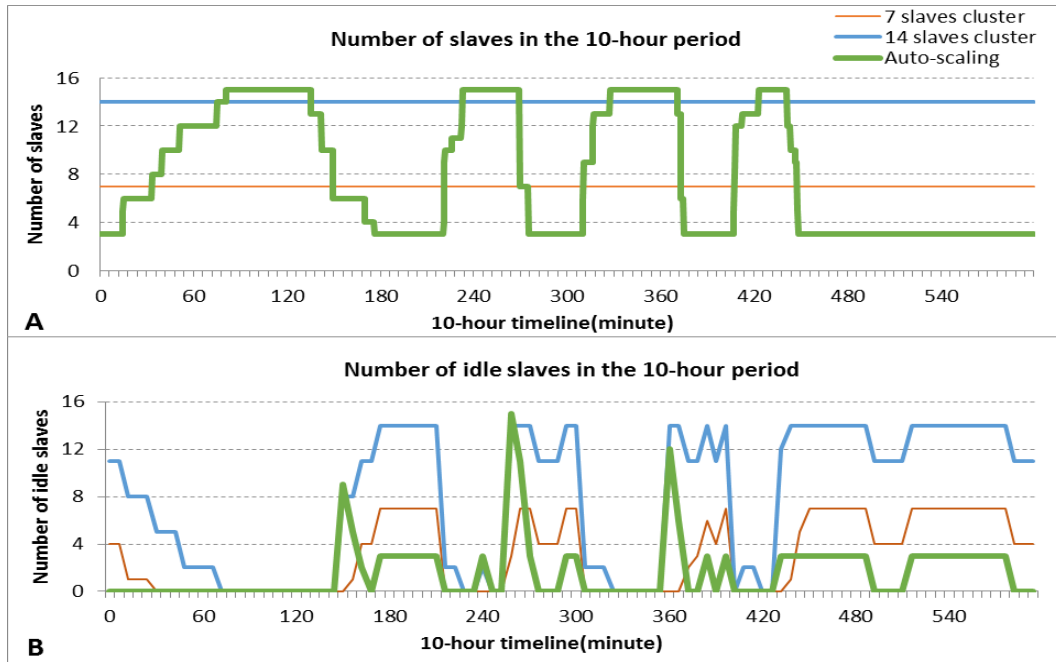


Figure 3.10 (A) The size change of the auto-scaling cluster in the 10-hour period (the number of slaves monitored every 30 seconds). The 7-slave and 14-slave clusters are represented as two parallel lines; (B) Number of idle slaves in the 10-hour period

For the auto-scaling cluster, the time consumed by each workload showed a similar pattern to that of the 14-slave full powered cluster. As illustrated in the zoomed-in view of Figure 3.9, the auto-scaling cluster had sharper fluctuations than the 14-slave cluster for the first 1.5 hours. These sharp jumps occurred during the scaling-up when the cluster was running with under-powered mode. Once scaling up was finished, the cluster was in full-power mode for current workload which explains the sharp drop of the workload finish time. This process is further illustrated in Figure 3.10A, where the changing size of the auto-scaling cluster driven by the dynamic workloads in the 10-hour period is evident. The auto-scaling cluster effectively handled the increasing and burst

workload patterns by increasing the computing power (compute slaves) to ensure the jobs were finished within an acceptable time. Overall, the auto-scaling cluster shows similar performance as the full powered cluster.

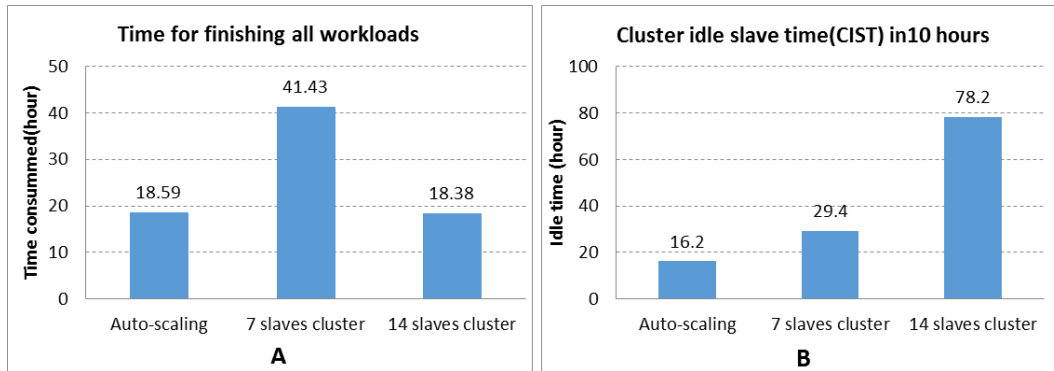


Figure 3.11 (A) Time consumed for finishing all workloads for each cluster. Time calculated by summing the 224 job finish times; (B) Cluster Idle Slave Time (CIST) for each cluster in the 10-hour period

## Resource consumption

To evaluate resource consumption and utilization rate of the auto-scaling cluster, the number slaves (Figure 3.10A) and idle slaves (Figure 3.10B) of the three clusters were recorded during the 10-hour period to monitor the slave status. Figure 3.10B shows that the general pattern of idle slaves following the pattern of workloads but with an opposite direction. When the workload peaked, all three clusters have no idle slaves. When the workload ebbed, the 14-slave cluster had the largest number of idle slaves, while the auto-scaling cluster had the fewest idle slaves. And the 7-slaves cluster is between these two extremes.

To better evaluate the performance and resource utilization of the auto-scaling cluster in a quantitative manner, the finish times of the 224 jobs for each cluster were summed (Figure 3.11A), from which Cluster Idle Slave Time (CIST) was calculated (Figure 3.11B). CIST is defined as the sum of idle time for each slave in a given time period for a cluster. The auto-scaling cluster spent 18.59 hours, only 12 minutes longer than the 18.38 hours of the 14-slave cluster (Figure 3.11A). Compared to the 7-slave cluster, the auto-scaling cluster spent 23 fewer hours (2.2X increase in performance) in completing all jobs. For the resource utilization the auto-scaling cluster had the lowest idle time (16.2 hours), 80% less than that of the idle time for the 14-slave cluster (78.2 hours), saving 62 instance hours in the 10-hour period (Figure 3.11B). If the cluster were to run for one month with repeated workload patterns, the auto-scaling cluster would save 4,464 instance hours. This equals to the saving of ~\$232/month if this cluster is deployed on Amazon EC2 using the similar instance (EC2's t2.medium instance type pricing at \$0.052/hour<sup>16</sup> not including the storage, network and other costs).

Generally, the 14-slave cluster maintained an excellent performance when workload was high but wasted more computing resources (more idle slaves, Figure 3.11B) when the workload was low. The 7-slave cluster showed better resource utilization as compared to the 14-slave cluster but sacrificed performance (Figure 3.9) when workload was high. By dynamically adjusting the cluster size based on the workload, the auto-scaling cluster showed overall the best resource utilization (the lowest idle time) while delivering noticeably better performance (nearly equaled to that of the 14-slave cluster).

---

<sup>16</sup> <http://calculator.s3.amazonaws.com/index.html>



The proposed auto-scaling framework is a smart approach to allocate the right amount of computing resources to the right workload in an automatic manner. Such a cloud-enabled, auto-scalable Hadoop cluster mechanism is a powerful tool to process big geoscience data in the cloud environment with optimized performance and reduced resource consumption. This could be used in planning better performed geospatial cyberinfrastructure to address computationally intensity challenges (Yang et al, 2010; Wang, 2010).

### **3.5 Chapter Summary**

By incorporating the big geoscience data processing strategy, the proposed framework manages and processes big geoscience data. The data decomposition and storage mechanism enables the multi-dimensional geoscience data to be effectively stored in a distributed environment (HBase), while the MapReduce-enabled processing framework enables data to be processed in parallel. An auto-scaling framework is proposed to automatically scale Hadoop clusters in a cloud environment based on the workload for supporting big geospatial data processing

## **CHAPTER 4 SERVICE-ORIENTED CLOUD-BASED SCIENTIFIC WORKFLOW**

### **4.1 Introduction**

To tackle the procedure complexity challenge, I propose a cloud-based workflow framework by incorporating cloud computing to provision on-demand workflow execution environment. In this framework, methodologies are proposed by leveraging cloud computing, parallel computing and Service Oriented Architecture (SOA). Specifically 1) a unified service model is proposed to define four types of basic services (process service, data service, model service, and infrastructure service), and enables these services to be chained together in a unified manner; 2) by incorporating cloud computing, the workflow framework enables on-demand provision computing resources during the workflow execution.

This workflow framework not only tackles the challenge of procedure complexity, but also enables MaaS (Chapter 2) and scalable climate data analytics (Chapter 3) to be seamlessly integrated. Such integration forms a geospatial cyberinfrastructure which reduces the experiment time of the study case from 2 months to 5 days with a few simple clicks.

### **4.2 Literature Review**

Scientific workflow serves as a problem-solving environment simplifying tasks by creating meaningful sub-tasks and combining to form executable data analysis pipelines (Lud äscher et al. 2006). Scientific workflow provides mechanisms to discover, share, analyze, and evaluate research tools (Wang 2010, Yang et al. 2010) and is a significant element of geospatial cyberinfrastructure (Lud äscher et al. 2006, Deelman and Zemankova 2006, Gil 2008, Taylor et al. 2007). Provenance tracking provided by workflow systems enables geoscientists to determine the reliability of the data and service products and validate and reproduce scientific results in cyberinfrastructure (Yue and He 2009).

There are several scientific workflow systems including Kepler (Lud äscher et al. 2006), Taverna (Oinn et al. 2004), Triana (Majithia et al. 2004), Trident (Barga et al. 2008) and VisTrails (Bavoil et al. 2005). These systems compose and schedule complex workflows on a distributed environment, such as clusters and Grids (Foster et al. 2001). As a new computing infrastructure, cloud computing is a new approach for deploying and executing scientific workflows (Juve and Deelman 2011). Preliminary studies to evaluate feasibility and performance of migrating scientific workflows into the cloud (Hoffa et al. 2008, Juve et al. 2009, Simmahan et al. 2009) have found that cloud computing provides comparable performance with better scalability and flexibility to traditional computing infrastructure given similar resources. However, these studies mainly focused on deploying current scientific workflow platforms to the cloud environment by replacing traditional physical machines with virtual machines in existing workflow deployment. A

more comprehensive study is desired to fully leverage the advantages of cloud computing to enable scientific workflow for supporting geoscience.

## **4.3 Methodologies**

### **4.3.1 Framework**

The framework (Figure 4.1) is layer-based and includes four layers: computing resource (Cloud Platform); processing (Hadoop Cluster); service; and presentation (Workflow Builder). Cloud platform provides the on-demand computing resources including computing, storage, and network as services. The cloud platform includes a processing layer where the workflow engine running on the virtualized Hadoop cluster (virtual machines as cluster nodes). By integrating cloud computing, computing resources associated with the workflow are provisioned or terminated on-demand to ensure performance while minimizing resource consumption.

The service layer is built on top of the cluster for registering, managing, and chaining the services. The services are chained as executable workflows in an on-demand and scalable computing environment. Processing layer and service layer form the workflow execution environment. On top is the presentation layer which enables users to publish, discover and use services to build workflows in a drag-and-drop style, and runs and monitors workflows in a web-based interface. Oozie<sup>17</sup> is adapted as the workflow engine due to its intrinsic integration with Hadoop MapReduce.

---

<sup>17</sup> Oozie <http://yahoo.github.com/oozie/>

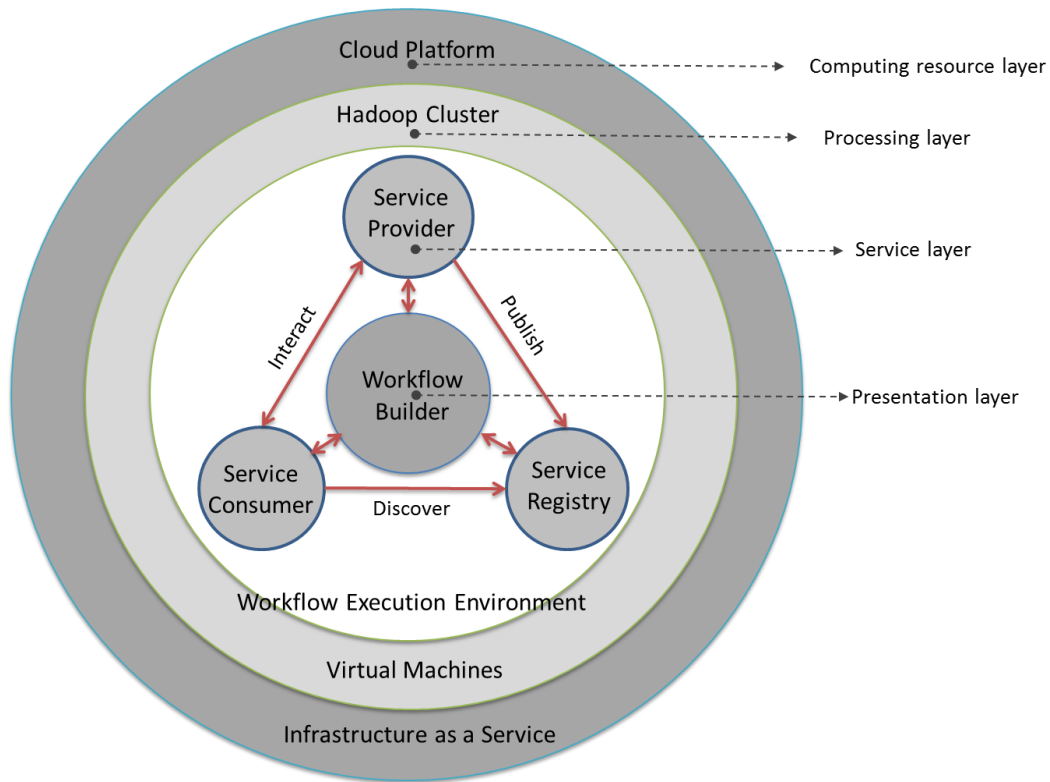


Figure 4.1 Framework architecture

Service Oriented Architecture (SOA) is adopted to publish different processes as individual services, and these not only include processing and data services but also offer infrastructure and tool services. Different from the traditional web services orchestration, the “service” herein does not refer to the “web service” but rather to a self-described functional unit plugged into the workflow. This section details the framework from big geoscience data process, service-oriented approach, and cloud-based workflow execution environment.

#### 4.3.2 Unified service model

The key to leveraging service-oriented concept in the workflow architecture is that each step in the workflow is abstracted as a service, and various services are chained to form a workflow. To ensure that different services can be connected in a unified fashion, we abstract each service as a processing unit with two general interfaces: input and output. For input, two types are defined: Input Parameter (IP) and Input Data (ID). Similarly there are two output types: Output Parameter (OP) and Output Data (OD). The input and output parameters are primitive, (e.g., numbers, short texts), whereas the input and output data refer to data files stored in the shared file system. Based on this, we define a unified service model as Expression 1, where the output data and parameter of one service are used as the input data and parameter by another service, thus enabling the servicing chaining.

$$Service(ID, IP) \xrightarrow{execute} (OD, OP) \quad \text{Expression 4.1}$$

Unlike traditional scientific workflow in which each step is normally the computational process, we define four types of services to build a workflow, and each is described below.

- **Processing Service** processes, analyzes or visualizes input data. Three types of programs are published as a process service: MapReduce program processing the big geoscience data stored in HBase; Java executable program conducting general processing task; and Shell script conducting data preprocess, statistics or visualization. For example, a Shell script calling R script to plot a climate variable is published as a process service.

- **Data Service** focuses on fetching data from outside of the workflow as service input and publishing output as various services to share.
- **Model Service** is enabled by Model as a Service, which runs geoscience models (e.g., climate model) with user specified model input; the modeling environment of software configuration and computing resources running the model are automatically provisioned in the cloud.
- **Infrastructure Service** provisions the virtual machine-based services by leveraging the IaaS. Three types are included: provisioning pure computing resources (e.g., bare-metal virtual machine); provisioning computing platforms (e.g. Hadoop or MPI-based cluster); and provisioning virtual machines with pre-installed software packages or applications (e.g., virtual machine with R environment).

Following the service model, each service is composed of service executable program and service definition metadata. Service definition metadata is an XML describing the services (Figure 4.2) and is comprised of three sections: service description of the general service information; service entry point indicating the location of the service executable program; and service interface detailing the service input and output along with semantic description. To register a service into the workflow framework, the service definition metadata is first interpreted to add the service in the service catalogue, and the service executable program is uploaded to the workflow execution environment.

```

<?xml version="1.0"?>
<!--A standard xml for describing the service metadata for the workflow platform -->
- <service name="ProvisionVM" xmlns="uri:cisc:service:0.1">
  <!--Service description-->
  <description>Provision a virtual machine based on user's configuration </description>
  <category>Infrastructure Service</category>
  - <provider name="CISC">
    <site>http://cisc.gmu.com</site>
    - <contact>
      <individual-name>Contact name</individual-name>
      <email>test@gmail.com</email>
      <phone>703-000-0000</phone>
    </contact>
  </provider>
  - <entry-point type="JAVA">
    <!--Executable program in a web accessible location, if not specified, this file must be uploaded manually during registration. -->
    <executable-location>runnable jar file in a web accessible location</executable-location>
    <!--Only needed when the entry-point type is "JAVA": specify the main class here.-->
    <main>gmucisc.service.ProvisionVM</main>
  </entry-point>
  <!--Service interface: input/output-->
  - <interface>
    - <input name="VM Image ID">
      <description>Image Id for the virtual machine</description>
      <default-value>emi-3EED351F</default-value>
    </input>
    - <input name="VM type">
      <description>Virtual machine types, following the standard of Amazon EC2, value could be m1.small, c1.medium, c1.xlarge</description>
      <default-value>m1.small</default-value>
    </input>
    - <output name="VM instance id">
      <description>Instance id for the virtual machine, can be used to terminate the virtual machine </description>
      <variable-id>VM_ID</variable-id>
    </output>
    - <output name="VM public ip">
      <description>Public IP address for this VM, can be used to access the virtual machine</description>
      <variable-id>VM_IP</variable-id>
    </output>
  </interface>
</service>

```

Figure 4.2 Service definition metadata example (Infrastructure Service: ProvisionVM)

### 4.3.3 Loosely-coupled Service I/O Mechanism

The workflow engine is deployed on Hadoop, and the workflow tasks (services) are executed on different machines. Hence, it is important that all services read input and write output data in a shared file system to avoid extra data transfer loads. The HDFS is used as such a file system in the framework, providing a unified service execution environment. However, geoscience analytics often requires small to mid-sized data from remote data services (e.g., WFS, WCS, and OPeNDAP) as part of the input, and publish the output as web services (e.g., WMS, WFS). One solution is that the service includes the function to fetch and publish data from remote services. However, at least two



problems arise. The first is that data handling is tightly coupled with the processing logic, which makes it difficult for the service to incorporate other types of data services. The second is that each service implements its own data handling function which cannot be reused. We propose a loosely-coupled, service Input/output (I/O) mechanism as illustrated in Figure 4.3 to address these shortcomings.

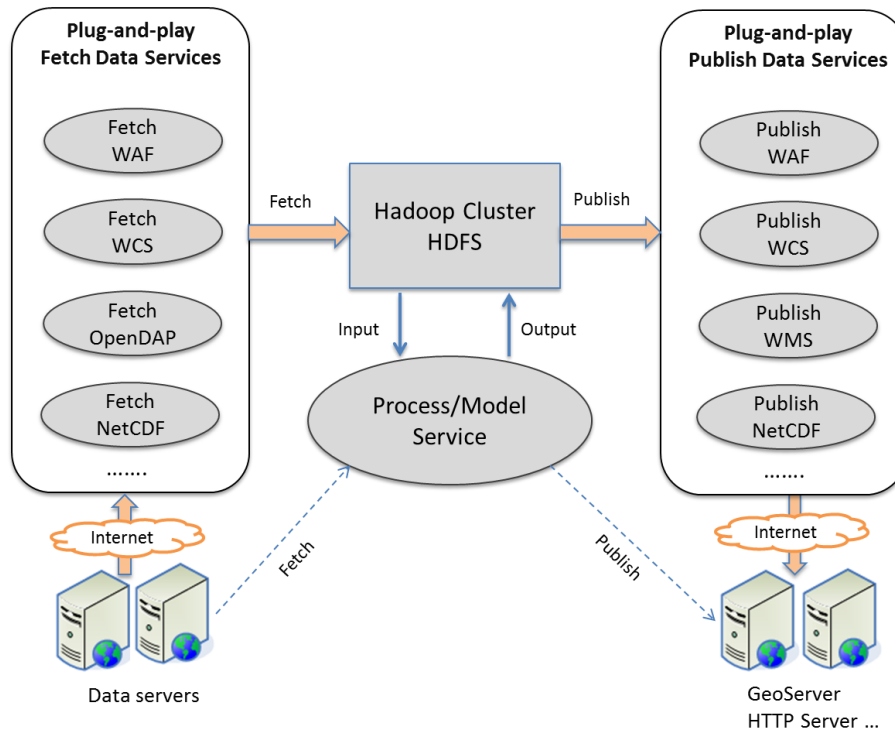


Figure 4.3 Loosely-coupled Service I/O mechanism

This mechanism extracts the data handling components and publishes them as individual workflow *Data Services*, including two categories: *Fetch Data Services* and *Publish Data Services*. *Fetch Data Service* fetches data from remote data servers and

loads them into HDFS. Other services, such as *Processing Service* and *Model Service*, can access the data directly from HDFS. For example, *Fetch WAF* (Web Access Folder) service downloads data from a WAF and loads them to HDFS; *Fetch OpenDAP* service subsets data from an OpenDAP server. *Publish Data Service* requires a server to host the data. For example, to publish a *Processing Service*'s output as WMS, a WMS server (e.g., GeoServer) is required to host the service, and, an *Infrastructure Service* can be integrated into the workflow to provision a virtual machine with pre-installed GeoServer.

Figure 4.4 shows a typical workflow consisting of four different services:

- ① A *Fetch Data Service* fetches vector data (U.S. state boundary) from a WFS server as the input of the *Processing Service*;
- ② The *Processing Service* is a MapReduce program which calculates the monthly mean land surface temperature from the climate data stored in *Geo-HBase* using the boundary data as the statistics unit;
- ③ Meanwhile, an *Infrastructure Service* provisions a virtual machine with pre-installed GeoServer from the cloud platform; and
- ④ *Publish Data Service* publishes output data from process service to GeoServer as WMS.

This service I/O mechanism is flexible and extendable in that external services are supported by developing corresponding data services in the workflow platform. Once a data service is registered, it can be used by any other services to fetch/publish

input/output. This service I/O mechanism addresses the challenge of heterogeneous and distributed data associated with each step's input and output in the workflow.

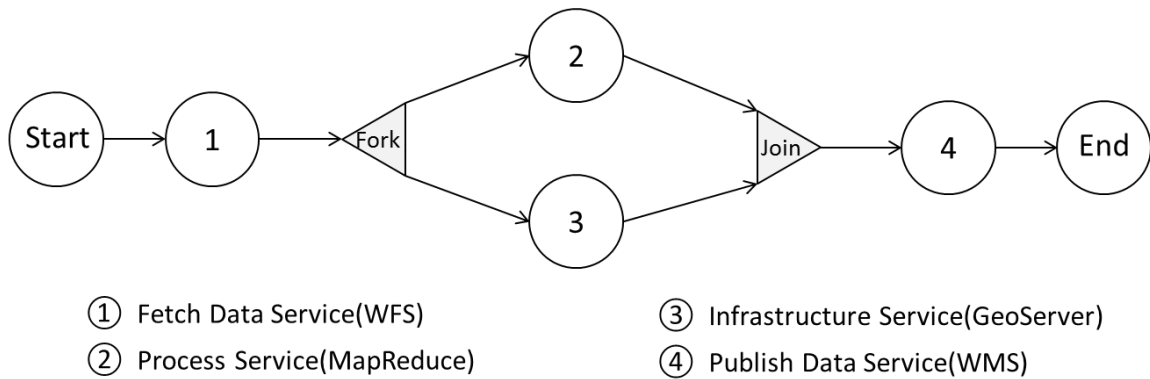


Figure 4.4 A typical workflow with four types of services

#### 4.3.4 Cloud-based Workflow Execution Environment

Scientific workflows normally require a collection of software components such as tools, libraries, programs, models, and applications, and these components are developed at different times by different people. In this case study the workflow needs to setup and run a climate model, first with NetCDF Operator (NCO) library to preprocess the model output, followed by Hadoop MapReduce to parallel process model output, and then fed to a Java program (or R script) to conduct linear regression analysis and visualization. These heterogeneous software components must be seamlessly integrated into a coherent workflow. To achieve this, a traditional workflow system needs to pre-install the required software components on the physical machine (s), and this poses two

problems. First, if the execution environment is backed by a cluster, the same software components must be configured on each machine, and any update to the execution environment is time consuming. Second, some software components are complex requiring specific execution environments that cannot be installed on the common environment. To address these shortcomings, we propose the workflow in the cloud environment with two mechanisms.

The first mechanism deploys the whole Workflow Execution Environment (WEE, Hadoop cluster) in the cloud. The entire WEE is “burned” to image, including Hadoop software, workflow engine, and library environment for executing the workflow tasks (e.g., R, NCO, JRE) and can be provisioned within minutes. The VMs are provisioned as cluster nodes based on the VM image (a snapshot of pre-configured operating system used to launch a VM). When an update is required, the VM image is re-built by installing new or removing old software components, and the WEE is re-provisioned quickly based on the new VM image. Another advantage is that new computing resources can be easily added to the WEE by provisioning more cluster nodes.

The second mechanism integrates specified software into VM images and publishes these images as *Infrastructure Services*. This is more flexible in that the software environment is self-contained and exposed as a standard infrastructure service in the workflow platform. These services are added and removed without affecting current WEE. In addition, the complex software components (e.g., climate model, GeoServer) are difficult to integrate into WEE due to the specified system requirement and high

resource occupation and publishing them as *Infrastructure Services* improves the system performance and flexibility. Furthermore, this mechanism provides an alternative to integrating legacy software that requires a specific execution environment into the workflow. Finally, the image-based *Infrastructure Service* offers a reproducible environment for certain tasks in the workflow.

#### **4.4 Implementation Architecture**

This section details the implementation architecture of the workflow framework. Evaluation of this framework is detailed in Chapter 5 where MaaS and big data processing are integrated as workflow services.

The prototype uses the same private cloud described in Chapter 2 (2.4.2). The prototype implementation architecture (Figure 4.5) contains four major components: *Eucalyptus Cloud*, *Workflow Execution Environment (WEE)*, *Web-based Workflow Builder*, and *Service/Workflow Registry*.

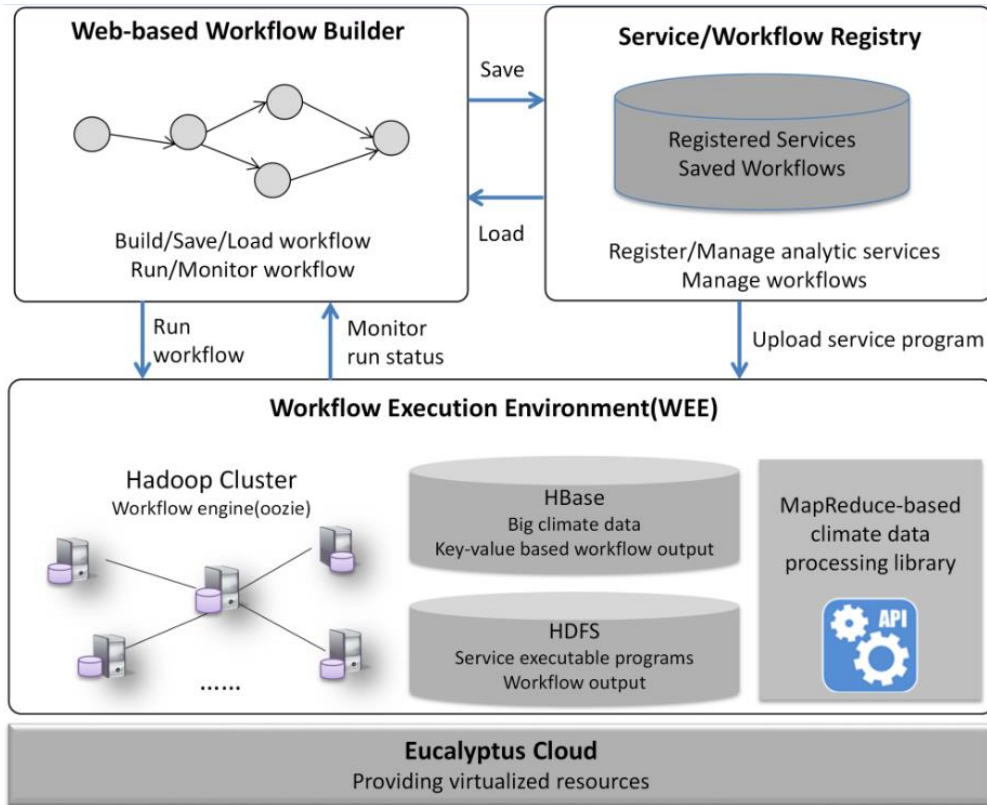


Figure 4.5 Prototype implementation architecture

*Eucalyptus Cloud* provides virtualized computing resources. The WEE, built on top of the cloud platform, consists of computing, storage and processing libraries. The computing is provided by a virtualized Hadoop cluster and coordinated by the workflow engine (powered by Oozie). Storage is provided by HBase and HDFS, where HBase stores the decomposed big climate data and key/value-based workflow output, whereas HDFS stores the service executable programs and other workflow output.

*Service/Workflow Registry* is the service layer providing a database for managing the registered services and saved workflows. Service definition metadata (XML) and

workflow definition files (XML) are stored in the database. During service registration, the service executable program is uploaded to WEE.

*Web-based Workflow Builder* is the graphic interface (Figure 4.6) through which users build workflow by visually connecting various services, run workflow by submitting the request to WEE with one-click, and monitor the workflow execution status in real time. Services and workflows are loaded to the builder from the registry. The workflow is saved to the server for re-running or downloaded as XML for sharing. The builder is based on the open source workflow-generator tool<sup>18</sup>.

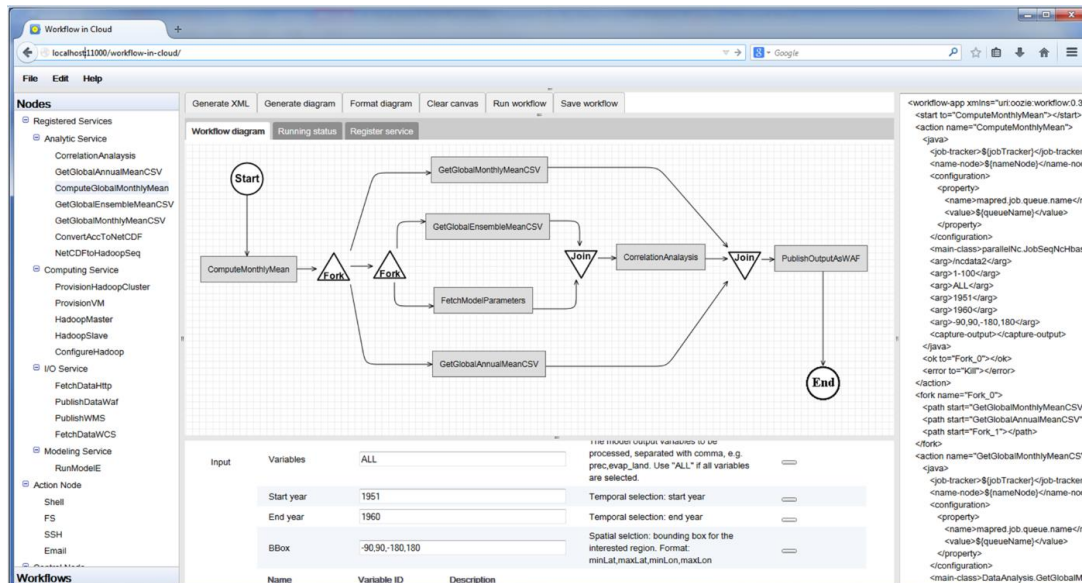


Figure 4.6 GUI of the web-based workflow builder

<sup>18</sup> <https://github.com/jabirahmed/oozie/tree/master/workflowgenerator>

## 4.5 Chapter Summary

This chapter proposes a service-oriented and cloud-based workflow framework to address the procedure complexity challenge. Methodologies for designing and implementing the framework are presented. By leveraging cloud computing, MapReduce, and SOA, this framework seamlessly integrates the proposed methodologies as a whole to form a scalable, reliable and interoperable workflow environment. Such a workflow environment enables scientists to transform complex geoscience experiment into intuitive diagram-based workflows by decoupling the experiment into reusable services. Serving as an integration platform for the GCI, the workflow framework is evaluated in Chapter 5 with ModelE sensitivity analysis.



## CHAPTER 5 INTEGRATION AND VALIDATION: MODELE SENSITIVITY ANALYSIS

### 5.1 Introduction

Chapter 2, 3, and 4 detailed the optimization methodologies for geospatial cyberinfrastructure by addressing three grand challenges of computing intensity, data intensity and procedure complexity facing climate studies. The optimization methodologies are evaluated in each chapter with prototype and experiment to demonstrate the efficiency of the proposed methods. This chapter elaborates how the optimization methodologies are integrated to efficiently conduct sensitivity analysis, thus to accelerate the scientific discovery in climate studies.

### 5.2 Services for the Study Case

Over ten services are developed following the unified service model proposed in the workflow framework (Chapter 4). These are registered to the workflow prototype to facilitate the 8 steps of the study case detailed in Chapter 1.

For climate simulation( step S1, S2 and S3), a *Model Service (RunModelE)* is developed based on Model as a Service proposed in Chapter 2 to setup and run ModelE automatically. *RunModelE* is also an *Infrastructure Service* since it provisions a virtual cluster with configured modeling environment to run models in parallel.

For climate data processing (step S5 and S6), two *Processing Services* are developed based on the big climate data analytic framework proposed in Chapter 3: *NetCDFtoHBase* decomposes the big climate data based on the spatiotemporal decomposition mechanism and subsequently upload to HBase; a MapReduce-enabled *Processing Service* computes the global monthly mean for all model output.

Finally, for S4 *AccToNetCDF* is a script-based service converting model out .acc files to NetCDF format; for S7 and S8 a Java-based *Processing Service* conducts linear regression analysis and plots the relationships for the most affected variables. To support input and output for the above services, *FetchDataHttp* downloads data from a web accessible folder or simply a URL to the workflow execution environment (WEE). *PublishDataWaf* publishes the data in the WEE to a web accessible folder.

### **5.3 Executable Workflow of the Study Case**

Once these services are registered, a executable workflow is built by visually dragging and connecting services in the *Web-based Workflow Builder* to conduct the experiment (Figure 5.1).

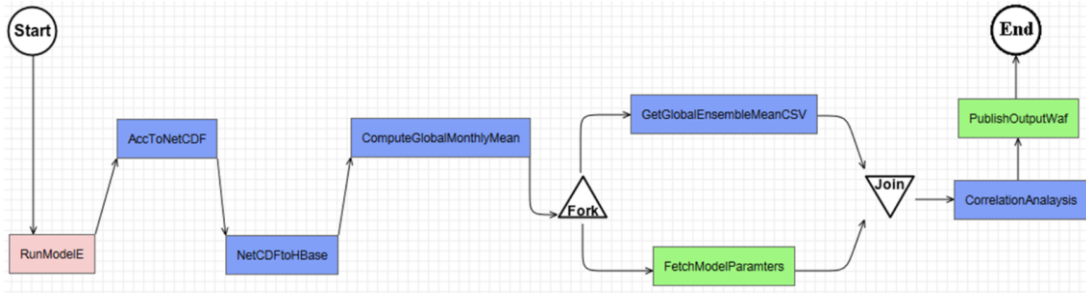


Figure 5.1 Executable workflow for the study case built in the prototype

In this workflow, *RunModelE* provisions virtual machines in cloud to conduct ensemble model-runs in parallel. When model-runs are finished, output are preprocessed and loaded to HBase with *ArcToNetCDF* and *NetCDFtoHBase*. Then global monthly mean for each output climate variable is calculated in parallel in the WEE with *ComputeGlobalMonthlyMean* service. Next, two services *GetGlobalEensembleMean* and *FetchModelParamters*, are executed in parallel. Once finished, *CorrelationAnalysis* service calculates linear regression statistics for each Parameter-Variable pair based on the variable ensemble mean values and the model input parameters. Finally, the workflow output (intermediate and final) is published on a web accessible folder (Figure 5.2).

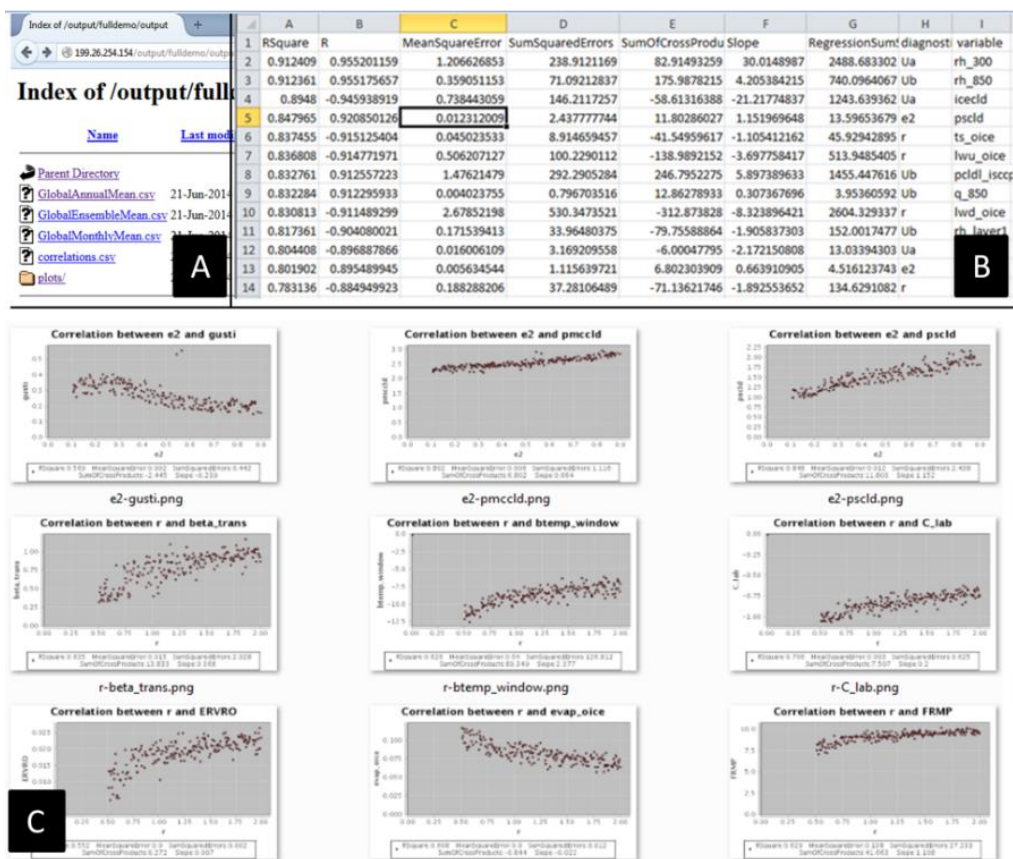


Figure 5.2 Workflow output published in a web accessible folder (A), presented as correlation statistics in CSV format (B), and plotted output climate variables highly affected by the seven model input parameters ( $R^2 > 0.6$ , 9 of 57 pairs)

To check the workflow running status/progress, one can login to the *Web-based Workflow Builder* (Chapter 4). Since *RunModelE* service normally takes days to finish, the status for each model-run can be monitored by logging into the MaaS web portal (Chapter 2) as illustrated in Figure 5.3.

Auto refresh		Refresh	Show running	Terminate all VMs							
Job name	VM type	VM status	VM launch time	Model Config	Task status	Task progress	Data produced(MB)	Simulation period	Running time(hours)	Time left(hours)	Explore data
14runs_task1	m1.small	Terminated,job finished	Wed Jul 10 23:38:45 EDT 2013	E4M20a_000047.R	Finished	100%	2088.1	1949-12 to 1961-1	125.940	0.000	Explore output
14runs_task10	m1.small	Terminated,job finished	Wed Jul 10 23:39:09 EDT 2013	E4M20a_000054.R	Finished	100%	2088.1	1949-12 to 1961-1	125.450	0.000	Explore output
14runs_task11	m1.small	Terminated,job finished	Wed Jul 10 23:39:11 EDT 2013	E4M20a_000051.R	Finished	100%	2088.1	1949-12 to 1961-1	125.400	0.000	Explore output
14runs_task12	m1.small	Terminated,job finished	Wed Jul 10 23:39:14 EDT 2013	E4M20a_000049.R	Finished	100%	2088.1	1949-12 to 1961-1	125.930	0.000	Explore output
14runs_task13	m1.small	Terminated,job finished	Wed Jul 10 23:39:17 EDT 2013	E4M20a_000055.R	Finished	100%	2088.1	1949-12 to 1961-1	124.980	0.000	Explore output
14runs_task14	m1.small	Terminated,job finished	Wed Jul 10 23:39:19 EDT 2013	E4M20a_000046.R	Finished	100%	2088.1	1949-12 to 1961-1	125.250	0.000	Explore output
14runs_task2	m1.small	Terminated,job finished	Wed Jul 10 23:38:48 EDT 2013	E4M20a_000044.R	Finished	100%	2088.1	1949-12 to 1961-1	125.650	0.000	Explore output
14runs_task3	m1.small	Terminated,job finished	Wed Jul 10 23:38:50 EDT 2013	E4M20a_000042.R	Finished	100%	2088.1	1949-12 to 1961-1	125.150	0.000	Explore output
14runs_task4	m1.small	Terminated,job finished	Wed Jul 10 23:38:53 EDT 2013	E4M20a_000050.R	Finished	100%	2088.1	1949-12 to 1961-1	125.610	0.000	Explore output
14runs_task5	m1.small	Terminated,job finished	Wed Jul 10 23:38:56 EDT 2013	E4M20a_000052.R	Finished	100%	2088.1	1949-12 to 1961-1	126.840	0.000	Explore output
14runs_task6	m1.small	Terminated,job finished	Wed Jul 10 23:38:58 EDT 2013	E4M20a_000045.R	Finished	100%	2088.1	1949-12 to 1961-1	125.920	0.000	Explore output
14runs_task7	m1.small	Terminated,job finished	Wed Jul 10 23:39:01 EDT 2013	E4M20a_000043.R	Finished	100%	2088.1	1949-12 to 1961-1	125.310	0.000	Explore output
14runs_task8	m1.small	Terminated,job finished	Wed Jul 10 23:39:03 EDT 2013	E4M20a_000048.R	Finished	100%	2088.1	1949-12 to 1961-1	126.040	0.000	Explore output
14runs_task9	m1.small	Terminated,job finished	Wed Jul 10 23:39:06 EDT 2013	E4M20a_000053.R	Finished	100%	2088.1	1949-12 to 1961-1	125.020	0.000	Explore output
3223_task1	m1.small	force terminated	Wed Jul 10 22:42:16 EDT 2013	E4M20a_000003.R	Force terminated	0%	0	1949-12 to 1961-1	-0.090	155.700	Explore output
3223_task2	m1.small	force terminated	Wed Jul 10 22:42:19 EDT 2013	E4M20a_000002.R	Force terminated	0%	0	1949-12 to 1961-1	-0.090	155.700	Explore output

Figure 5.3 Monitor the status of model-run in a web-based application

Once model-runs are finished, model outputs can be analyzed and visualized directly through the integrated online visual analytic system without downloading and managing the data on a local computer. The online analytic system is powered by the big climate data analytic framework proposed in Chapter 3. Figure 5.4 shows the yearly mean “net thermal radiation” for three simulation scenarios of a selected study area from 1951 to 1960.

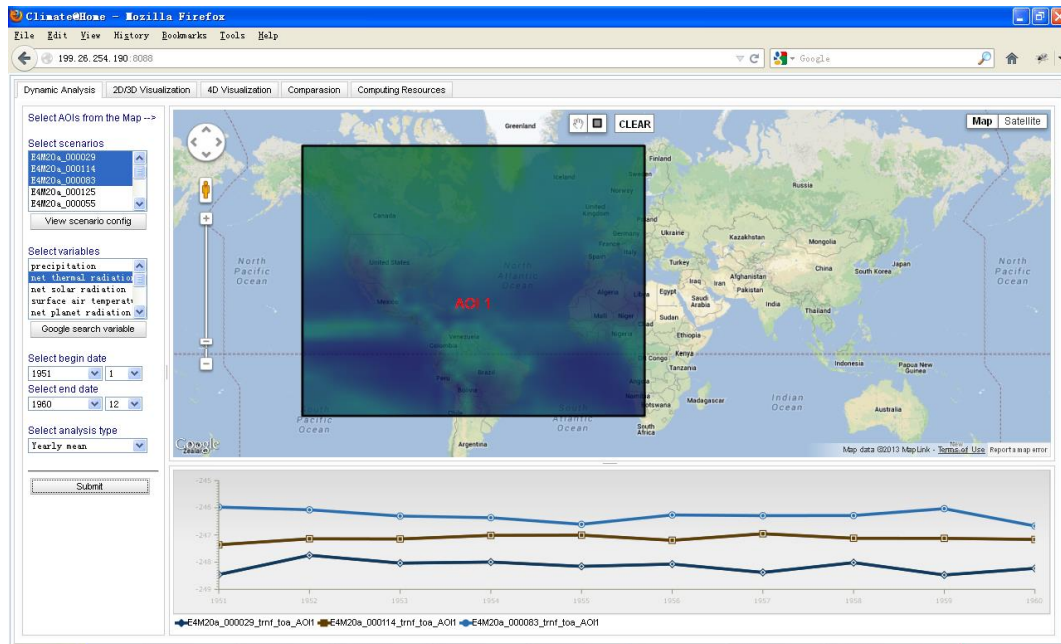


Figure 5.4 Yearly mean “net thermal radiation” for three simulation scenarios for a select study area

## 5.4 Chapter Summary

By integrating the optimization methodologies, the ModelE sensitivity analysis experiment is transformed into an intuitive diagram-based executable workflow, which reduces the experiment time from 2 months to 5 days with a few simple clicks.

## **CHAPTER 6 CONCLUSIONS AND FUTURE RESEARCH**

### **6.1 Conclusions**

This dissertation optimizes geospatial cyberinfrastructure from three computational aspects to help answer the climate question of “How to efficiently search through this multi-dimensional parameter space to identify which parameters are more sensitive to simulated climate changes?”. The optimized GCI helps scientists conduct climate studies in a more efficient way by transforming the complex experiment into an executable workflow. The efficiency is demonstrated in the study case of ModelE sensitivity analysis, for which the experiment time is reduced from 2 months to 5 days after using the optimized GCI. Such a significant improvement is achieved by addressing the three computational challenges of computing intensity, data intensity and procedure complexity in a systematic and integrative manner.

- For computing intensity, MaaS is proposed to on-demand provision a cluster of virtual machines with pre-configured model environment and parameter configuration to conduct ensemble model-runs in parallel. With MaaS, climate models are published as services, and these services can be accessed through a simple web interface.
- For data intensity, a scalable big spatiotemporal data analytics framework is proposed to process big model output efficiently. By exploring the decomposition

mechanism and parallelization technique for general spatiotemporal data, this framework employs MapReduce, No-SQL database and cloud computing to efficiently process big climate data in an auto-scalable computing environment.

- For procedure complexity, the service-oriented cloud-based scientific workflow framework is developed to enable scientists to conduct complex experiment by dragging and connecting steps in a visually intuitive diagram. By incorporating cloud computing, the workflow framework enables on-demand provision computing resources during the workflow execution.

Finally, the three optimization methodologies are seamlessly integrated into the workflow framework through the proposed unified service model. This dissertation provides a valuable guideline to bridge the computing infrastructure and computing requirements for climate studies. Since the challenges of computing intensity, data intensity and procedure complexity are quite common in geosciences, the proposed optimization methodologies can be easily broaden from the climate science to much broader geoscience domains.

## **6.2 Future Research**

Further research is desired to improve the optimization techniques from several aspects.

- The current MaaS supports parallelization in the experiment-level but is not optimized for models that run on a HPC cluster such as NMM-dust model (Xie et al. 2010). Exploring possible approaches to automatically provision a HPC cluster in a



cloud environment is essential for supporting MPI-enabled model parallelization.

Huang et al (2013) introduced detailed steps to manually provision a HPC cluster on Amazon EC2 to run the MPI-enabled NMM-dust model in parallel.

- The current MaaS architecture enables users to manually upload the model input files. A more comprehensive mechanism would be to link with online datasets integrated in the model configuration for model to fetch data from data server directly. Future research on accepting standard geospatial web services as model input/output to support model chaining and scientific workflow are needed (e.g., adopting OGC Sensor Observation Service (SOS) for real time observation data input, Web Coverage Service (WCS) for raster data input/output).
- The big climate data processing framework for data storage currently uses virtual storage attached to the VMs to form the HDFS. The storage attached to each VM is of two types. The first is virtualized directly from the physical machine on which the VM is hosted, and the stored data are accessible directly by the VM without going through any network. However, such storage is not permanent, and data are lost with the termination of the VM. The second storage type is virtualized from a storage cluster connected to the cloud platform and persists even when the VM is terminated. However, since the storage is from a storage cluster instead of the VM's host machine, the VM needs to frequently access the data through network. Therefore, neither storage type is optimized for the framework. Further study is desired to explore a new storage mechanism to support both local access and persistence.

- Integrating cloud cost model in the geospatial cyberinfrastructure to archive the pay-as-you-go style for using cloud resources is desirable. For example, a cost model in MaaS could assess the potential cost and recommend on what kind of VMs should be launched based on the model types and configurations.

## REFERENCES

- Anderson, D. P. (2004). Boinc: A system for public-resource computing and storage. In *Grid Computing Proceedings. Fifth IEEE/ACM International Workshop* (pp. 4-10). IEEE.
- Anderson, D. P., Cobb, J., Korpela, E., Lebofsky, M., & Werthimer, D. (2002). SETI@home: an experiment in public-resource computing. *Communications of the ACM*, 45(11), 56-61.
- Barga R, Jackson J, Araujo N, Guo D, Gautam N, et al.(2008) The trident scientific workflow workbench. *IEEE*. pp. 317-318.
- Bastin, L., Cornford, D., Jones, R., Heuvelink, G., Pebesma, E., Stasch, C., ... & Williams, M. (2013). Managing uncertainty in integrated environmental modeling: The UncertWeb framework. *Environmental Modelling & Software*, 39, 116-134.
- Bavoil L, Callahan SP, Crossno PJ, Freire J, Scheidegger CE, et al.(2005) Vistrails: Enabling interactive multiple-view visualizations *IEEE*. pp. 135-142.
- Beckman, P. H. (2005) Building the TeraGrid. *Philosophical Transactions of the Royal Society A: Mathematical, Physical and Engineering Sciences*, 363(1833), 1715-1728.
- Bernholdt, D., Bharathi, S., Brown, D., Chanchio, K., Chen, M., Chervenak, A., ... & Williams, D. (2005). The earth system grid: Supporting the next generation of climate modeling research. *Proceedings of the IEEE*, 93(3), 485-495.
- Chang F, Dean J, Ghemawat S, Hsieh WC, Wallach DA, et al. (2008a) Bigtable: A distributed storage system for structured data. *ACM Transactions on Computer Systems (TOCS)* 26: 4.
- Chang, H. I., Niyogi, D., Chen, F., Kumar, A., Song, C., Zhao, L., ... & Scheeringa, K. (2008b). Developing a TeraGrid Based Land Surface Hydrology and Weather Modeling Interface. In *Proceedings of the TeraGrid 2008 Conference*.
- Chen J, Zheng G, Chen H (2013) ELM-MapReduce: MapReduce accelerated extreme learning machine for big spatial data analysis; *IEEE*. pp. 400-405.
- Cinquini L, Crichton D, Mattmann C, Bell GM, Drach B, et al. (2012) The Earth System Grid Federation: An open infrastructure for access to distributed geospatial data *IEEE*. pp. 1-10.

- Dean J, Ghemawat S (2008) MapReduce: simplified data processing on large clusters. *Communications of the ACM* 51: 107-113.
- Deelman E, Gil Y, Zemankova M (2006) NSF workshop on the challenges of scientific workflows 1-2.
- Dongarra J (2011) The international exascale software project roadmap. *International Journal of High Performance Computing Applications*: 1094342010391989.
- Duffy DQ, Schnase JL, Thompson JH, Freeman SM, Clune TL (2012) Preliminary Evaluation of MapReduce for High-Performance Climate Data Analysis. NASA new technology report white paper.
- Edwards PN (2010) *A vast machine: Computer models, climate data, and the politics of global warming*: MIT Press.
- Evangelinos, C., & Hill, C. (2008) Cloud Computing for parallel Scientific HPC Applications: Feasibility of running Coupled Atmosphere-Ocean Climate Models on Amazon's EC2. In *The 1st Workshop on Cloud Computing and its Applications*, (pp.2-34).
- Fernández-Quiruelas, V., Fernández, J., Cofiño, A. S., Fita, L., & Gutiérrez, J. M. (2011) Benefits and requirements of grid computing for climate applications. An example with the community atmospheric model. *Environmental Modelling & Software*, 26(9), 1057-1069.
- Ferraro, R., Sato, T., Brasseur, G., Deluca, C., & Guilyardi, E. (2003). Modeling the Earth system. Critical computational technologies that enable us to predict our planet's future. In *Geoscience and Remote Sensing Symposium. IGARSS'03. Proceedings. 2003 IEEE International* (pp. 630-633). IEEE.
- Fiore S, Negro A, Aloisio G (2012) The Climate-G Portal: The context, key features and a multi-dimensional analysis. *Future Generation Computer Systems* 28: 1-8.
- Foster I, Kesselman C, Tuecke S (2001) The anatomy of the grid: Enabling scalable virtual organizations. *International journal of high performance computing applications* 15: 200-222.
- Gao, S., Li, L., Li, W., Janowicz, K., & Zhang, Y. (2014) Constructing gazetteers from volunteered big geo-data based on Hadoop. *Computers, Environment and Urban Systems*.
- Geist A (1994) *PVM: Parallel virtual machine: a users' guide and tutorial for networked parallel computing*: MIT press.
- Geller, G. N., & Melton, F. (2008) Looking forward: Applying an ecological model web to assess impacts of climate change. *Biodiversity*, 9(3-4), 79-83
- Geller, G. N., & Turner, W. (2007) The model web: a concept for ecological forecasting. In *Geoscience and Remote Sensing Symposium. IGARSS 2007. IEEE International* (pp. 2469-2472). IEEE.

- Gil Y (2008) From data to knowledge to discoveries: Scientific workflows and artificial intelligence. *Scientific Programming* 16: 4
- Goodall, J. L., Robinson, B. F., & Castronova, A. M. (2011) Modeling water resource systems using a service-oriented computing paradigm. *Environmental Modelling & Software*, 26(5), 573-582.
- Gropp W, Lusk E, Skjellum A (1999) Using MPI: portable parallel programming with the message-passing interface: MIT press.
- Grosslight, L., Unger, C., Jay, E., & Smith, C. L. (1991) Understanding models and their use in science: Conceptions of middle and high school students and experts. *Journal of Research in Science teaching*, 28(9), 799-822.
- Harrop, C. W., Bernardet, L., Govett, M., Smith, J. S., & Weygandt, S. (2008) A Workflow Management System for Automating Weather and Climate Simulations. In *CIRES'Annual, Institute-wide Symposium*.
- Herodotou, H., Dong, F., & Babu, S., 2011 No one (cluster) size fits all: automatic cluster sizing for data-intensive analytics. In *Proceedings of the 2nd ACM Symposium on Cloud Computing* (p. 18). ACM. October 26-28, 2011, Cascais, Portugal
- Hoffa C, Mehta G, Freeman T, Deelman E, Keahey K, et al. (2008) On the use of cloud computing for scientific workflows *IEEE*. pp. 640-645.
- Huang Q, Yang C, Liu K, Xia J, Xu C, et al. (2013a) Evaluating open-source cloud computing solutions for geosciences. *Computers & Geosciences* 59: 41-52.
- Huang Q., Li Z., Liu K., Xia J., Jiang Y., Xu C., Yang C.(2013b) Chapter 16 handling intensities of data, computation, concurrent access, and spatiotemporal patterns, In *Spatial Cloud Computing: a practical approach*, edited by Yang C., Huang Q., Li Z., Xu C., Liu K., CRC Press: pp. 275-293.
- Huang, Q., & Yang, C. (2011) Optimizing grid computing configuration and scheduling for geospatial analysis: An example with interpolating DEM. *Computers & Geosciences*, 37(2), 165-176.
- Huang, Q., Yang, C., Benedict, K., Chen, S., Rezgui, A., & Xie, J. (2013c) Utilize cloud computing to support dust storm forecasting. *International Journal of Digital Earth*, 6(4), 338-355.
- Huang, Q., Yang, C., Nebert, D., Liu, K., & Wu, H. (2010) Cloud computing for geosciences: deployment of GEOSS clearinghouse on Amazon's EC2. In *Proceedings of the ACM SIGSPATIAL International Workshop on High Performance and Distributed Geographic Information Systems* (pp. 35-38). ACM.
- Jacobson, M. J., & Wilensky, U. (2006) Complex systems in education: Scientific and educational importance and implications for the learning sciences. *The Journal of the Learning Sciences*, 15(1), 11-34.

- Juve G, Deelman E, Vahi K, Mehta G, Berriman B, et al.(2009) Scientific workflow applications on Amazon EC2 IEEE. pp. 59-66.
- Kaushik, R. T., & Bhandarkar, M. (2010) GreenHDFS: Towards an Energy-Conserving Storage-Efficient, Hybrid Hadoop Compute Cluster. In Proceedings of the USENIX Annual Technical Conference. June 23–25, 2010, Boston, USA
- Keenlyside, N. S., Latif, M., Jungclaus, J., Kornblueh, L., & Roeckner, E. (2008) Advancing decadal-scale climate prediction in the North Atlantic sector. *Nature*, 453(7191), 84-88.
- Khetrapal A, Ganesh V (2006) HBase and Hypertable for large scale distributed storage systems. Dept of Computer Science, Purdue University. Available at: <http://cloud.pubs.dbs.uni-leipzig.de/sites/cloud.pubs.dbs.uni-leipzig.de/files/Khetrapal2008HBaseandHypertableforlargescaledistributedstorage.pdf> . Accessed on 27 December 2014
- Krishnan, S., Baru, C., & Crosby, C. (2010) Evaluation of MapReduce for gridding LIDAR data. In Cloud Computing Technology and Science (CloudCom), 2010 IEEE Second International Conference on (pp. 33-40). IEEE.
- Lakshman A, Malik P (2010) Cassandra: a decentralized structured storage system. *ACM SIGOPS Operating Systems Review* 44: 35-40.
- Leverich, J., & Kozyrakis, C. (2010) On the energy (in) efficiency of hadoop clusters. *ACM SIGOPS Operating Systems Review*, 44(1), 61-65.
- Li J., Li, Z., Sun M., Liu K.(2013) Cloud-enabling Climate@Home. In Yang C., Huang Q., Li Z., Xu C., Liu K.,(Eds.), *Spatial cloud computing: a practical approach* (pp. 143-160). CRC Press/Taylor & Francis
- Li Z, Yang C, Huang Q, Liu K, Sun M, et al. (2014) Building Model as a Service to support geosciences. *Computers, Environment and Urban Systems*. doi:10.1016/j.compenvurbsys.2014.06.004
- Li, W., Li, L., Goodchild, M. F., & Anselin, L. (2013) A geospatial cyberinfrastructure for urban economic analysis and spatial decision-making. *ISPRS International Journal of Geo-Information*, 2(2), 413-431.
- Li, Z., Yang, C. P., Wu, H., Li, W., & Miao, L. (2011) An optimized framework for seamlessly integrating OGC Web Services to support geospatial sciences. *International Journal of Geographical Information Science*, 25(4), 595-613.
- Li, Z., Yang, C., Jin, B., Yu, M., Liu, K., Sun, M., & Zhan, M. (2015) Enabling Big Geoscience Data Analytics with a Cloud-Based, MapReduce-Enabled and Service-Oriented Workflow Framework. *PloS one*, 10(3), e0116781.
- Li, Z., Yang, C., Sun, M., Li, J., Xu, C., Huang, Q., & Liu, K. (2013) A high performance web-based system for analyzing and visualizing spatiotemporal data for climate studies. In *Web and Wireless Geographical Information Systems* (pp. 190-198). Springer Berlin Heidelberg, Banff, AL, Canada, April 4-5, 2013.

- Liang, S. H., & Huang, C. Y. (2013) GeoCENS: A geospatial cyberinfrastructure for the world-wide sensor web. *Sensors*, 13(10), 13402-13424.
- Lin, F. C., Chung, L. K., Wang, C. J., Ku, W. Y., & Chou, T. Y. (2013) Storage and processing of massive remote sensing images using a novel cloud computing platform. *GIScience & Remote Sensing*, 50(3), 322-336.
- Liu Y, Chen B, He W, Fang Y (2013a) Massive image data management using HBase and MapReduce. *IEEE*. pp. 1-5.
- Liu,K., Huang,Q., Xia,J.(2013) Chapter 5 Cloud-enabling geoscience applications, 2013b. In *Spatial Cloud Computing: a practical approach*, edited by Yang, C., Huang, Q., Li, Z., Xu, C., Liu, K. CRC Press: pp. 73-89.
- Lud äscher B, Altintas I, Berkley C, Higgins D, Jaeger E, et al. (2006) Scientific workflow management and the Kepler system. *Concurrency and Computation: Practice and Experience* 18: 1039-1065.
- Maheshwari, N., Nanduri, R., & Varma, V. (2012) Dynamic energy efficient data placement and cluster reconfiguration algorithm for MapReduce framework. *Future Generation Computer Systems*, 28(1), 119-127.
- Majithia S, Shields M, Taylor I, Wang I (2004) Triana: A graphical web service composition and execution toolkit *IEEE*. pp. 514-521.
- Mell, P., & Grance, T.(2009) The NIST definition of cloud computing. *National Institute of Standards and Technology*, 53(6), 50.
- Murphy JM, Sexton DM, Barnett DN, Jones GS, Webb MJ, et al. (2004) Quantification of modelling uncertainties in a large ensemble of climate change simulations. *Nature* 430: 768-772.
- Nativi, S., Mazzetti, P., & Geller, G. N. (2013) Environmental model access and interoperability: The GEO Model Web initiative. *Environmental Modelling & Software*, 39, 214-228.
- Nefedova, V., Jacob, R., Foster, I., Liu, Z., Liu, Y., Deelman, E., ... & Vahi, K. (2006) Automating climate science: Large ensemble simulations on the TeraGrid with the GriPhyN Virtual Data System. In *e-Science and Grid Computing, 2006. e-Science'06. Second IEEE International Conference* (pp. 32-32). *IEEE*.
- NSF, Earth Cube Guidance for the Community, (2011) Available at: <http://www.nsf.gov/pubs/2011/nsf11085/nsf11085.pdf> (accessed on May 17, 2013)
- Oinn T, Addis M, Ferris J, Marvin D, Senger M, et al. (2004) Taverna: a tool for the composition and enactment of bioinformatics workflows. *Bioinformatics* 20: 3045-3054.

- Ostermann, S., Iosup, A., Yigitbasi, N., Prodan, R., Fahringer, T., & Epema, D. (2010) A performance analysis of EC2 cloud computing services for scientific computing. In *Cloud Computing* (pp. 115-131). Springer Berlin Heidelberg.
- Pordes, R., Petravick, D., Kramer, B., Olson, D., Livny, M., Roy, A., ... & Quick, R. (2007) The open science grid. In *Journal of Physics: Conference Series*, 78(1), (pp. 012057). IOP Publishing.
- Porter JH (2000) Scientific databases. In: Michener W. K., & Brunt J. W. editors. *Ecological data: Design, management and processing*: 48-69.
- Rizvandi NB, Boloori AJ, Kamyabpour N, Zomaya AY(2011) MapReduce implementation of prestack Kirchhoff time migration (PKTM) on seismic data. *IEEE*. pp. 86-91.
- Roman, D., Schade, S., Berre, A. J., Bodsberg, N. R., & Langlois, J. (2009) Model as a service (MaaS). In *AGILE Workshop: Grid Technologies for Geospatial Applications*, Hannover, Germany.
- Rõme T. (2010) Autoscaling Hadoop Clusters, MSc thesis, University of Tartu.
- Schnase JL, Duffy DQ, Tamkin GS, Nadeau D, Thompson JH, et al. (2014) MERRA analytic services: Meeting the big data challenges of climate science through cloud-enabled climate analytics-as-a-service. *Computers, Environment and Urban Systems* (doi:10.1016/j.compenvurbsys.2013.12.003).
- Schnase, J. L., Duffy, D. Q., Tamkin, G. S., Nadeau, D., Thompson, J. H., Grieg, C. M., ... & Webster, W. P. (2014) MERRA analytic services: Meeting the big data challenges of climate science through cloud-enabled climate analytics-as-a-service. *Computers, Environment and Urban Systems*.
- Schmidt, G. A., Kelley, M., Nazarenko, L., Ruedy, R., Russell, G. L., Aleinov, I., ... & Zhang, J. (2014) Configuration and assessment of the GISS ModelE2 contributions to the CMIP5 archive. *Journal of Advances in Modeling Earth Systems*, 6(1), 141-184.
- Simmhan Y, Barga R, van Ingen C, Lazowska E, Szalay A(2009) Building the trident scientific workflow workbench for data management in the cloud *IEEE*. pp. 41-50.
- Skamarock, W. C., and J. B. Klemp. (2008) A time-split nonhydrostatic atmospheric model for weather research and forecasting applications. *Journal of Computational Physics* 227, no. 7 : 3465-3485.
- Skytland, N. (2012) Big data: What is nasa doing with big data today. *Open. Gov open access article*.
- Stonebraker M (2010) SQL databases v. NoSQL databases. *Communications of the ACM* 53: 10-11.



- Sun M., Li J., Yang C., Schmidt G.A., Bambacus M., Cahalan R., Huang Q., Xu C., Noble E.U., Li Z. (2012) A Web-Based Geovisual Analytical System for Climate Studies. *Future Internet*, 4(4):1069-1085.
- Taylor IJ, Deelman E, Gannon D, Shields M (2007) *Workflows for e-Science*: Springer-Verlag London Limited.
- Vecchiola, C., Pandey, S., & Buyya, R. (2009) High-performance cloud computing: A view of scientific applications. In *Pervasive Systems, Algorithms, and Networks (ISPAN)*, 2009 10th International Symposium (pp. 4-16). IEEE.
- Wang S (2010) A CyberGIS framework for the synthesis of cyberinfrastructure, GIS, and spatial analysis. *Annals of the Association of American Geographers* 100: 535-557.
- Williams DN, Drach R, Ananthakrishnan R, Foster I, Fraser D, et al. (2009) The Earth System Grid: Enabling access to multimodel climate simulation data. *Bulletin of the American Meteorological Society* 90: 195-205.
- Wright D. J., & Wang S. (2011) The emergence of spatial cyberinfrastructure. *Proceedings of the National Academy of Sciences* 108(14), 5488-5491.
- Xie, J., Yang, C., Zhou, B., & Huang, Q. (2010). High-performance computing for the simulation of dust storms. *Computers, Environment and Urban Systems*, 34(4), 278-290.
- Yang C, Raskin R, Goodchild M, Gahegan M (2010) Geospatial cyberinfrastructure: past, present and future. *Computers, Environment and Urban Systems* 34: 264-277.
- Yang C, Wu H, Huang Q, Li Z, Li J (2011b) Using spatial principles to optimize distributed computing for enabling the physical science discoveries. *Proceedings of the National Academy of Sciences* 108: 5498-5503.
- Yang C., Sun M., Liu K., Huang Q., Li Z., Gui Z., Jiang Y., Xia J., Yu M., Xu C., Lostritto P., Zhou N. (2014) Contemporary Computing Technologies for Processing Big Spatiotemporal Data, in *Space-Time Integration in Geography and GIScience: Research Frontiers in the U.S. and China*. Mei-Po Kwan, Douglas Richardson, Donggen Wang and Chenghu Zhou (Eds), Dordrecht: Springer (in press).
- Yang, C. P., Cao, Y., Evans, J., Kafatos, M., & Bambacus, M. (2006) Spatial Web portal for building spatial data infrastructure. *Geographic Information Sciences*, 12(1), 38-43.
- Yang, C., Goodchild, M., Huang, Q., Nebert, D., Raskin, R., Xu, Y., ... & Fay, D. (2011a) Spatial cloud computing: how can the geospatial sciences use and help shape cloud computing? *International Journal of Digital Earth*, 4(4), 305-329.

- Yang, C., Xu, Y., & Nebert, D. (2013) Redefining the possibility of digital Earth and geosciences with spatial cloud computing. *International Journal of Digital Earth*, 6(4), 1-16.
- Yue P, He L (2009) Geospatial data provenance in cyberinfrastructure. *IEEE*. pp. 1-4.
- Zhang H, Liu M, Shi Y, Yuen DA, Yan Z, et al. (2007) Toward an automated parallel computing environment for geosciences. *Physics of the Earth and Planetary Interiors* 163: 2-22.
- Zhang, T., & Tsou, M. H. (2009) Developing a grid-enabled spatial Web portal for Internet GIServices and geospatial cyberinfrastructure. *International Journal of Geographical Information Science*, 23(5), 605-630.
- Zhao H, Ai S, Lv Z, Li B (2010) Parallel accessing massive NetCDF data based on mapreduce. *Web Information Systems and Mining: Springer*. pp. 425-431

## **BIOGRAPHY**

Zhenlong Li received his Bachelor of Engineering in Geographic Information Systems from Wuhan University, China, 2006. He then joined Heilongjiang Bureau of Surveying and Mapping (HLJBSM) as a geospatial software engineer. In 2010, he received his Master degree in Earth System Science from the Department of Geography and Geoinformation Science at George Mason University (GMU). In 2015, he received his Doctorate in Earth Systems and Geoinformation Sciences from GMU. Zhenlong has great passion for research and teaching, and will join University of South Carolina in Fall 2015 as Assistant Professor in the Department of Geography.