

ABSTRACT

Title of Dissertation: ADDRESSING GEOGRAPHICAL
CHALLENGES IN THE BIG DATA ERA
UTILIZING CLOUD COMPUTING

Hai LAN, Doctor of Philosophy, 2020

Dissertation directed by: Professor, Kathleen Stewart,
Department of Geographical Sciences

Processing, mining and analyzing big data adds significant value towards solving previously unverified research questions or improving our ability to understand problems in geographical sciences. This dissertation contributes to developing a solution that supports researchers who may not otherwise have access to traditional high-performance computing resources so they benefit from the “big data” era, and implement big geographical research in ways that have not been previously possible. Using approaches from the fields of geographic information science, remote sensing and computer science, this dissertation addresses three major challenges in big geographical research: 1) how to exploit cloud computing to implement a universal scalable solution to classify multi-sourced remotely sensed imagery datasets with high efficiency; 2) how to overcome the missing data issue in land use land cover

studies with a high-performance framework on the cloud through the use of available auxiliary datasets; and 3) the design considerations underlying a universal massive scale voxel geographical simulation model to implement complex geographical systems simulation using a three dimensional spatial perspective. This dissertation implements an in-memory distributed remotely sensed imagery classification framework on the cloud using both unsupervised and supervised classifiers, and classifies remotely sensed imagery datasets of the Suez Canal area, Egypt and Inner Mongolia, China under different cloud environments. This dissertation also implements and tests a cloud-based gap filling model with eleven auxiliary datasets in biophysical and social-economics in Inner Mongolia, China. This research also extends a voxel-based Cellular Automata model using graph theory and develops this model as a massive scale voxel geographical simulation framework to simulate dynamic processes, such as air pollution particles dispersal on cloud.

ADDRESSING GEOGRAPHICAL CHALLENGES IN THE BIG DATA ERA
UTILIZING CLOUD COMPUTING

by

Hai Lan

Dissertation submitted to the Faculty of the Graduate School of the
University of Maryland, College Park, in partial fulfillment
of the requirements for the degree of
Doctor of Philosophy
2020

Advisory Committee:
Professor Kathleen Stewart, Chair
Professor Leila De Florian
Professor Tatiana Loboda
Professor Yichun Xie
Professor Xin-Zhong Liang

© Copyright by
Hai Lan
2020

Dedication

To Lianfang Zhao (赵莲芳), Jintang Lan (兰金堂),

Yaqi Wang (王亚琦), and Colin Lan (兰皓洋)

For their love, support, encouragement, and companionship

Acknowledgements

First and foremost, I would like to express my sincere gratitude to my advisor, Prof. Kathleen Stewart. She accepted me at the most difficult time during my academic career. Without her guidance, strongly support and encouragement, it is impossible for me to finish the long journey of pursuing my PhD. It is my honor and lucky to have you as one of the most important persons in my life who is keeping me on the right track and overcome obstacles in scientific research.

I would also thank to Prof. Yichun Xie from Eastern Michigan University. He led me to the door of scientific research and has kept advising me since I started my GIS studies in the U.S. He also provided the ground survey dataset to support the experiments in Chapter 2 and Chapter 3 of my dissertation.

My sincere thanks also go to other members in my dissertation advisory committee. Thanks to Prof. Tatiana Loboda for advising me and correcting me many times from the perspective of remote sensing and for giving me a lot of very valuable comments on my dissertation draft. She guided me on how to organize the remote sensing parts in my dissertation in the correct direction. I would also thank Prof. Leila De Floriani and Prof. Xin-Zhong Liang for spending time to talk with me and advising me on how to improve this dissertation and prepare for the defense presentation.

Special thanks to Prof. Michael Batty, CASA of University College London, Prof. Jimmy Lin, Chair in the School of Computer Science at the University of Waterloo and Prof. Huy T. Vo in the Department of Computer Science and

Engineering, New York University. Thank you for your strongly support, enlightenment and advising.

Thanks to all the colleagues and friends, Cheng, Xin, Yanjia, Junchun, Yao, Zhiyue and Zheng in Center for Geospatial Information Science, UMD. It is a great pleasure to work with them and thanks for their support and companionship. My thanks to Ruibo Han, Jack Ma, and Jonathan Resop to offer me the chance to work with you as a TA in the MPGIS program. I would also thank Rachel Berndtson, who helped me a lot with patience to administrative issues.

Last but not the least, I would like to thank my wife Yaqi, my son Colin and my parents for their deepest love and unconditional support. It is impossible to finish this long march without them.

Table of Contents

Dedication.....	ii
Acknowledgements.....	iii
Table of Contents.....	v
List of Tables.....	vii
List of Figures.....	viii
List of Abbreviations.....	x
Chapter 1 : Introduction.....	1
1. Background and Motivation.....	1
2. Dissertation Structure.....	6
2.1 Chapter 2: A Cloud-based Computing Framework to Unfold Processing Efficiencies for Pixel-Wise Remotely Sensed Imagery Data Classification.....	8
2.2 Chapter 3: Data Gap Filling for Land Use and Land Cover Study using Cloud-based Distributed Markov Chain Cellular Automata.....	9
2.3 Chapter 4: Massive Voxel Cellular Automata Using Giraph: Application to Air Pollutant Particles Dispersal.....	10
Chapter 2 : A Cloud-based Computing Framework to Unfold Processing Efficiencies for Pixel-wise Remotely Sensed Imagery Data Classification.....	13
1. Introduction.....	13
1.1 The Rapid Expansion of Remotely Sensed Data in Big Data Era.....	13
1.2 Computing as A Resource for Big Data Processing.....	16
1.3 Cloud Computing for Processing Remotely Sensed Images.....	19
2. Study Area and Dataset.....	23
3. Method.....	29
3.1 Cloud-based Distributed NDVI/NDWI Threshold Classification.....	29
3.2 Cloud-based Distributed SVM Classification.....	37
4. Results and Discussion.....	40
4.1 Experiment Environment.....	40
4.2 Experiments Results.....	41
5. Conclusions.....	50
Chapter 3 : Data Gap Filling for Land Use and Land Cover Study Using Cloud-based Distributed Markov Chain Cellular Automata.....	53
1. Introduction.....	53
1.1 Significance of Land Use and Land Cover Change.....	53

1.2 Missing Data in LULC Change Research.....	54
1.3 Gap Filling Modeling Approaches.....	56
2. Study Area and Dataset.....	59
2.1 Study Area	59
2.2 Data Availability.....	63
3. Method	64
3.1 Workflow	64
3.2 LULC Classification	67
3.3 Sub-regions Clustering.....	68
3.4 Implementing Cloud-based Markov Chain - CA Model	71
4. Results and Discussion	80
4.1 Experiment Environment.....	80
4.2 Simulation Results and Accuracy Assessment	81
5. Conclusions.....	91
Chapter 4 : Massive Voxel Cellular Automata Using Giraph: A Use Case of Air Pollutant Particles Dispersal	93
1. Introduction.....	93
2. Big Voxel CA Implementation	97
2.1 HPC Solutions.....	99
2.2 Cloud Solutions.....	100
3. Giraph-based Cellular Automata	104
3.1 Rethink CA in Giraph	104
3.2 Implementing Massive GoL CA with Giraph.....	107
3.3 Performance Tests and Discussion	108
4. Massive Scale Voxel Air Pollutant Particle Dispersal Simulation	114
4.1 Physical Model.....	115
4.2 Graph-Based Voxel Air Pollutant CA	117
4.3 Experiment Results and Discussion.....	120
5. Conclusions.....	129
Chapter 5 : Conclusions and Future Work.....	131
1. Conclusions and Limitations.....	131
2. Significant contributions.....	136
3. Future work.....	139
Bibliography	141

List of Tables

Table 2-1: Detailed acquisition time of each dataset.	27
Table 2-2. Computing environment for Landsat 8 classification on local cloud environment	40
Table 2-3. Computing environment for MODIS classification on AWS	41
Table 2-4. Data size and processing time	42
Table 3-1. IMAR LULC classes including grassland subclasses	60
Table 3-2. Suitability factors.....	74
Table 3-3. Computing environment.....	81
Table 3-4. Transition matrix	81
Table 3-5. Overall accuracy assessment without using sub-region strategy	86
Table 3-6. Overall accuracy assessment by using sub-region strategy.....	86
Table 3-7. Sub-regions accuracy assessment.....	87
Table 4-1. Computing environment for 1 trillion cells GoL experiment.....	108
Table 4-2. Computing environment for air pollutant CA experiments.....	122
Table 4-3. A comparison of Jjumba’s study and ours	127

List of Figures

Figure 1-1. Conceptualization model of the three studies in this dissertation organized in two research perspective.....	7
Figure 2-1. Study area at SC area with remotely sensed imagery dataset coverage at three different scales	26
Figure 2-2. Study area at IMAR	28
Figure 2-3. <i>Spark</i> -based remote sensing image processing workflow	31
Figure 2-4. Algorithm for NDVI/NDWI threshold classification	33
Figure 2-5. Algorithm for result visualization	35
Figure 2-6. Detailed workflow for images classification and visualization	36
Figure 2-7. Algorithm for <i>Spark</i> -based distributed SVM classification.....	39
Figure 2-8. Visualization of NDVI/NDWI threshold classification results for time series from 2013 to 2017.....	44
Figure 2-9. Processing time of large-size Landsat dataset under different configurations	46
Figure 2-10. Visualization of SVM classification result in IMAR.....	47
Figure 3-1. Landsat 8 OLI images at IMAR from on August 2016	60
Figure 3-2. Data gap in IMAR on August 2016	62
Figure 3-3. Workflow for cloud based LULC gap filling process.....	65
Figure 3-4. Six clusters generated by county-based multivariate geographic k-medoids clustering analysis	70
Figure 3-5. Algorithm of distributed calculating probability matrix using Apache <i>Spark</i>	73
Figure 3-6. Algorithm of distributed processing Markov CA model with Apache <i>Giraph</i>	79
Figure 3-7. Overall results. (a) actual LULC mapping for 2016 and (b) LULC mapping with simulated gap filling for 2016.....	85
Figure 3-8. LULC maps of three grassland areas: (A) simulated Hinggan League area (meadow grassland), (B) real Hinggan League area, (C) simulated Xilingol League area (steppe grassland), (D) real Xilingol League area, (E) simulated Bayan Nur area (desert grassland), and (F) real Bayan Nur area	89
Figure 4-1. Radenski’s cloud-based 2D GoL algorithm.....	101
Figure 4-2. Marques’s 2D/3D GoL algorithm on MapReduce.....	102
Figure 4-3. workflow to accelerate voxel CA models	106
Figure 4-4. A 3D Von Neumann neighborhood scheme	107

Figure 4-5. Voxel GoL CA implementation on Giraph.....	108
Figure 4-6. Comparison results of GoL: default hash vs. customized partitioning ..	109
Figure 4-7. Giraph-based GoL with customized partitioning.....	111
Figure 4-8. Partitioning strategy of a voxel GoL CA	112
Figure 4-9. Traditional voxel air CA vs. Graph-based voxel air CA.....	119
Figure 4-10. Air pollutant particles dispersal CA on Giraph.....	121
Figure 4-11. Visualization of first timestep of single source air pollutant diffusion simulation.....	123
Figure 4-12. Single source air pollutant natural diffusion simulation for 50 timesteps	124
Figure 4-13. Multi-sources air pollutant natural diffusion simulation.....	125
Figure 4-14. One-billion-cells single source air pollutant dispersal simulation for 1000 timesteps	126

List of Abbreviations

AWS: Amazon Web Service

CA: Cellular Automata

GEE: Google Earth Engine

GoL: Game of Life

HDFS: Hadoop Distributed File System

HPC: High Performance Computing

IMAR: The Inner Mongolia Autonomous Region

LULC: Land Use Land Cover

MCE: Multi-Criteria Evaluation

Mlib: Machine Learning Library

RS: Remotely Sensed

SR: Sub Region

SVM: Support Vector Machine

Chapter 1 : Introduction

1. Background and Motivation

The arrival of the “Big data” era brings new opportunities and challenges to scientific research in various research areas (Lynch, 2008). By processing and analysing big data, researchers can add significant value to solving previously unverified research questions or gain a better understanding of problems in scientific research.

Geographical science, along with many other fields of study, is a data-driven science (Goodchild et al., 2012). Remotely sensed (RS) data is one of the main geospatial data sources that are accessible to support research studies in geographical science, as well as in earth science, environmental science and urban planning. RS imagery datasets support studies of sustainable ecosystems, such as land use land cover (LULC) ecosystem dynamics, sustainable urban ecosystems, and natural phenomena descriptions especially of large areas (Townshend, Justice, Li, Gurney, & McManus, 1991). Over the past few decades, space-borne and air-borne earth observation sensors have continually provided large volume datasets. For example, Landsat 8, the latest Landsat mission launched in 2013, can collect more than 700 images per day, corresponding to approximately 86 Terabytes of data per year (Y. Ma et al., 2015), 14 times as much data as collected in the 1980s (Wulder & Coops, 2014). And with the arrival of the Big Data era, where new data generated by processing and data mining, co-exists with the massive volumes of observed RS imagery data along with additional meteorological, environmental, hydrological and

even biological data, all will be stored and used as massive new data to support further scientific investigations (Yang, Yu, Hu, Jiang, & Li, 2017). With such large datasets, there comes many challenges in managing, storage, accessing and processing these datasets.

RS imagery datasets classification converts RS imagery dataset into valuable and meaningful information to further support geographical research (M. Li, Zang, Zhang, Li, & Wu, 2014). More specifically, different types of RS imagery datasets record LULC information for a large geographical area, even globally, with a certain temporal frequency. Mining LULC data from those imagery datasets provides essential information to many observation-based research studies, such as socio-economic studies, environmental applications, and urban planning (L. Ma et al., 2017). On the other hand, classified RS imagery data can be applied in modeling-based research such as complex geo-process simulation as inputs to drive simulation, calibrate models and validate results.

However, both observation-based research and modeling-based geographical research that need to process big data or consume big data will inevitably lead to a big computational burden (Y. Ma et al., 2015). For example, big volumes of RS imagery data need to be collected, pre-processed, and classified to support a large coverage high-resolution LULC dynamics study at a single time or within a time series. For modeling-based research, e.g., large scale (note that in this dissertation, the term “scale” refers to the study area extent. We do not refer to “scale” as it is used in cartography, but refer to e.g., a larger study area extent as being at larger scale), geo-complex systems simulation and prediction, in addition to processing the remotely

sensed data as part of its input, it also requires that the models themselves are capable of supporting big data processing and be capable of generating meaningful results in a reasonable time. For this part, theoretically, it would be ideal to utilize a universal computing model that can solve the major problems of complex systems at a massive scale (Heppenstall, Crooks, See, & Batty, 2011). Then, it would be desirable to optimize this big model for big data and to utilize this model to address big data challenges in many areas. This effort will contribute more than just the optimization of a single specific model that can only be used in a narrow area. Therefore, three major requirements emerge for big geographical research in big data era that include: 1) that remotely sensed datasets can be accessed in a computationally efficient way; 2) these datasets can be classified using existing or novel classifiers with high computational performance; and 3) a model that can consume big geographical input data and accommodate diverse geospatial complex systems simulation. Hence, how to address the challenges of implementing big geographical data processing and driving big models for large scale complex scenarios simulation within a reasonable processing time need to be carefully considered.

Traditionally, big geographical datasets can be processed on supercomputer and high performance computing (HPC) clusters with proper algorithms in high efficiency (Mineter, Dowers, & Gittings, 2000). However, most geographical researchers do not have access to a supercomputer and HPC computing resources, which makes it nearly impossible to do large scale geographical research that exceeded the computing capacity of single workstation hence blocks them to fully benefit from this big data era to further make significant progress on geographical scientific research. In this

dissertation, we introduce an alternative way – using cutting edge cloud computing technologies to support geographical researchers to handle big geographical data processing and large-scale models processing.

Cloud computing is a type of network computing framework that can be used to allocate and share the software and hardware resources on the internet based on user needs (Marques et al., 2013). By using this technology, the remote service providers can supply the same powerful performance network services as a "supercomputer" that can reach millions or even billions of tasks being computed in a few seconds. Many cloud-based geographical processing web-services have been published and hosted on different cloud platforms such as Google Earth Engine (GEE) (Gorelick et al., 2017) and Microsoft Machine Learning AI (Salvaris, Dean, & Tok, 2018). They allow normal user to access to their cloud-based resource and further perform cloud-based RS image processing with simple scripting. However, those web services could not fully solve the problems we mentioned before. Using GEE as an example, it is not a fully opened environment that allows normal users to acquire specific amount of computing resource; monitor how many resource is able to use at specific stage; inquire how many users before their submitted applications in the task queue, and find out when their applications will be finished or failed. Therefore, it is not realistic to use GEE to run very large scale (cannot fit ~ 600 Landsat 8 images classification in a single run according to our test) geographical related process for normal users.

In this dissertation, we applied open source cloud computing frameworks based on Hadoop distributed file system (HDFS) ecosystem to build customized cloud computing environments on both local and commercial cloud platform (Borthakur,

2007). Using these technologies, users can link their workstations by gathering all available bare metal resources together to support their research, which is not possible for HPC structure because HPC cluster does not support normal hardware. Users can also build a customized system with (in theory) unlimited computing capacities to implement the processing of data and models on commercial cloud computing platforms such as Amazon web services (AWS) (Amazon, 2015) and Microsoft Azure (Wilder, 2012), albeit limited by computing frameworks and budgets in real life. Another benefit for using those cloud platforms is users can easily access to different RS dataset from cloud data warehouse directly, for example AWS S3 (Calera, Campos, Osann, D'Urso, & Menenti, 2017), without downloading them to local machines, which could significantly reduce the data transfer cost especially for very large geographical research. Also, some pre-processed data have been deposited on those cloud platforms, e.g. Landsat 8 surface reflectance images, which might be useful for geographical research in some cases.

This dissertation makes a significant contribution to increasing our understanding about using cutting edge cloud computing technologies to support geographical researchers in fully exploiting the benefits from big data to better solve research questions in geographical science. In order to achieve this goal, this dissertation focuses on addressing three major challenges with big geographical research respectively:

- 1) How to design and implement a universal scalable solution with alterable core function for classifying multi-scale remote sensing datasets in multi-spatial, multi-spectral or multi-temporal cases with only minor adjustments, and capable of

exploiting benefits from cloud computing to access and process RS imagery datasets on local computing clusters and cloud platform in an effective and efficient manner based on performance assessment.

2) How to couple traditional Markov Chain - Cellular Automata (CA) models with a high-performance cloud computing framework in order to exploit accessible computing resources for LULC data gap filling using multivariable ground truth datasets as auxiliary and produce accurate results based on testing.

3) How to re-think and rebuild traditional CAs with respect to modeling, computational implementation, and computing resource access with cloud computing framework to accommodate massive scale voxel-based complex geographic systems in cloud environments.

2. Dissertation Structure

This dissertation investigates how to address the above three challenges for geographical research in the big data era using cloud computing technologies. Chapter 1 introduces the background, motivation, and the overall structure of this dissertation. Chapters 2, 3 and 4 each focus on one of the three major challenges. Those challenges are organized from two main perspectives, research dimension and research type (Figure 1-1).

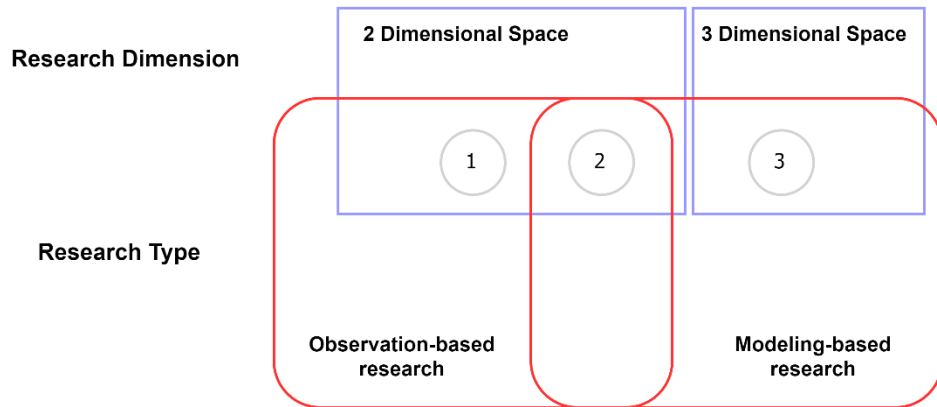


Figure 1-1. Conceptualization model of the three studies in this dissertation organized in two research perspective

From the perspective of research type, two directions are considered: 1) observation-based research; and 2) model-based studies. Assuming researchers can access massive volume, multi-source, high quality data sets, they will need a computing framework to mine those data and convert them into valuable and useful information. Chapter 2 focuses on research involving LULC classification to show how to process a LULC task with multi-source remote sensing imagery dataset in efficient and effective ways using a cloud computing framework. However, in applied studies especially for large-area studies, researchers are not always able to guarantee that all data are perfect quality. Missing information is quite common in large-area and long time series research (Shen et al., 2015). Chapter 3 focuses on migrating an existing and mature LULC data gap filling model—Markov Chain-CA—on the cloud to further enhance this model using cloud computing technologies and makes it a better fit for the big data era. This study is a mix of both observation-based research and modeling-based studies. Chapter 4 is purely modeling-based, where with this research we investigate how to implement a universal model to process a massive scale complex geo-simulation using a voxel CA on the cloud.

From the perspective of different types of spatial framework, these three topics can be treated in 2-dimensional (2D) and 3-dimensional (3D) space. The processing in both Chapter 2 and Chapter 3 uses a 2D spatial perspective because for pixel-based remotely sensed imagery data processing and LULC related studies, all information is stored as a flat surface. However, to simulate complex geographical phenomena requires models to be built in 3D for most cases to better describe and analyze the details of the system in each dimension. By extending the computational framework developed in Chapter 3, Chapter 4 moves to a 3D representation where a voxel-based CA model will be applied to implement a massive voxel GoL CA and to accommodate an air pollutant particle dispersal simulation in 3D with synthetic data on cloud, which are used for testing the computational performance of our cloud-based voxel CA on both standard performance testbed in literature and actual geographical application.

2.1 Chapter 2: A Cloud-based Computing Framework to Unfold Processing Efficiencies for Pixel-Wise Remotely Sensed Imagery Data Classification

Chapter 2 focuses on investigating a scalable geographical data processing framework by using cloud computing to support RS imagery datasets accessing and processing in an effective and efficient manner as determined by performance assessment on multi-sourced RS imagery datasets classification in different scales on different computing environments. The motivation of this study aims to seek an alternative approach to HPC solutions to handle large scale geographical research in the big data era. In this chapter, RS imagery classification is selected as an example task to demonstrate the flexibility, extensibility and accessibility of an open source

scalable framework that has been developed by exploiting Apache Spark (Sun, Chen, Chi, & Zhu, 2015), to implement parallel, in-memory image processing with an ability to rapidly classify multi-spatial, multi-spectral or multi-temporal RS images for either a single time point or for a time series. This framework is deployed on a local cloud environment and commercial cloud platform with direct AWS S3 data access to test the feasibility of the approach. Two study areas: Suez Canal, Egypt and Inner Mongolia, China are chosen with two RS imagery datasets, Landsat 8 OLI and MODIS, at different scales. These datasets are used as test environments to verify the practicability and performance of our framework using two different classification approaches, NDVI/NDWI threshold classification method and a Support Vector Machine (SVM) classifier in distributed mode on the cloud. Tests using this framework demonstrate that the framework can be used on multi-sourced remotely sensed imagery datasets with alterable classifiers and different cloud platforms, to implement the classification tasks using only minor parameter adjustments rather than having to fully rebuild a brand new tool. This work demonstrates that this framework could be a possible solution to support RS imagery dataset processing in big data era.

2.2 Chapter 3: Data Gap Filling for Land Use and Land Cover Study using Cloud-based Distributed Markov Chain Cellular Automata

An inevitable issue during the application of the framework presented in Chapter 2 is that to classify RS imagery datasets especially for big study area, it is not unusual to be faced with missing data due to poor weather conditions or possible sensor malfunctions during data collection. This issue appears more obvious once researchers are trying to collect remotely sensed data to support large coverage study

areas or studies with long time series. A possible solution to overcome this issue is to apply a gap filling algorithm to make up those missing data gaps. However, another challenge lies with gap filling the classified datasets generated by the framework in Chapter 2, is that the algorithm cannot be processed using a single workstation without supporting a large-scale processing framework. Therefore, the motivation of Chapter 3 is to solve this issue in LULC related studies by offering researchers an open source cloud-based gap filling framework that: 1) integrates seamlessly with existing cloud-based RS imagery processing framework and 2) is capable of handling LULC data gap filling and provide a complete LULC map in selected study areas.

Follow this idea, we applied cloud-based and open source distributed frameworks, Apache Spark and Apache Giraph to build an infrastructure to fill gaps within a LULC dataset that supports big geographical research. This infrastructure was built with a Markov Chain - Cellular Automata model that fully exploited the scalability and high performance of cloud computing. This study further explores the feasibility, accuracy performance of this framework by using whole Inner Mongolia, China as a study area for gap filling test. Ground truth data that included biophysical data, ground feature information and socioeconomic data is collected from 2000 to 2016 to support this study. An accuracy assessment is performed to prove the feasibility of this framework.

2.3 Chapter 4: Massive Voxel Cellular Automata Using Giraph: Application to Air Pollutant Particles Dispersal

The results of the research in Chapters 2 and 3 can provide a completed LULC map to support observation-based geographical research, and they can also be applied

as input datasets to drive model-based geographical studies. However, the computational framework in Chapter 2 and Chapter 3 cannot directly accelerate geographical models except by providing big input datasets. The cloud-based CA model developed in Chapter 3, is developed using a 2D spatial perspective, while actual complex geographical systems will often require 3D representation in space to fully analyze details in all dimensions. In Chapter 4, the 2D cloud-based CA is extended to make this CA model fully capable of handling massive scale geospatial complex systems modeling and simulation. In this chapter, we reconstruct the traditional CA model from the perspective of modeling, computational schema and computing resource accessing to migrate a traditional 2D CA into 3D space. We present an approach using Giraph, an open source HDFS-based large scale graph processing framework, to implement massive scale voxel CA models. We demonstrate the scheme on two test applications. First, on a 1 trillion cell 3D Game of Life (GoL) (Conway, 1970) CA model testbed, where processed on AWS cloud platform, we achieved an average processing time of about eight minutes per time step for this test. In comparison with existing 1 trillion scale approach in literature, the computing efficiency is about 90 times faster than theirs (Marques et al., 2013). In addition to the GoL test model, we further explored a real geographical model – air pollutant particle dispersal simulation on the cloud. By integrating an existing air pollutant particles dispersal physical model (Jjumba & Dragicevic, 2015) with our computational voxel CA framework, we successfully simulate the air pollutant particles advection and diffusion with 1 billion cell scale at local cloud environment. After performance assessment, we found our solution could process 1 billion cells

simulation for 47.3 seconds per step while the original MATLAB based series programming approach took at least 24 hours.

Chapter 2 : A Cloud-based Computing Framework to Unfold Processing Efficiencies for Pixel-wise Remotely Sensed Imagery Data Classification¹

1. Introduction

1.1 The Rapid Expansion of Remotely Sensed Data in Big Data Era

Data provided by remote sensing have long presented as a critical resource in monitoring, measuring, and explaining natural and physical phenomena. Indeed, remote sensing might justly be characterized as one of the first “big data” sciences (Goodchild et al., 2012). Steadfastly, advances in the sensing capabilities of remote, Earth-observing platforms have continued to produce more and more data, with increasing observational breadth and finesse of detail. These developments carry a dual benefit and problem: analysis and inquiry in the environmental and Earth sciences are routinely awash with data, but also often struggle to match pace in building empirical knowledge from those data because the data are incoming with such haste and heft. Strategies to manage big remotely-sensed data are required to fully exploit the benefits those data hold for applied scientific inquiry, and the topic of how computing might be leveraged to ease pathways between science and sensing holds significant currency across many fields, with particularly rapid adoption of

¹ An earlier version of the research described in this chapter has been published in *Journal of Sensors* as Lan, H., Zheng, X., & Torrens, P. M. (2018). Spark Sensing: A Cloud Computing Framework to Unfold Processing Efficiencies for Large and Multiscale Remotely Sensed Data, with Examples on Landsat 8 and MODIS Data. *Journal of Sensors*. (Lan, Zheng, & Torrens, 2018)

high-performance computing (Yang, Yu, Hu, Jiang, & Li, 2017) and cloud computing in the geographical sciences (Lee, Gasster, Plaza, Chang, & Huang, 2011). Remote sensing imagery is a commonly used source to support those studies of sustainable ecosystems, such as ecosystem dynamics, grassland degradation, and urban ecosystem restoration, especially in large areas (Wulder & Coops, 2014).

Traditionally, studies with pure remotely sensed data involved only a few scenes of data in a limited study area, or they rely on low-resolution remotely-sensed images in large-area experiments (Z. Wang, Zhong, Lan, Wang, & Sha, 2017). Those traditions are changing as new data have dramatically altered the underlying substrate for analysis. For example, in the past few decades, the space-borne and air-borne earth observation sensors are continually providing large volume data sets. For example, Landsat 8, the latest Landsat mission launched in 2013, can collect more than 700 images per day, corresponding to approximately 86 Terabytes of data per year (Y. Ma et al., 2015), which is 14 times as much as that in the 1980s (Wulder & Coops, 2014). Processing the massive volume of remotely sensed data is now not the only problem: the intrinsic complexity of those data is also an important issue that must be considered.

The sensors that are actuated in remote sensing are usually designed to serve specific requirements of analysis for different fields of study. To fulfil those different needs, sensors are usually tasked to capture images at different resolutions. For example, high-resolution satellite sensors such as *WorldView-4* can produce imagery with a spatial resolution of 0.31-m in panchromatic vistas, and 1.24-m spatial

resolution in multispectral vistas (Satellite Imaging Corporation, 2017b), while the *QuickBird* platform can image the Earth with 0.61-m spatial resolution in panchromatic form and 2.44-m spatial resolution in multispectral form (Satellite Imaging Corporation, 2017a). These resolutions, on the order of fractions of a meter to a few meters in spatial resolution, presented significant opportunities to monitor the Earth, and represent the state of the art in Earth-observing imaging detail. Concurrently, other remote sensing platforms are tasked with refreshing observations of the whole Earth's surface, aiming for coverage of large areas with temporal consistency, rather than small-area detail. For example, relatively medium-resolution sensors, such as *Landsat*, and relatively low-resolution sensors, such as *MODIS*, are deployed as long-term Earth observatories. *Landsat* provides 15-m panchromatic and 30-m multispectral imagery, which is very widely used in studies of large-area grassland degradation and urban land cover dynamics (Lan & Xie, 2013). *MODIS* offers 250-m multispectral imagery and can build a mosaic view of the entire Earth once every few days. *MODIS* data has been widely adopted in global-scale research studies, particularly those trained on studying vegetation canopies for investigation of world-wide forest cover dynamics (Friedl et al., 2002).

Many sensors support multispectral imaging. For example, *WorldView-4* data includes four spectral bands, and *Landsat 8 OLI/TIRS* provides 11 spectral bands. For some spectrally sensitive studies, higher spectral resolution imagery is required. In those studies, hyperspectral sensors (such as *Hyperion*, which generates 220 bands between 0.4–2.5 μm (U.S. Geological Survey, 2011)) can produce detailed spectral data over a very small wavelength range. Furthermore, different sensors offer

different temporal resolution in their rate of imaging as well as the timing of their coverage of subjects under their purview. For example, the *WorldView-4* satellite is capable of revisiting views every 4.5 days (sometimes sooner), while Landsat can deliver repeats views every 16 days (Turner et al., 2003). Higher temporal resolution in return views facilitate study of dynamics on the Earth surface, so that the time series and the time interval between visits become significant attributes of the observation, alongside the spatial resolution and spectral range. Therefore, methods to streamline a feasible, effective, and efficient approach to processing archived and continually-incoming multi-spatial, multi-spectral, and multi-temporal remote sensing data are an ongoing requirement across many potential applications of remote sensing to applied scientific inquiry.

In this research, possible scalable solutions were introduced to address issues of processing multi-scale remote sensing datasets in multi-spatial, multi-spectral or multi-temporal cases. The aim was to implement a tool that can process different proposed remote sensed tasks with only minor adjustments rather than fully rebuild new toolkits. Furthermore, this solution should be fully capable of exploiting benefits from cutting-edge cloud computing technologies, resources and platforms to help researchers process and analyze remotely sensed datasets that was difficult to process on local machines in an effective and efficient manner.

1.2 Computing as A Resource for Big Data Processing

Many researchers have made significant progress in advancing feasible, effective, and efficient processing for multi-prong attributes of remotely sensed data, using developments in computer engineering. In particular, research into how

graphical processing units (GPUs) and cluster-based high-performance computing (HPC) might be leveraged to advance image processing for remote sensing has been particularly fruitful (Q. Huang et al., 2013). More recently, cloud computing is increasingly being considered as a resource in processing remotely sensed imagery, largely because of cloud computing's native abilities to scale computing in kind as the data being processed also scale. Furthermore, significant cloud computing resources are now available commercially, on a "pay as you go" model, from providers such as *Amazon Web Services (AWS)* (Amazon, 2015), *Microsoft Azure* (Wilder, 2012), and *Google's Compute Engine* (Sanderson, 2009). These resources can be brought to bear on image processing tasks as IaaS (Infrastructure as a Service), PaaS (Platform as a Service) or SaaS (Software as a Service).

Cloud computing is useful in providing some of the flexibility required to match pace between incoming data, large existing data silos, and evolving analytical needs in image processing that authors alluded to in the introduction. Cloud computing affords this flexibility by allowing users to allocate and share software and hardware resources on the Internet in a distributed fashion, by splitting large computational tasks into many small parallel computing tasks, then assigning them to as many computing instances as are required to achieve computing goals based on data size, data fusion, resource use, or computing time. After all the distributed nodes of the cloud service have completed their assigned tasks, the results are bundled and returned to the users' local database. In this way, virtual instances, applications, and software are provided on an as-requested basis, and users may pay for those services as demanded. This affords a user access to a theoretically limitless size computing

capacity (although very strong limits of available financial budgets to pay for the services quickly dock theoretical capacities to tangible practical realities in many real instances).

A promising community of computing frameworks have co-developed alongside cloud computing hardware, and several of these frameworks hold significant promise for processing remotely sensed imagery of the Earth's surface. For example, MapReduce was introduced by Dean and Ghemawat (Dean & Ghemawat, 2008), ten years ago. In the decade since, a number of open source implementations of the MapReduce model have emerged as promising frameworks for mediating the computing between image processing for remotely sensed data and cloud resources that are available to distribute and/or accelerate that computing on commercial (or user-run) clouds. Chief among these open source implementations of MapReduce is *Apache Hadoop*. While *Hadoop* MapReduce relies on reading and writing data to a disk, another variant, *Apache Spark* (Apache Software Foundation, 2018a) maintains data partitions in memory (a so-called in-memory computing framework). *Spark* also provides a network buffer for each reducing task, rather than merging outputs into a single partition, with the result that *Spark* can be one hundred times faster than *Hadoop* MapReduce on some big data tasks (Guller, 2015). Nevertheless, one advantage that *Hadoop* might hold over *Spark* is that *Hadoop* allows parallel processing of large amounts of data that are bigger in physical storage size than the available memory. In fact, many remote sensing datasets are of a size that is so massive that they exceed the memory available in local machines or small clusters. Furthermore, physical disk resources are usually much less expensive in financial cost

(of owning or accessing) than memory resources are. So, in cases for which limited memory may become a constraining factor, *Hadoop* presents as a better option in some cases for processing large amounts of remote sensing data.

However, cloud computing frameworks are agile relative to resource constraints. And that novel advances in cloud computing technologies and cloud platforms allow *Spark* to leverage resources from and across different cloud computing platforms, with the possibility that memory limitations may no longer loom as a constraint for big remote sensing data processing scenarios. Consider, if memory as a resource that can be drawn upon on an as-needed basis, researchers can access theoretically unlimited memory on the cloud. Furthermore, *Spark* can run on a single workstation, as well as local computing clusters and cloud platforms. And, *Spark* could access diverse data sources, such as *Amazon S3*, *Hadoop Distributed File System (HDFS)*, *Cassandra*, and *HBase*. In other words, *Spark* can access not only local private data warehouses, but can also reach cloud-stored remote sensing datasets, and do so via the cloud platform directly, with the ability to process those data on the cloud, and then stream back the required results.

1.3 Cloud Computing for Processing Remotely Sensed Images

One of the common computing solutions to the burden of processing, analyzing, and managing large-scale remote sensing data is to parallelize the remote sensing processing tasks: to spread the burden over multiple computing units to reduce the overall processing time (Y. Ma et al., 2015). For example, Huang and her colleagues used the Message Passing Interface (MPI) as a computing framework for their work on dust storm simulation and forecasting on the *Amazon EC2* commercial cloud

service (Q. Huang et al., 2013). They deployed MPI on the Amazon cloud and applied loosely coupled nested models to process a high-resolution dust storm dataset. Their performance tests showed efficient and economical results. Cavallaro *et al.* (Cavallaro, Riedel, Bodenstein, et al., 2015) used GPUs to implement a support vector machine (SVM) classifier with MPI and *openMPI* frameworks. As an alternative to using HPC computing frameworks, other researchers have developed their own bespoke parallel large-scale remote sensing data processing platforms. For example, Wang *et al.* (L. Wang, Ma, Yan, Chang, & Zomaya, 2018) developed *pipsCloud*, which is a cloud-based approach to process remote sensing on-demand and in real-time. To further enhance performance, Wang *et al.* (L. Wang et al., 2018) used Hilbert-R⁺ tree indexing.

The turn toward development of tools by remote sensing scientists, leveraging computing techniques but departing in ways that are special to remote sensing applications is a wonderful development for remote sensing science. Nevertheless, bespoke solutions (particularly in academic settings) cannot feasibly contribute the massive levels of computing available now commercially, with the result that absolute performance will always lag behind that which might otherwise be available on the marketplace. Also, applying MPI or self-developed systems often requires significant research and development effort into programming, debugging, and tuning the computing system and environment, and one might perhaps make an argument that the time devoted to these tasks could be used on the applied science instead. In some cases, building these systems on a bespoke basis is very challenging. For example, consider MPI, programming tasks designed for serial computing and

converting them to parallel form can be significantly burdensome, and particularly so for some complex image processing algorithms. Moreover, the networking security, near ubiquitous availability, and fast-moving hardware compatibility (for example in shared memory clusters) of commercial platforms offer significant practical advantages. Some existing work points to the potential advantages that are obtainable in cloud processing of big geospatial data. For example, Chen and Zhou (Chen & Zhou, 2015) demonstrated that *Apache Hadoop* can be leveraged for partitioning using a Mean Shift algorithm. With a local mode test, they successfully increased the processing speed by ~2 times (Chen & Zhou, 2015). Also, Giachetta (Giachetta, 2015) introduced a *Hadoop*-based geospatial data management and processing toolkit, *AEGIS*, which was compared against existing MapReduce-based frameworks, such as *SpatialHadoop*, *Hadoop-GIS*, *HIFI* and *MrGeo* with spatial join, query, and aggregation operations (Giachetta, 2015).

As mentioned in the introduction, compared to MapReduce-based approaches, solutions based on *Apache Spark* have been found to generate results at higher efficiency. For example, Shangguan *et al.* applied K-means clustering with Spark on an OpenStack-based cloud computing platform to process a 7-band Landsat TM data with 5244 rows and 5205 columns. They concluded that when K is a small value such as 10, the paralleled K-means model does not show better performance than normal, but at K=80, I=20, the algorithm in parallel ran 1600 sec faster, at 712 seconds (Shangguan & Yue, 2018). Absardi *et al.* used Spark to compare the performance and accuracy of processing big RS imagery dataset by using multi-layer perceptron algorithms (MLPC) and linear Support Vector Machine (SVM). They adopted

Landsat 7 ETM+ images from 6 Iranian cities and achieved 88% average accuracy with SVM while 92% with MLPC. Additionally, they found that the MLPC took longer with HDFS, compared to a Spark standalone mode, which averaged 105 seconds (Absardi & Javidan, 2017). Lunga *et al.* demonstrated a novel RS data processing flow, RESFlow, which integrated Spark and deep learning on a NVIDIA DGX platform to process RS imagery dataset. They tested the model on many different study areas including Ethiopia, South Sudan, Yemen, Zambia, Alabama, Arizona, Puerto Rico, and New Mexico using Digital Globe constellations. By using Spark, they processed 21,028 TB of imagery data and output maps at area rate of 5.245 sq.km/s, a total of 453,168 sq.km/day, which reduced a 28-day workload to 21 hours (Lunga, Gerrand, Yang, Layton, & Stewart, 2020). Wang *et al.* developed a new image segmentation method using Scala with Spark and GeoTrellis framework on IntelliJIDEA. They tested it with 8 Sentinel-2 images, which covered rural areas, urban regions, and suburban zones. They decomposed those images into sub-images and ran their algorithms in parallel with different computers to get the final reduced result, which shown a performance lifting according to their experiments (N. Wang, Chen, Yu, & Qin, 2020). Sun *et al.* (Sun et al., 2015) used *MLlib* in *Spark* to test the multi-iteration singular value decomposition (SVD) algorithm on high-resolution hyperspectral remote sensing images. Compared with the *Apache Mahout* (MapReduce) approach, they found that the *Spark* approach can essentially trounce MapReduce in their tests, once *Spark* is able to access enough hardware resources. Another study using *Spark* to process remote sensing data, by Huang and his colleagues (W. Huang, Meng, Zhang, & Zhang, 2017) demonstrated a series of

comprehensive performance tests, using *Spark* to implement different types of algorithms in remote sensing. Huang *et al.* (W. Huang et al., 2017) discussed the performance of each of the tests on different running environments, including local, standalone and *Yet Another Resource Negotiator* (YARN). They also proposed a self-defined, strip-based partitioning approach to replace the default hash partitioning method (W. Huang et al., 2017). In this research, we will use *Spark* as a medium to develop a large scale RS imagery dataset processing framework that be able to handle tasks of classifying multi-source RS images with alterable classifiers as well as be capable to be deployed on different types of cloud platforms with only minor parameters adjustment.

2. Study Area and Dataset

To prove the applied utility of the proposed scheme, four experiments were performed, and two different remote sensed imagery datasets were involved. The first experiment used the Landsat 8 Operational Land Imager (OLI) dataset at three different scales to test a workflow of classification and visualization algorithms with *Spark* on a local cloud environment. To further study the tuning performance of this scheme, a second experiment was designed to run this tool under different execution configurations with the same *Landsat 8* dataset. The third experiment was to demonstrate that the classifier in this framework was alterable, and that this framework was capable of handling actual classification tasks in a geographical study. In this experiment, we used a cloud-based distributed Support Vector Machine (SVM), one of the most commonly used classification methods to replace the NDVI/NDWI threshold classifier in our framework and tested it on data for Inner

Mongolia, China. The last experiment was designed to demonstrate that this tool can consume remote sensed imagery datasets from different sources and can be deployed on different cloud environments using only minor parameter adjustments. Using the NDVI/NDWI threshold classification method presented above, a MODIS dataset on *Amazon EC2* a commercial cloud platform, was processed.

Landsat 8 scans the entire planet surface in a 16-day period (U.S. Department of the Interior & U.S. Geological Survey, 2018). Equipped with the latest OLI sensor, *Landsat 8* provides unprecedented spectral information with two additional spectral bands in the whole Landsat instruments family. In addition to offering a 15-m panchromatic band and a 30-m multispectral band, as in many previous products, *Landsat 8* also includes a Quality Assessment (QA) band to support pixel-based cloud, shadow, and terrain occlusion filtering. A relatively short revisit period, medium spatial resolution, and seven spectral bands make *Landsat 8* OLI products commonly used and freely accessible remote sensing imagery datasets for science that rely upon spatial, spectral, and temporal Earth attributes for environmental research. Furthermore, the *Landsat 8* dataset is currently available for public use on cloud platforms, such as *Amazon S3* (Amazon Web Services Inc, 2018a) and *Google Earth Engine* (GEE), making it a great data source to serve remote sensing datasets processing in the cloud.

Like *Landsat 8* OLI, *MODIS* satellite datasets are available on *Amazon S3* (Amazon Web Services Inc, 2018b) and *GEE* since 2017. *MODIS* provides a variety of planet observation products with daily temporal resolution. In this study, MODIS/Terra Surface Reflectance Daily L2G Global 1 km and 500 m SIN Grid

V006 (MOD09GA) will be used as secondary data input, which can provide 7 bands of surface spectral reflectance with 500-meter spatial resolution. In addition, 1-kilometer resolution observation and geolocation statistics bands are also offered in this product.

The coverage of each of the datasets for the Suez Canal (SC) area is illustrated in Figure 2-1 (Figure 2-1). Three scales of *Landsat 8* imagery datasets at small-, medium-, and large- scale were used as input data sources in the experiments. This multi-scale approach allowed us to assess whether our tool can perform multi-scale remote sensing image processing with only minor parameter adjustments, and to assess whether a wide array of remote sensing imagery datasets can be processed by our proposed framework in real scenarios. Because remote sensing images, which are collected by different sensors at different spatial resolutions, with various bands of spectral information, and at diverse temporal scales, are essentially formed by pixels, pixel-based algorithms implemented via *Spark* in the cloud should be capable of performing a very wide array of pixel-based processing and analysis. However, to further demonstrate that the tool can handle multi-source remote sensing images, a *MODIS* dataset that fitted the same coverage as the large-size *Landsat 8* dataset was

used to test the framework.

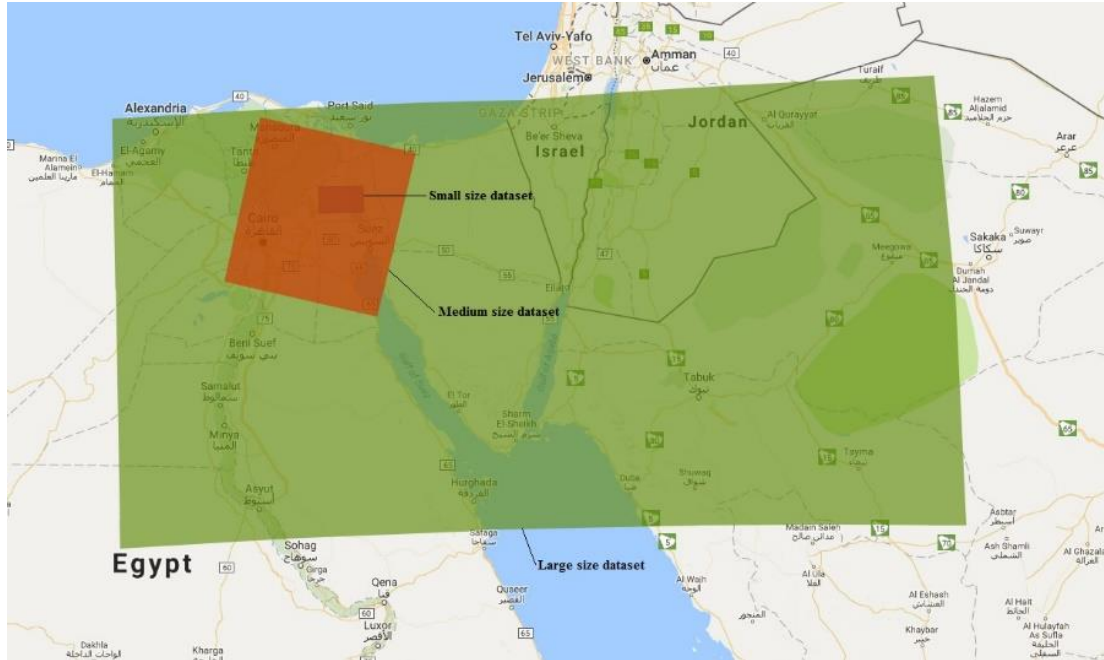


Figure 2-1. Study area at SC area with remotely sensed imagery dataset coverage at three different scales

To verify the capability of this framework and to classify the remotely sensed images in a time series, one image per year was chosen from a imagery dataset of the Suez Canal (SC) spanning 2013 to 2017 to generate a time series. Small-size and medium-size images were approximately matched to one single scene of a *Landsat* image. Hence, images that were acquired on a similar date for each year were selected to reduce the land use land cover (LULC) changes caused by seasonal variation factors as is common in time series related research. However, the large dataset was formed by using over 15 scenes of original images. In this case, it was not possible to guarantee that each tile of this dataset could be filled by images with the same date for each year. Hence, by broadening the filter criteria to month, it was possible to ensure we could obtain enough tiles to form the whole area. Another important factor

was that the chosen images were preferably of high quality to reduce any cloud and haze problems (i.e., gaps). For the *MODIS* dataset, with relatively coarse spatial resolution and large coverage per scene, it took about half of a single scene to fit the large-size Landsat dataset coverage. Also, the acquisition time matched the dates of the small- and medium-size Landsat dataset because *MODIS* provides daily products. The detailed acquisition time of each dataset is listed in Table 2-1.

Table 2-1: Detailed acquisition time of each dataset.

	2013	2014	2015	2016	2017
SC_Small	4/29	3/31	4/19	3/4	4/5
SC_Medium	4/29	3/31	4/19	3/4	4/5
SC_Large	Apr	Mar	Apr	Apr	Apr
MODIS	4/29	3/31	4/19	3/4	4/5
IMAR	N/A	N/A	N/A	Aug	N/A

The reason for the choice of the study area shown in Figure 2-1 was that in the SC area, the LULC mainly included sparse and dense vegetation, a natural water body, the SC (with water), and bare sands and rocks. Thus, the study area was an excellent test area for our NDVI/NDWI threshold classification method to detect detailed LULC and generate accurate results. Another reason for the choice of the study area was that the SC has been expanding since 2013 and another branch has been constructed (Suez Canal Authority, 2013). This expansion can be clearly monitored as demonstrated by the classification results.

A second study area located in Inner Mongolia (IMAR), China was also used for this research (Figure 2-2). As the SVM classifier was a supervised classification approach that required an available ground truth dataset as a training dataset, the IMAR region was chosen due to the ground truth data availability in this area. A high accuracy LULC vector mapping dataset that was generated based on visual

interpretation assisted by sample plots was available for 2000, 2010 and 2016 for this region. The training dataset was extracted using the 2016 LULC map. Landsat 8 images were acquired for this area on August 2016 with approximately 600 scenes.



Figure 2-2. Study area at IMAR

In these experiments, all remote sensing image datasets were exported from *GEE*, a cloud-based platform that serves remote sensing data sources with customized criteria (Gorelick et al., 2017). For example, setting (1) the region boundaries to acquire data belonging to our study area, and (2) the cloud mask to filter cloud pixels on images before exporting the data to our cloud computing drive. *Landsat 8* and *MODIS* (MOD09GA) both provided surface reflectance produced on *GEE*. If other datasets are used, a strictly image-based atmospheric correction should be followed to remove any haze impact on those images (Chavez, 1996).

3. Method

In this research, we extended recent developments in cloud computing as a resource for processing remotely sensed imagery. Specifically, a *Spark*-based remote sensing image processing tool capable of classifying multi-source remote sensed imagery datasets with alterable classifiers was deployed on a cloud platform. We applied Normalized Difference Vegetation Index (NDVI)/Normalized Difference Water Index (NDWI) threshold classification method as a testbed to assess the feasibility and performance of the developed cloud computing tool. Also, a robust SVM classifier was tested with this framework to verify the capability of this framework using an additional, popular method for classification. One of the key advantages of this approach was its ability to easily and straightforwardly support users' access to different remotely sensed imagery datasets that archived on cloud data warehouse and process LULC classification on them download those data to local machines before the classification results were generated. Moreover, a visualization approach to save text-based output from the analysis in common picture format, allowed users to examine results easily and quickly.

3.1 Cloud-based Distributed NDVI/NDWI Threshold Classification

The approach presented in this study was based on the YARN cloud environment. The goal, in using YARN, was to facilitate the availability of the processing environment in ways that were widely applicable to real-world scenarios. YARN has been widely used in many current cloud environments (X. Fan, Lang, Zhou, & Zang, 2017). Unlike traditional versions of *Hadoop* MapReduce, YARN allows for allocation of system resources as containers to the various applications. In

other words, different computing frameworks can be deployed on a single physical cluster in a noninterfering manner. In YARN mode, a *Spark* framework is composed of a master instance and many workers. The master instance is responsible for negotiating with the YARN *Resource Manager* to request enough computing resources, as needed, by analyzing the *Spark* applications submitted by clients. Once the master instance is loaded, it will schedule the tasks to executors with allocated containers. Until all tasks have been finished, *Resource Manager* will revoke all allocated resources for further possible tasks.

Resilient Distributed Datasets (RDD) form the basic abstraction of a dataset in *Spark*. RDDs can be created from an external dataset, or from existing RDDs. In this research, for cases using an NDVI/NDWI threshold classification method, RDDs were created from the original remote sensing images stored in HDFS. For each image, three RDDs were created for green, red, and near-infrared bands for the NDVI and NDWI LULC classification at the initial stage. RDDs contain the data partitions and the metadata records. According to the *Spark* mechanism, RDDs go through a series of transformations and action operations to process the data partitions in a distributive manner. Transformation operations of RDDs were executed on individual partitions of an RDD and those operations returned a new RDD. Action operations summarize information from an RDD by user-defined functions and return a result. For example, the join operation, as a commonly used transformation operation in *Spark*, involves joining partitions belonging to two RDDs and creating a new one. In this case, the data blocks will not be moved at the current stage because of the *lazy*

mechanism. However, action operations, such as count, will process the data partitions in RDD and perform real computing.

Based on the features of *Spark*, a workflow was designed (Figure 2-3). The first step in that workflow was to extract each band of a raster dataset by using the *Geospatial Data Abstraction Library* (GDAL), which was an open-source translator library for raster and vector geospatial data formats (Qin, Zhan, & Zhu, 2014). Each line of the extracted data contained the geographical coordinates and the Digital Number (DN) value of the raster cell in the original raster dataset. These files were then put into the HDFS as data input sources for distributive processing.

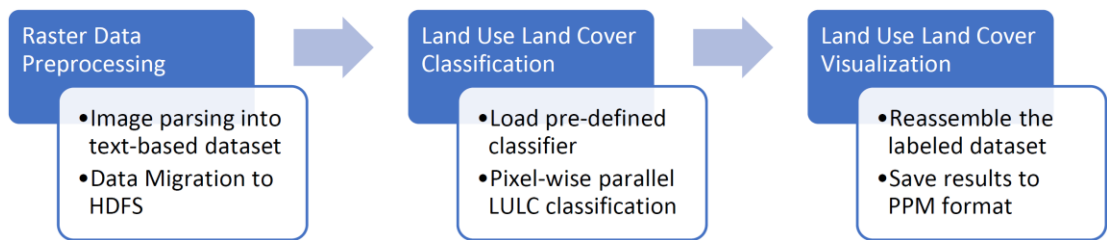


Figure 2-3. *Spark*-based remote sensing image processing workflow

The second step was to perform parallel processing and image(s) classification with the pre-defined classifier(s). As introduced earlier, for each image that must be processed by this tool, three RDDs will be created for green, red, and near-infrared bands. The basic units of the RDD were key-value pairs. In this case, the geographical coordinates were set as the key, and the DN value of a raster cell was the value. In this manner, parallel LULC classifications could be conducted across many computer units in a cloud computing cluster. The extent of parallel processing depended on the *Spark* application configuration and the cluster hardware specifications. Because the basic parallel processing unit was pixel-wise, this method was highly flexible and scalable, allowing raster datasets to be partitioned in any

manner, regardless of the spatial structures of the raster. However, partitioning strategies still must be considered seriously, as an appropriate partitioning strategy will help to optimize the performance with better usage of the computing resources of the cluster. *Spark* supports hash partitioning and range partitioning by default; these applications were appropriate for many case studies. However, according to the results of research by Huang *et al.* (W. Huang et al., 2017), the partition scale should not be too small or too large. A very small partition scale will result in low-performance computing and even increase the fault recovery cost. A very large partition scale may lead to an out-of-memory error. Inspired by their strip-based partitioning method, splitting data into chunks with a configured HDFS block size was needed in the following experiments.

Algorithm 1 NDVI/NDWI threshold classification

```
1: Input = TIF1, TIF2, ..., TIFK; Dimension = X * Y
2: For k = 1 to K-1 do:
3:   InputTIF1 = TIFk
4:   InputTIF2 = TIFk+1
5:   For m = 1 to 2 do:
6:     For x in 1 to X:
7:       For y in 1 to Y:
8:         ndvixy = (InputTIFxy.NIR – InputTIFxy.R)/(InputTIFxy.NIR + InputTIFxy.R)
9:         ndwixy = (InputTIFxy.G – InputTIFxy.NIR)/(InputTIFxy.G + InputTIFxy.NIR)
10:        lulcxy = CLASSIFIER(ndvixy, ndwixy)
11:        lulcm[x,y] = lulcxy
12:      endFor
13:    endFor
14:  endFor
15:  Output = lulck
16: endFor
```

Figure 2-4. Algorithm for NDVI/NDWI threshold classification

The details of an algorithm to use NDVI/NDWI threshold as a classification method on remote sensing images is shown in Figure 2-4. Several transformation operations were performed on each partition in RDDs. For k input *GeoTIFF* remote sensing images, the algorithm first parsed the dimension of them. Moreover, the RDDs were marked by a time sequence in the time series. For each data chunk, in each image RDD the NDVI/NDWI threshold classifier was applied to each pixel to calculate the classification indication values. The NDVI (Rouse Jr, Haas, Schell, & Deering, 1974) can be calculated as:

$$NDVI = \frac{NIR - RED}{NIR + RED}$$

The range of NDVI is from negative one to one, with different ranges to denote sparse vegetation, dense vegetation, barren rock and sand, and water. However, NDVI may not always correctly distinguish water bodies, especially when there is a noisy signal in the water area (mud for example). To further improve the accuracy of classification, NDWI was applied according to McFeeters (McFeeters, 1996) as follows:

$$NDWI = \frac{GREEN - NIR}{GREEN + NIR}$$

NDWI was able to distinguish water features with the positive values indicator. From the perspective of algorithm implementation, RDDs of three bands were joined as a single RDD for NDVI/NDWI calculation of each raster cell with mapping operations. LULC classification can, therefore, be conducted in parallel using the calculated NDVI/ NDWI results. Once the LULC features were classified by NDVI/NDWI indicators, each feature was labelled with a class ID.

Algorithm 2 Visualization

```
1: Input = lulc1, lulc2, ..., lulcK; Dimension = X * Y
2: For k = 1 to K do:
3:   inputKVk = lulck
4:   rgbKVk = RGB(inputKVk)
5:   Sort rgbKVk by two dimensions
6:   rgbVk = values of rgbKVk
7:   Reshape rgbVk to dimensions X * Y
8:   Output = rgbVk
9:   Save rgbVk to files
10: endFor
```

Figure 2-5. Algorithm for result visualization

The final step was to visualize the LULC classification results, as illustrated in pseudo code (Figure 2-5). With all k numbers of LULC results generated by Algorithm 1 for each image pair, visualization images were created in PPM format from the output RDD, which was a lowest common denominator color image file format (Frery & Perciano, 2013). Although redundant, PPM was an easy format to write and manage text-based outputs into human-readable figures. The RDDs of LULC classifications from Algorithm 1 were scanned and values of key-value pairs were converted to colors according to a user-defined RGB color scheme. Those RDDs were then sorted to the original order and reduced in a manner such that each key-value pair represented a row of the original raster image. Finally, to produce a PPM-format image, those RDDs were appended to the PPM file header to create complete PPM images.

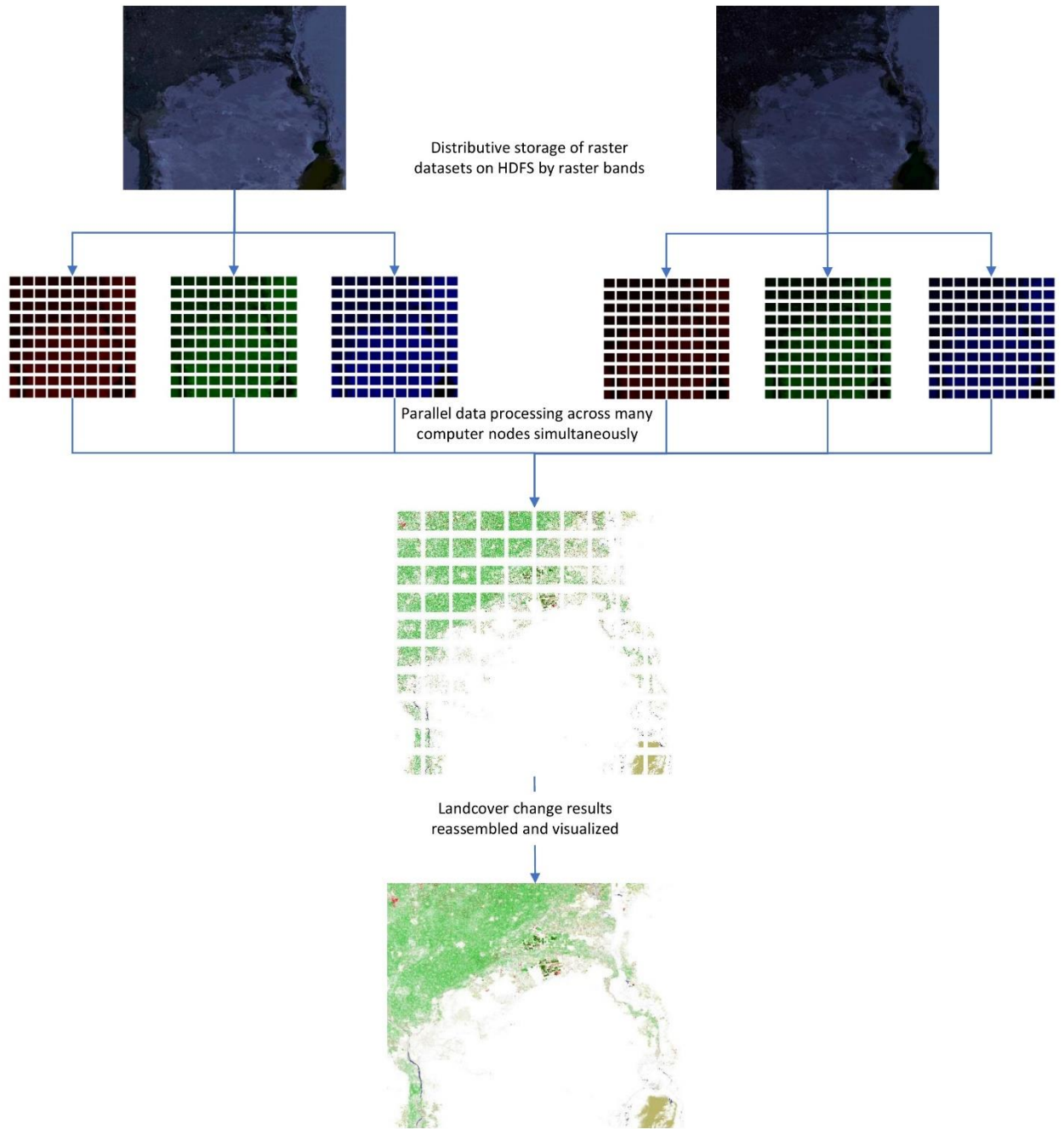


Figure 2-6. Detailed workflow for images classification and visualization

An overview of the steps in the process is shown in Figure 2-6. Four steps were applied including: (1) read and parse each pair of input images and create RDDs for green, red, and near-infrared bands; (2) split data into chunks and process them in

parallel; (3) gather results and assign labels to each pixel; and (4) reconstruct whole images by sorting output RDDs and visualize them in PPM format.

3.2 Cloud-based Distributed SVM Classification

NDVI/NDWI threshold classification was a simple unsupervised classification method that was designed as a testbed in this study to demonstrate the feasibility and performance of our framework. However, more robust classifiers are often needed in actual geographical research. For example, researchers applied unsupervised classification methods such as ISODATA (Ball & Hall, 1965) and K-Means to classify remotely sensed images without ground truth dataset (Ferro-Famil, Pottier, & Lee, 2001). With the support of ground truth data, researchers could use supervised classifiers to achieve classification results with high accuracy than simply applying unsupervised classification approaches. SVM is one of the most commonly used supervised classifiers (Cavallaro, Riedel, Richerzhagen, Benediktsson, & Plaza, 2015). In this study, we developed a distributed cloud based SVM using our framework to demonstrate 1) the classifier in our framework could be adjusted to support different classification tasks without a major change of the framework; 2) our framework can be applied in a geographical research setting that required highly accurate remotely sensed image classification.

SVM is a supervised machine learning model that is able to support classification, regression and outliers detection (Chuang, Su, Jeng, & Hsiao, 2002). The general idea of the SVM model is to seek the hyperplanes that separate the training samples with maximum margins in the space where training samples are located (Cortes & Vapnik, 1995). It requires labeled training dataset for each category

as input to train the classifier and to further categorize the unlabeled portion of a dataset. In this study, we adopted the SVM function from Apache Spark machine learning library (Mllib) to support SVM classification using our framework on the cloud (Meng et al., 2016). Spark Mllib provided a basic linear SVM classification function to address binary classification problems with linearly separable datasets in high computing efficiency. However, in terms of remotely sensed imagery dataset classification for geographical research, a multi-class remotely sensed imagery classifier was a must because only in certain rare cases, the images may need to be separated into two classes. On the other hand, spectral information of multi-spectral images or hyper-spectral images (requiring principle components analysis) was used as vectors to train the SVM classifier with labels. This involved the non-linearly separable issue that is common in high-dimensional vector scenarios. Hence, the original SVM was not able to be applied directly to most of the remotely sensed imagery classification tasks without some modifications.

To solve these issues, we developed a distributed cloud-based algorithm (Figure 2-7) by extending the SVM function in Spark Mllib with a kernel trick (Haasdonk, 2005) and a *one-against-all* computing strategy (Y. Liu & Zheng, 2005) to make it fit with multi-class non-linear classification found in geographical research.

Algorithm 3 Distributed SVM classification algorithm

```
1: def Mapper (label, list<float>vector) //read training dataset
2:   Foreach vector in training dataset:
3:     Emit (label, RBF kernel transform(list<float>vector))
4:
5: def Mapper (label, list<float>transformed_vector)
6:   Foreach class in pre-set classes:
7:     Classifieri = SVM.train(i, list<float>transformed_vector)
8:     Emit (label, list<Classifier>)
9:
10: def Reducer (list<coordinate,vector>, list<label, Classifier>)
11:   float MAX_probability =0.0;
12:   Foreach pixel in image dataset:
13:     Foreach classifier in Classifier:
14:       if SVM.predict(vector) > MAX_probability
15:         MAX_probability= SVM.predict(vector)
16:       label (pixel, list<label, Classifier>)
17:     Emit (list<coordinate>, label)
```

Figure 2-7. Algorithm for Spark-based distributed SVM classification

The first step of this algorithm was to transform the training dataset with kernel function to map the samples into a higher dimension, which helped solve the non-linear-separable issue. In this study, a Gaussian radial basis function (RBF) was applied for kernel trick processing (Chang, Hsieh, Chang, Ringgaard, & Lin, 2010). The second step was based on a *one-against-all* computing strategy to separate the training dataset into n types of patterns. Here n was equal to the total number of classes that the images needed to be classified. For each pattern in n , the entire training dataset was separated into two groups. One group included one class training sample and another group included all the remaining training samples. Then, a binary SVM training function was applied to train those datasets into n classifiers. This binary SVM training function was supported by Spark Mlib. After that, the third step was to import the vector information from the remotely sensed images that required

to be classified, and test each pixel for each classifier. By using the predict function in Spark Mllib, the probability values of confidence for label pixels to each class were returned. Comparing the probability values of a specific pixel for different classifiers, the pixel could be labeled with the classifier that provided the highest probability of confidence. The final step was to output the pixel labels with the coordinates in the input image using a visualization algorithm (Algorithm 2) to re-construct the entire image with the classified results in different colors.

4. Results and Discussion

4.1 Experiment Environment

Two experimental environments were used for this research for local cloud environment testing and real commercial cloud platform testing respectively. The local computing cluster contained 1008 computing vcores and 5.12 Tb memory available with 2 Pb HDFS storage (Table 2-2). The *Amazon EC2* based computing cluster was another computing environment. A three nodes cluster was built and all nodes in it were involved in computing (Table 2-3). The total number of vcores on AWS environment was 8 and the overall memory was 32 G. 20 G storage per node were attached. Both environments configured HDFS block size as 128 M and replica factor as 3.

Table 2-2. Computing environment for Landsat 8 classification on local cloud environment

Hardware				Software	
Role	Count	CPU	RAM	Name	Version
	2		256 G	Apache Spark	2.2.0

Master Node		2 x Intel Xeon E5-2680v4 2.4GHz		Apache Giraph	1.2.0
Computing Node	18	2 x Intel Xeon E5-2690v4 2.6GHz	256 G	Centos	6.9
				Java Server VM	1.8.0_152 64Bit
Network	10 Gbps			Cloudera	5.12.0

Table 2-3. Computing environment for MODIS classification on AWS

Hardware				Software	
Role (instance type)	Count	CPU	RAM	Name	Version
Master Node (t2.xlarge)	1	Intel Broadwell E5-2686v4 2.3 GHz	16 G	Apache Spark	2.2.0
				Apache Giraph	1.2.0
Computing Node (t2.large)	2	Intel Broadwell E5-2686v4 2.3 GHz	8 G	Ubuntu Server	14.04 LTS
				Java Server VM	1.8.0_152 64Bit
Network	1 Gbps			Cloudera	5.12.0

4.2 Experiments Results

In this section, we discuss the application of the workflow on multi-scale and multi-source remote sensing datasets to test the performance and feasibility of the *Spark*-sensing scheme. The performance of the tool and the visualization of the experimental results is shown. The flexibility, extensibility and accessibility gained by using the *Spark*-based solution for remote sensing image dataset processing is discussed further.

The first experiment was performed on a 20-node YARN computing cluster. In this experiment, a multi-scale remote sensing image dataset with the *Spark*-based NDVI/NDWI threshold classification algorithm was successfully processed. The detailed data size and processing times is shown in Table 2-4. The processing time

was recorded under configurations with default block size, 50 executors with 20 cores each, and 20G executor memory. The MODIS dataset and SC_small took 54 seconds and 67 seconds respectively to finish the classification tasks with NDVI/NDWI threshold classification approach. It seems like the performance of them was just similar as they were processed with a desktop series software (e.g, ENVI). The reason is it will take time to launch a cloud-based job includes submit job, read dataset from HDFS, processing, generate results and recycle system garbage so that if the processing time is even shorter than other parts in the lifecycle of a cloud computing job, the overall performance is not quite obvious good in comparison with some desktop solutions. However, for tasks like SC_large or IMAR, the performance of Spark-based computing was significant faster than using desktop solutions. For example, in IMAR classification process with SVM, a desktop solution took extreme longer time (over 24 hours) than finished it with cloud computing approach in about 38 minutes according to our tests.

Table 2-4. Data size and processing time

	MODIS	SC_small	SC_medium	SC_large	IMAR
Image Size	~0.25 G	~1.2 G	~10 G	~ 107.4 G	~1050 G
Classifier	NDVI/NDWI	NDVI/NDWI	NDVI/NDWI	NDVI/NDWI	SVM
Processing Time	54 s	67 s	276 s	1018 s	2284 s

Furthermore, *Spark* can offer highly efficient processing for remote sensing images, especially with a relatively large cluster (with available resources more than the resource needs of a specific task), a sufficient computing resource can support the entire distributed computing process, ensuring that the processing time does not significantly increase as the data size increases. From the perspective of algorithm design, by anatomizing the overall processing time in each operation time-consuming

segment, the join operations were found to be very time-consuming, especially for a dataset with massive numbers of pixels. By contrast, mapping operations for classification were much faster than the join operations. Hence, designing an image processing algorithm with fewer join operations and appropriate partitioning to reduce the data blocks moving may enhance the performance. We found that repartitioning operations should also be avoided because these can result in a very time-consuming shuffle process.

The LULC feature uses four classes (other represents non-classified pixels or no data): dense vegetation, sparse vegetation, barren areas, build-ups and sand, and water areas and wetlands (Figure 2-8). In the legend, colors are set for the classification results by pixel. Other indicates those areas in the images with no data or with erroneous data. In this figure, the changes between each pair of images can be clearly seen. With the maps of 2014 and 2015 as an example, the results show that April 2014 was very dry for this area and during this year, the new branch of the Suez Canal was being built and filled with water. It also showed that there was new vegetation growing along the Canal; such vegetation may be new farmland because most of the vegetation regions were in artificial (human built) shapes.

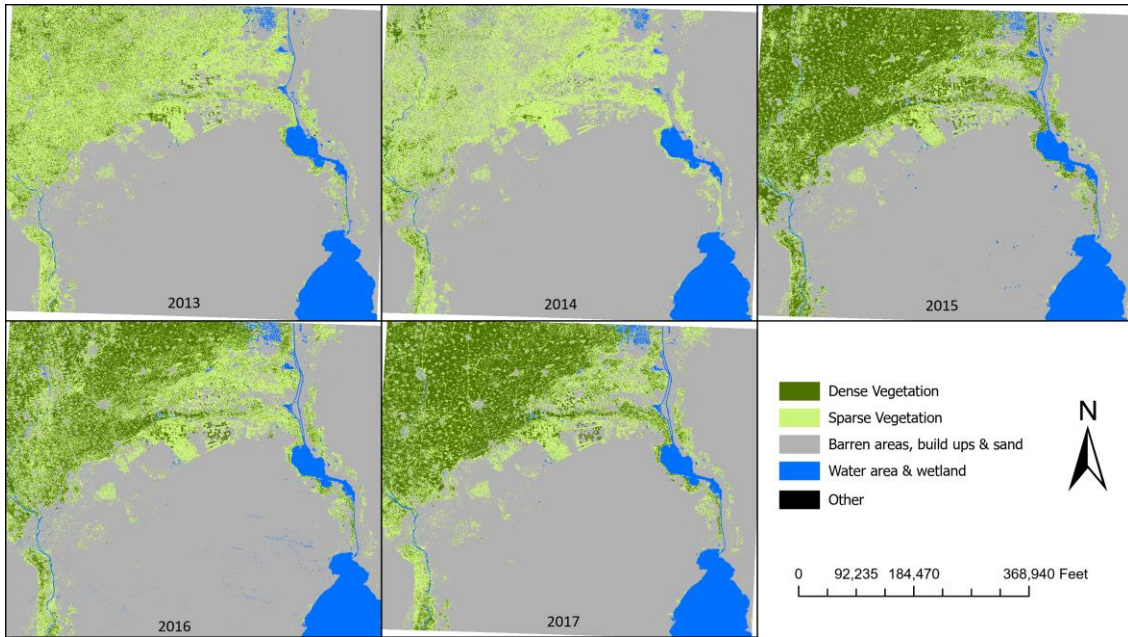


Figure 2-8. Visualization of NDVI/NDWI threshold classification results for time series from 2013 to 2017

In addition to enhancing overall performance with the algorithm design, the second experiment was developed to further study if the execution configurations may affect processing performance. Only the large-size *Landsat 8* dataset was used in this experiment to assess the processing time with different execution configurations. According to *Spark* performance tuning official documents (Apache Software Foundation, 2018b), the executor numbers, executor cores and executor memory were three main factors that may effect on performance of *Spark*-based applications. Budgeting available computing resources in advance was usually needed for users to gain satisfying processing performance. Here, three different configuration sets were assigned as following: (1) 10 executors with 100 G memory and 100 cores each as set 1; (2) 50 executors with 20 G memory and 20 cores each as set 2; and (3) 500 executors with 2 G memory and 2 cores as set 3. These three configuration sets were designed with same total computing vcores and memory. It is also worth pointing out

that block size may sometimes effect performance as well. As discussed above, repartition operations (especially for increasing partitions) usually should be avoided to eliminate unnecessary shuffle processes. Under this circumstance, partitions will be mainly decided by block size during I/O process when RDDs were created. If the block size were too small, massive numbers of partitions will be created especially with a very large input dataset, which will lead to increasing the overhead of tasks management, though a coalesce operation may be applied to decrease the partitions sometimes without shuffling. If the block size were too large, only a few partitions will be created so that not all the cores in the available computing resource can be sufficiently utilized. In other words, the feature of parallelism was not fully exploited to enhance the performance. Here, three different block sizes were set during the experiments. The performance changes with processing the dataset under different execution configurations and different block sizes are presented in Figure 2-9. The shortest run was offered by 50 executors with 20 G memory and 20 cores each under default block size. With the same executor configuration set, the performance was a bit lower with the 64 M block size, which may result from the total tasks number being increased with smaller block size. However, the execution time for each task was not significant reduced with the corresponding settings. The performance of the same configuration set with 256 M block size was also beaten by the case with the default block size. This was due to the partitions generated under this block size being too few, so that parts of the executors were not active during processing. This problem became more obvious when applying configuration set 3. Because the number of executors in this case was far more than tasks for this case, and too many

computing resources were in idle, this lead to the lower performance. The execution configuration set 1 showed very similar performance with set 2 under each block size setting. However, by monitoring the utilization of cores with set 1, this was lower than with set 2. This may mean that with increasing numbers of executors and decreasing the number of cores per executor, may lift the utilization of cores and may also help enhance the overall performance in some cases especially for very large datasets with relatively limited computing resources.

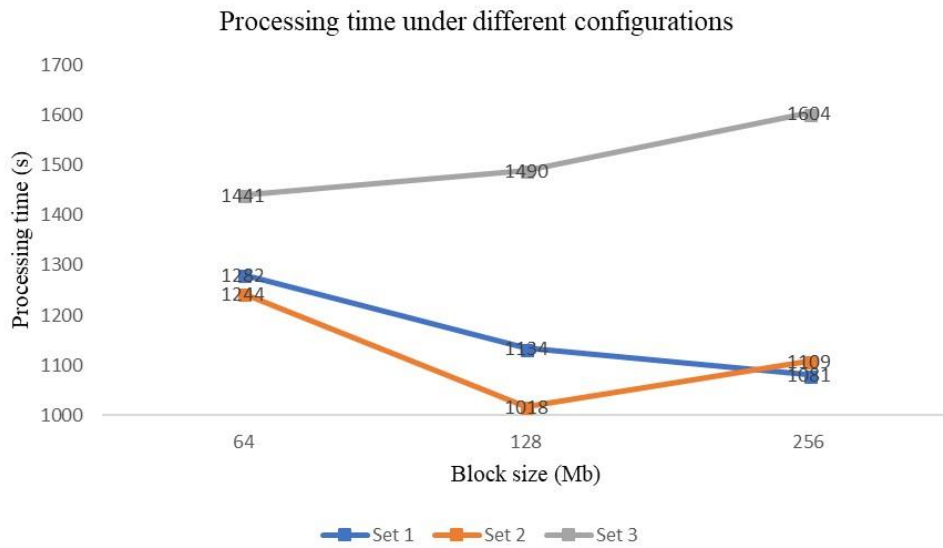


Figure 2-9. Processing time of large-size Landsat dataset under different configurations

The third experiment adopted a distributed cloud based SVM into our framework to process over 1050 GB of Landsat 8 images with a classification task set in IMAR. Cloud cover was inevitable for remotely sensed images in such a large area during the one month acquired period. Hence, a cloud removal pre-processing step was performed using a cloud mask attached with the Landsat 8 dataset. The pixels that were covered by cloud were labeled as “No Data” and were ignored during sample points extraction and classification.

To train the SVM classifiers, 30000 training sample points were extracted from the LULC map in 2016 and were applied to train 7 classifiers according to the pre-defined number of classes. Those sample points were generated by using a stratified random strategy to make sure there were enough sample points for each class. The overall processing time was 2284 seconds (Table 2-4) and the visualization of 7 LULC classes was generated using Algorithm 2 (Figure 2-10).

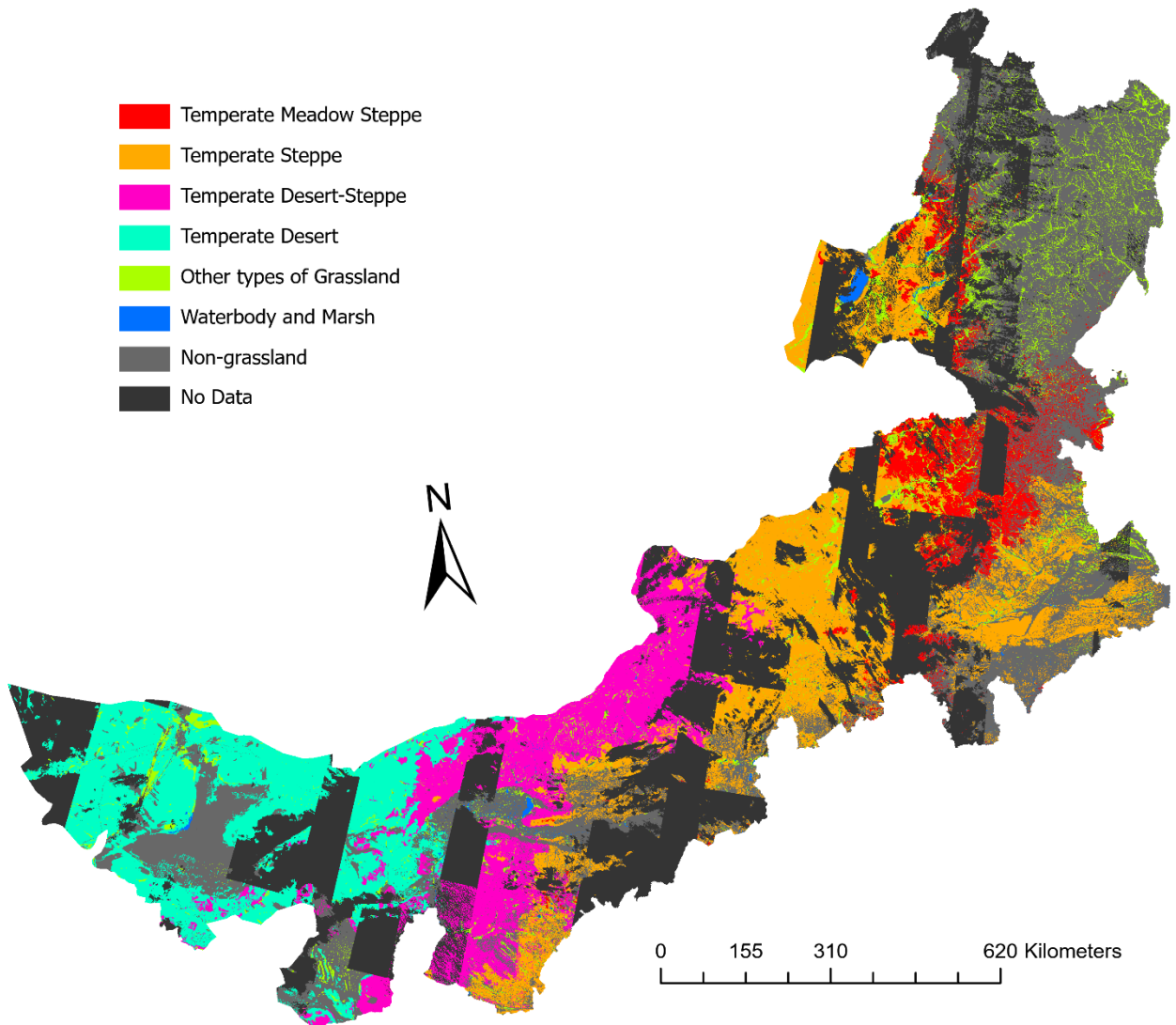


Figure 2-10. Visualization of SVM classification result in IMAR

As we have discussed, the developed tool can be deployed on commercial cloud platforms with no change (if the hardware configuration is highly different, strategies of balancing workload may need to be reconsidered). The last experiment was designed to use this framework on *Amazon EC2* to process a *MODIS* dataset. Different from *Landsat 8* products with band 4 as RED, band 5 as NIR and band 3 as Green, *MODIS* (MOD09GA) products set the RED band as band 1, NIR as band 2 and GREEN as band 4. Except for changing the band index for input, the processing tool was ready to launch with no additional adjustment needed in coding. This experiment ran on 3 nodes of *Amazon EC2* cloud computing cluster. By applying 2 executors with 2 cores and 4 G memory each, the experiment was successfully completed in 54 seconds. It is worth pointing out that 54 seconds processing time was not fast with such a small input dataset as ~0.5 scene *MODIS* image was only about 50 M. In this case, most of the time was occupied by job submission, task management, resource allocation etc., and so the performance of *Spark* applications was only showed to be significantly improved with the relatively large dataset. However, this experiment still was able to demonstrate that the proposed framework can be deployed on actual commercial platforms to process multi-source remote sensing images with only minor parameter adjustments.

These experiments represent a robust solution for constructing remote sensing image processing tools for multiple purposes that were flexible (the ability of the tool to fit multi-source datasets at different scales), extensible (the ability of the tool to expand and accommodate the volume of computing to be resolved) and accessible (the ability of the tool to access data from multiple storage platforms and locations on

local resources, data centers, and the cloud). The test image datasets in these experiments were *Landsat 8* with 30 meters spatial resolution and *MODIS* images with 500 meters spatial resolution. However, it should be noted that the input dataset can be any raster-based images with different spatial, temporal, and spectral resolutions, because the algorithm implemented in our experiment was a pixel-based processing algorithm. Following a similar mechanism, all pixel-based algorithms can be implemented by processing the DN values of each pixel to generate required results under this framework. Hence, regardless of whether this workflow was applied to an existing dataset or used with a next-horizon dataset that will be collected in the future with current sensors or new sensors, this solution was capable of handling the tasks with only minor changes in coding, thereby saving the high cost usually invoked in reprogramming different tools for different research goals.

With the *Spark*-based implementation, as demonstrated in the experiment, the main structure of this tool does not require modification as the volume of dataset changes. It is possible for example, that even if a *Landsat 8* dataset that covers a complete year (47.33 Tb) (Y. Ma et al., 2015) is involved in processing at the same time, by the support of cloud platforms, in theory users can always gain sufficient computing resources (memory especially, for *Spark*). Consider that *Amazon EC2* now provides the “*x1e.32xlarge*” instance, which contains 128 virtual CPUs, ~4 Tb memory, and ~4 Tb storage. Users can apply fewer than 20 instances to implement in-memory computing with *Spark*-based approaches for this dataset in many different processing purposes. Nevertheless, to maintain the high performance of the tool, the partitioning strategies should be tailored based on real execution environments,

especially for clusters with unevenly distributed computing resources and networking performance.

As discussed in the introduction, *Spark*-based approaches can easily access HDFS, *Amazon S3*, *Cassandra*, and *HBase*. Benefitting from cloud storage and management development, an increasing amount of data has been stored in the cloud and is open to the public. The approach as demonstrated, provided cloud-based data resources to support users in performing a “pure” cloud analysis and in creating new products from it, without transferring unnecessary original and intermediate data to local storage before the final results are generated. This solution can also support data from multiple sources at the same time. For example, users can access cloud-stored public data as part of their data source and can also access and load their private data stored on local HDFS in a single *Spark* application.

5. Conclusions

In this study, we demonstrated that the current state-of-the-art for remote sensing in big data era, involving RS datasets being generated from existing and next-generation satellites and earth observation platforms was, in many cases, proceeding at paces that outstrip our analytical capabilities to keep up with the information products associated with those data. While data and analysis are out of alignment, researchers perhaps have missed opportunities to build the necessary science that might otherwise be attainable if data and analysis could be better-connected. In this study, we discussed a set of current and widely-used approaches that have been developed in previous studies as a means to cater to the call of the community for an effective and efficient computing framework to process remote sensing datasets.

Based on a comparison with other possible approaches, using *Spark* to build a robust scalable tool in a cloud environment was demonstrated to be an important and practical option to match data with analysis at scale. To this end, a *Spark*-based multi-scale remote sensing classification has been introduced, and its successful deployment and experimental testing in a cloud environment has been shown.

The approach in this study suggests several promising advantages. First, the scheme for *Spark*-sensing offers considerable flexibility for processing remote sensing datasets in multi-spatial, multispectral, or multi-temporal cases. Indeed, shifting between resolutions and spectrums is possible with slight adjustments, thereby significantly saving the time and cost of re-programming brand new toolkits for different purposes. Second, this scheme makes it possible to exploit the benefits of cloud platforms to gain (theoretically) unlimited computing resources, with highly efficient performance to potentially support very big RS imagery datasets processing. Third, the presented approach is natively highly accessible to multisource data storage, even in the cloud, which is useful in reducing data transformation costs.

The tools and framework discussed in this study were obviously a prototypical framework and thus, improvements can still be made. The work here serves to demonstrate the general principle and mechanisms necessary to launch experiments in this area, and hopefully this research will encourage others in the community to build on this foundation. One extension of our approach could include the implementation of more complex remote sensing image processing algorithms, especially with respect to classification (e.g., random forest, or even non-pixel-based classification approach such as objected oriented classifiers), to extend the types of cases encountered in

different research areas. Another improvement could be explored in designing better partitioning strategies to further enhance the computing performance. Moreover, using a dataset from the cloud directly may reduce the unnecessary data transfer from the data source to the cloud environment, which may better fit the usage in real-world problem-solving environments.

Chapter 3 : Data Gap Filling for Land Use and Land Cover Study Using Cloud-based Distributed Markov Chain Cellular Automata

1. Introduction

1.1 Significance of Land Use and Land Cover Change

Alteration of land surface conditions affects the earth ecosystem functions significantly and profoundly in many different ways (Lambin et al., 2001; Mitsova, Shuster, & Wang, 2011). For example, changes in land use, such as deforestation and urbanization resulting from human activities, have become important factors that impact the global carbon cycle and the global climate (Foley et al., 2005). In addition, land cover changes can also affect local and global climate from both a biophysical and biochemistry perspective (Feddema et al., 2005). For example, changes in albedo and roughness of surface due to vegetation structure alteration may affect surface energy, such as momentum and heat transport (McGuffie, Henderson-Sellers, Zhang, Durbidge, & Pitman, 1995). Environmental factors and human activities together can both contribute to these changes. However, in past few decades, the unprecedented expansion of development and the rapid pace of urbanization makes human activity a principal factor that affects climate through ongoing land use and land cover (LULC) change (Singh, Mustak, Srivastava, Szabó, & Islam, 2015). For this reason, understanding the current state and trend of regional and global LULC change are

very important topics for regional and global sustainable systems research and development (Fan, Wang, & Wang, 2008).

Many research studies have been undertaken to explore the effects of LULC change on climate. In general, observation and modeling are two main directions for this research (Y. He, Lee, & Warner, 2017). For example, temperature records were applied by Kaufmann and Stern as observational evidence to analyze global climate change under human influence (Kaufmann & Stern, 1997). Douglas et al. modeled the moisture and energy fluxes from the perspective of biophysics to explore the impact of agricultural land use in India (Douglas et al., 2006). Pielke and colleagues reviewed LULC change studies via modeling and observational evidence for each continent across the globe (Pielke Sr et al., 2011). In these studies, both observational and modeling studies required LULC maps as input to detect the pattern of changes in LULC over time. However, due to the limited availability of large area and long-term LULC maps, previous climate studies sometimes simply applied a single map from a specific year to represent the LULC for an entire study period (Zhu, 2012). This research presents an approach that offers high quality large-scale (i.e., datasets that are beyond the computational capacity of a single workstation) LULC dataset where data gaps are minimized, to support related research.

1.2 Missing Data in LULC Change Research

In 1991, Townshend et al. claimed in their research that only remotely sensed (RS) data could potentially provide accurate and repeatable global LULC for monitoring change (Townshend et al., 1991). From then on, many researchers (Dewan & Yamaguchi, 2009; J. Liu et al., 2014) employed remotely sensed data to

generate LULC maps. However, this research either applied relatively low temporal frequency of acquired images or employed mixed data sources from multiple different satellites to build an LULC time series. The main reason is that it is computationally challenging, and missing data has been a stumbling block to building a high resolution LULC dataset for large scale and long term time series using the same optical RS dataset data source for a continuous period, and for the same time intervals (X. Li, Shen, Li, & Zhang, 2016).

The problem of missing RS data occurs mainly due to two reasons: one is that aging sensors may experience hardware failures and a second reason relates to weather. Under the first scenario if there is a malfunction, a satellite cannot send accurate and correct ground information back as usual. For example, the scan line corrector of Landsat 7 enhanced thematic mapper plus (ETM+) has permanently malfunctioned since 2003 (Zeng, Shen, & Zhang, 2013), which impedes the use of Landsat ETM+ datasets for research purposes after 2003. When satellites are scanning cloudy areas or areas where rain is falling on the earth's surface, those thick clouds or haze may obscure the land surface, and in turn result in missing ground information for those areas. Although many different cloud removal approaches have been developed to support various cases (Gafurov & Bárdossy, 2009; Tseng, Tseng, & Chien, 2008), those algorithms cannot always return optimal results especially when the input images are heavily cloud-contaminated. Hence, a critical research need is to develop efficient methods to reconstruct missing data in imagery dataset since gaps in remotely sensed data are inevitable, especially for longer term and large scale LULC studies (X. Li et al., 2016).

1.3 Gap Filling Modeling Approaches

Shen and his colleagues reviewed current reconstruction approaches to address missing data issues in optical remotely sensed datasets (Shen et al., 2015). The four classes of methods that they derived have been the focus of much research: 1) spatial-based methods (Ballester, Bertalmio, Caselles, Sapiro, & Verdera, 2000; C. Zhang, Li, & Travis, 2007); 2) spectral-based methods (Gladkova, Grossberg, Shahriar, Bonev, & Romanov, 2012; Shen, Zeng, & Zhang, 2011); 3) temporal-based methods (Melgani, 2006; J. Zhang, Clayton, & Townsend, 2011); and 4) hybrid methods (Qin et al., 2014). Spatial-based methods involve filling in for missing data by traditional interpolation methods or further enhancing interpolation techniques using spatial relationships. Spectral-based methods use redundant information found in other normal bands to derive the data in missing bands. Temporal-based methods apply data that were for the same geographical region but with different acquisition times as supplementary information to calculate the missing data. Finally, hybrid methods use multitemporal *and* multispatial datasets to support filling missing gap for the same time period and for the same locations. One example used a Markov Chain Cellular automata model to reconstruct the missing data in a time series gap for LULC mapping as well as for predicting future data based on current known information (Halmy, Gessler, Hicke, & Salem, 2015).

Markov Chain models are stochastic models that have been widely used for predicting LULC change at different scales (Baker, 1989). Markov Chain models contain a series of random values. The probabilities of those values at specific time steps are dependent only on the value at the previous time step (Fan et al., 2008). This

is part of a basic assumption of Markov Chain models from physics where the probability of the state in a system at a certain time point can be determined by the state at a previous known time point (Bell & Hinojosa, 1977). Applying this principal in LULC studies, researchers assume that the LULC can be regarded as a stochastic process and the different LULC classes can be treated as states of a chain, which makes this model a simple approach to modeling LULC change (Weng, 2002). Markov analysis is suitable for spatially dependent LULC data because the statistical independence of the data will not be necessarily checked for those data (Overmars, De Koning, & Veldkamp, 2003). More importantly, unlike Geomod (Pontius Jr & Chen, 2006) or other approaches that simulate one-way transitions from one class to another, Markov Chain models can be used to simulate all classes in LULC change. However, Markov modeling itself offers no spatially explicit support. In other words, the transition probabilities that can be derived from observed data sets can potentially predict accurate change among LULC classes, but the spatial distribution of these changes will not be represented explicitly (Ye & Bai, 2007). Fortunately, this shortcoming of Markov Chain analysis can be addressed by integrating other models that can add spatial properties to the results (Halmy et al., 2015).

A cellular automata (CA) model is a spatial and dynamic model that can be utilized to simulate and represent complex spatial and dynamic processes in many research fields (Mendes, Santos, Martins, & Vilela, 2001; Qiu, Kandhai, & Sloom, 2007; Zhao, Billings, Coca, Ristic, & DeMatos, 2009). In geospatial science, the CA model has been extensively used to simulate dynamics of land use change and other natural geo-processes (Di Gregorio, Kongo, Siciliano, Sorriso-Valvo, & Spataro,

1999; C. He, Okada, Zhang, Shi, & Li, 2008) The simple modeling and computational efficiency of CA models make them a possible choice for modeling LULC dynamics at multiple scales. Grid-based dynamics are structured to operate locally and be controlled by transition rules that are designed by modelers and are the same for each grid cell (Fonstad, 2006). The states of all cells are updated simultaneously based on the progression of time and according to transition rules.

Integrating Markov chain models with CA models is a robust approach for simulating LULC change at different scales (Guan et al., 2011). With the addition of spatial information from a CA model, the Markov-CA has the advantage of simulating two-way transitions among all LULC classes (Theobald & Hobbs, 1998). However, even though the Markov-CA model has been widely used in LULC change prediction and information reconstruction, the fact is, until now, the vast majority of CA-based models are implemented to run on a single computer or single workstation, with the result that computing capacity is often limited. Because CA-based models are theoretically scalable to any size, improving the computational capabilities of cellular automata can be of significant benefit to researchers. Especially in the Big Data era, to design an approach for how to make CA-based models benefit from big data input and how these models can be accelerated via cloud computing, will better support LULC change analysis and gap filling procedures especially for large areas and longer term time series research.

In this study, we discuss a workflow that uses a cloud-based Markov CA simulator to support LULC analyses including gap filling. We developed a distributed computational framework to support our study using a cloud environment. We tested

our framework to implement gap filling for a LULC dataset collected for The Inner Mongolia Autonomous Region (IMAR), China with 30-meter resolution LULC mapping from 2000, 2010 and 2016 as input data. Eleven environmental and human factors including digital elevation model (DEM), slope, precipitation, temperature, population density, livestock density, compensation policy, water area, roads, railways, and other human land use are applied as constraint and restraint factors to calibrate the Markov-CA model. LULC classes were not expected to change into other classes as a result of the impact of any of the constraint factors. On the other hand, some classes were possibly changed into other types of LULC classes under the impact of some of the restraint factors. The accuracy of gap filling is calculated using a confusion matrix and kappa statistics.

2. Study Area and Dataset

2.1 Study Area

The IMAR is located in northern China (Figure 3-1). It is the third largest province in China with 103 counties (or banners as named by the local Mongolian population). The overall area is about 1.18 million square kilometers extending from 37° 24' N to 53° 23' N and 97° 12' E to 126° 04' E. The shape of IMAR is long and narrow. The climate is variable across the region as in the western part of IMAR it is semi-arid, while the eastern and northern areas correspond to a semi-humid climate.

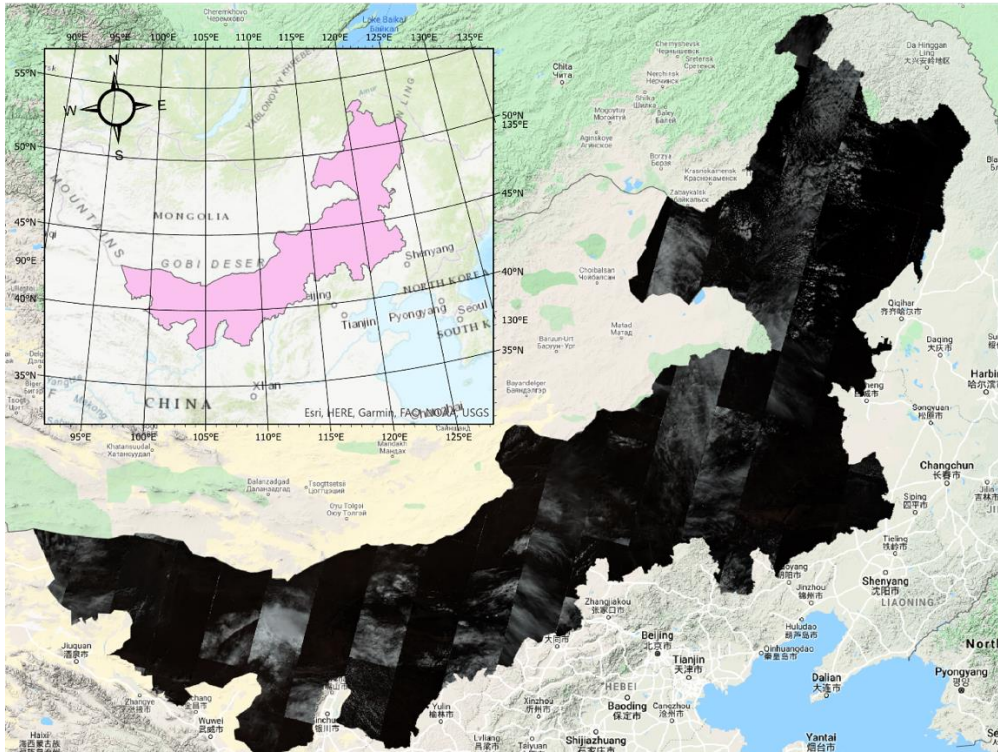


Figure 3-1. Landsat 8 OLI images at IMAR from on August 2016

The major land cover in IMAR is grassland. The Hulunbeir Grassland, located in northern IMAR, is one of the largest natural grasslands in the world. In this study, we classified IMAR into ten LULC classes (Table 3-1). In this table, there are seven grassland classes classified by using Ivanov moisture coefficient (Kononova, 1961).

Table 3-1. IMAR LULC classes including grassland subclasses

Class ID	Classes	Subclass ID	Subclasses
1	Temperate Meadow-steppe (TMS)	30011	Plain and hilly area of meadow steppe subclass
		30012	Mountain land of meadow steppe subclass
		30013	Sand land of meadow steppe subclass
2	Temperate steppe (TS)	30021	Plain and hilly area of steppe subclass
		30022	Mountain land of steppe subclass
		30023	Sand land of steppe subclass
3	Temperate Desert-steppe (TDS)	30031	Plain and hilly area of desert-steppe subclass
		30032	Mountain land of desert-steppe subclass
		30033	Sand land of desert-steppe subclass
4	Temperate Steppe-desert (TSD)	30070	Temperate Steppe-desert subclass
5	Temperate Desert (TD)	30081	Gravel soil of desert subclass
		30082	Sand soil of desert subclass

		30083	Salt soil of desert subclass
6	Lowland Meadow (LM)	30151	Lowland meadow subclass
		30152	Salinized lowland meadow subclass
		30154	Marshy lowland meadow subclass
7	Montane Meadow (MM)	30161	Low-middle hills of mountains meadow subclass
		30162	Subalpine of mountains meadow subclass
8	Marsh (MA)	30180	Marsh
9	Water Area (WA)	w	N/A
10	Non-grassland (NG)	f	Forest
		s	Sand
		r	Road
		1	Human Land Use and all others non-grassland

In recent decades, researchers have studied IMAR focusing on the LULC classification (Lan & Xie, 2013), grassland degradation (Tong, Wu, Yong, Yang, & Yong, 2004), ecosystem stability (Bai, Han, Wu, Chen, & Li, 2004), long term climate change trends (Xie et al., 2018) and net primary productivity (Z. Wang, Zhong, Lan, Wang, & Sha, 2019). In these studies, RS images from Landsat and MODIS are important data sources for both input and validation. However, the images are often affected by weather conditions and unable to generate products with acceptable quality due to the high degree of gaps in the data. For example, the Landsat 8 OLI images in Figure 3-1 represent the best quality images available for August 2016. After running a cloud removal algorithm with band 9 of Landsat 8 as input, the cirrus clouds are detected and removed. The portions of Figure 3-2 in yellow, shows cloud covered areas during August 2016 that correspond to where data gaps exist in IMAR during this time period.

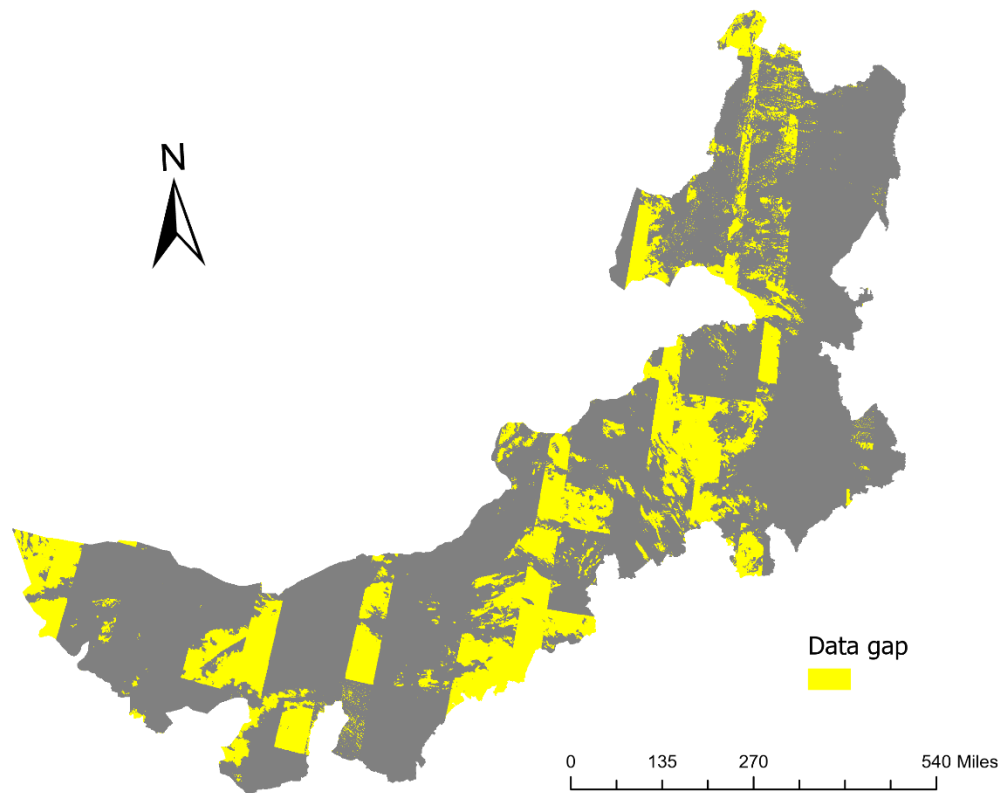


Figure 3-2. Data gap in IMAR on August 2016

There are in total, ~ 1.28 billion (1281956192) pixels/cells per band in this study area, and approximately 0.49 billion (492899116) pixels were covered by cloud during this month, i.e., the cloud coverage was over 38.45%. In the northern part of the region, where cloud coverage was above 85%, clouds were a major obstacle that potentially impedes LULC research on this region.

In this study, we undertake gap filling for the remotely sensed dataset for the entire IMAR region. The processing framework we propose was able to handle and process the entire area during a single run. To assess the accuracy of our simulated results, we focused in particular on three locations in IMAR, the Hinggan League region, Xilingol League, and Bayan Nur region. The main objectives of this study

were: 1) to develop a workflow to be able to process LULC data gap filling and easily to be extended to very large scale with sufficient computing resources; 2) to implement a cloud-based Markov Chain CA model and test it using a cloud computing environment; 3) to apply historical LULC data and auxiliary data collected from 2000 and 2010 as training data to simulate the LULC coverage for 2016, and 4) to assess the accuracy of the gap filling results.

2.2 Data Availability

A very high accuracy LULC vector mapping dataset that was generated by visual interpretation assisted with sample plots that were available for 2000, 2010 and 2016 for the IMAR region, and which were used as input and validation LULC datasets in this study. An initial set of 26 land cover subclasses were reclassified into the ten classes of land cover types (Table 3-1). A DEM dataset was created using The Advanced Spaceborne Thermal Emission and Reflection Radiometer (ASTER) Global Digital Elevation Model (GDEM) program, that was downloaded from NASA Earthdata portal. Slope for this area was calculated by processing the DEM data. A roads and railways vector dataset was acquired from OpenStreetMap (OpenStreetMap, 2020). Distance for each cell to roads and railways was calculated as one of the suitability factors for LULC change. Meteorological data including yearly precipitation and temperature were provided by the Chinese Meteorological Data Service Center (CMDC, 2020) from 2000 to 2016. These data were interpolated and rasterized using AUSPLIN, a meteorological data processing framework (Hutchinson & Xu, 2004).

Population and livestock survey data from 2000 to 2016 were provided by the Department of Agriculture and Animal Husbandry, China. Population density and livestock density were calculated based on county vector data. Compensation policy data that was in effect for 2010 to 2015 was also provided by Department of Agriculture and Animal Husbandry, China.

3. Method

3.1 Workflow

In our study, we used historical LULC data from 2000 and 2010 as well as available data from 2016 to train a Markov Chain-CA model to simulate LULC data that was missing, causing gaps in the record. Ecological, economic, demographic and urban factors were also integrated as part of model training as constraint and restraint conditions to improve the accuracy of simulation results. The workflow for gap filling LULC involved five different components (Figure 3-3).

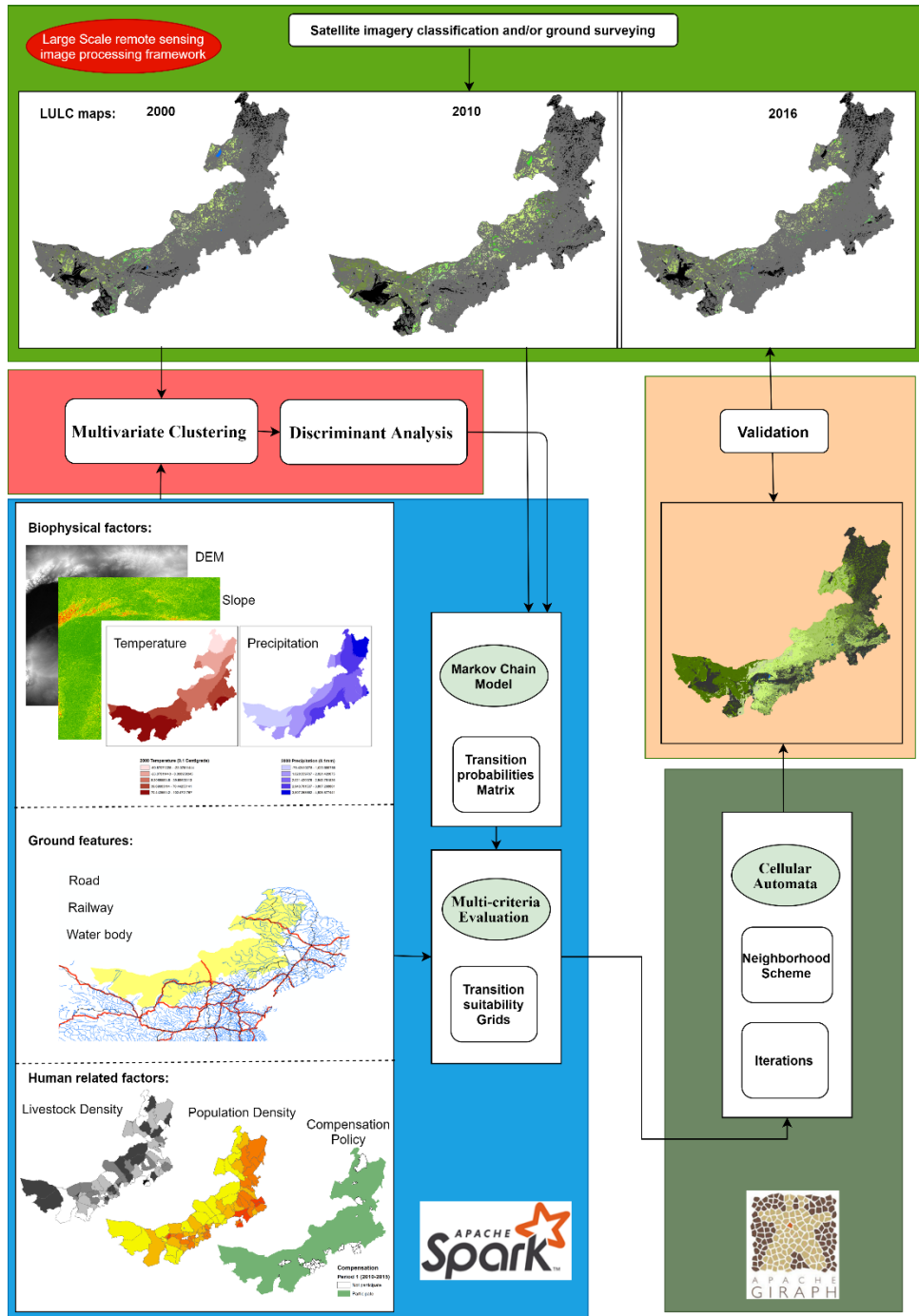


Figure 3-3. Workflow for cloud based LULC gap filling process

The first step in the workflow was LULC classification and mapping. A common approach to generate LULC datasets is by performing image processing and classification on RS imagery datasets, e.g., Landsat 8 OLI. In other words, LULC

data can be generated by using a large-scale RS processing framework on a high-performance computing cluster or cloud environment. A LULC dataset can also be generated from a ground survey, which was available for the IMAR region and for our study. Secondly, as the environment, climate, and human activities were variable across the study area, the region was divided into sub-regions (SR). This is important especially for a large scale study because it allowed the design of functions to be tailored to integrate constraint and restraint factors for subregions with relatively similar impact factors, which was better able to calibrate and train the model for each region. Third, a Markov Chain model was trained, and a set of suitability grids were built using multi-criteria evaluation (MCE) applied to the LULC data. In this step, existing data for the region for 2016 was used as a mask to extract LULC data from 2000 and 2010. A cloud-based Markov Chain analysis was processed to generate transition probability matrices for each sub-region by using the masked LULC dataset from 2000, 2010 and 2016 as input. For each sub-region, difference functions and control points for each constraint and restraint factor based on statistics and conditions of eleven environmental and human impact data were designed and applied to determine the suitability and location of transitions using MCE. This step was integrated using Apache Spark. Then, the CA model was run to add a spatial component, where a spatial neighborhood filter was applied via the CA model to assign weights to suitability scores of each cell. The number of iterations will be assigned at the initialization step of the CA model. Once the CA processing starts, it updated cells status with transition rule to decide the highest possible simulated transitions among each LULC class. This step was built using Apache Giraph. Lastly,

an accuracy assessment was run for the simulated results using the 2016 LULC data as validation. A detailed discussion for each step of this workflow is presented in the sections that follow.

3.2 LULC Classification

To deal with LULC studies starting with raw RS imagery datasets, researchers are required to handle LULC classification and mapping, and also handle the processing of some ground truth points. For this research, the analysis of the entire IMAR region is considered as a large-scale study. Using, for example, Landsat 8 OLI, a widely used, medium spatial resolution RS imagery dataset with a relatively short revisit period (U.S. Department of the Interior & U.S. Geological Survey, 2018), to cover all of IMAR, it would require ~190 scenes of images for a single month. To test our cloud-based Markov CA simulator, two years of input data and one year of validation data were used. This involved, ~600 scenes of Landsat 8 OLI images that were processed including handling atmospheric correction, cloud removal and mosaicking. The overall data input was over 1TB, which precluded processing on a single workstation in a timely fashion. In this study therefore, exploiting a big data processing framework was a must.

Processing RS datasets and classifying them into LULC is an active area of study and many researchers have developed methods and frameworks to process large scale RS images. For example, Huang and her colleagues used Message Passing Interface (MPI) as a computing framework to support dust storm simulation and forecasting on the Amazon EC2 commercial cloud service (Q. Huang et al., 2013). Wang *et al.* developed pipsCloud, a cloud-based approach to process RS on-demand in real-time

(L. Wang et al., 2018). For this research, we developed a Spark-based RS image processing tool that can be deployed on cloud platforms and potentially to be extended to very large scale with enough computing resources (Lan et al., 2018).

However, it is worth pointing out that all LULC classification and mapping needs to pass an accuracy assessment before use. The quality of input images, auxiliary data and classification methods will all impact the final accuracy of results. We directly applied already existing masked LULC maps from 2000, 2010 and 2016 as input for Markov Chain training analysis, and used the entire LULC data from 2016 for validation. The highly accurate, already-classified LULC maps can help us better focus on the task of gap filling by using a cloud-based Markov CA. To derive measures of overall accuracy for the gap filling task, confusion matrices were used and Cohen's Kappa statistics were also calculated to determine the overall accuracy and consistency for this method.

3.3 Sub-regions Clustering

To implement LULC gap filling, it is necessary to use existing dataset to train a model to simulate the missing LULC data in gap of the targeted time. However, it led a problem that, for a large-scale study, the LULC, environmental factors and human activities could not be easily distinguished along the whole study area. The model training process will try to find the optimized parameters for fitting the whole area if the datasets of this area were used as a single input. It should be noted that this may result in models for certain specific regions of the area that might not have best-fit parameters due to the fact that those parameters also have to be fitted for other areas

at the same time. Hence the overall accuracy may be lower and the optimized gap filling results cannot be achieved.

To solve this issue, a pre-processing step was performed for the study area before model training using machine learning techniques. Multivariate clustering was applied using the LULC dataset, environmental conditions, and data on human activities to divide the whole study area into several sub-regions. Then, we designed and calibrated a tailored model for each sub-region to run the gap filling. In this study, a county-based multivariate geographic k-medoids clustering analysis was performed (Ippoliti et al., 2019). Several continuous clustering sub-region parts were created as shown in following map. In this analysis, LULC classes, DEM, slope, precipitation, temperature, population density and livestock density were included as variables. Zonal statistics were used to assign values of each variable into each county in IMAR. The cells were sorted by county, and the majority value was assigned to a county as the LULC value. For all other variables, the mean value was calculated for all cells in each county to generate the final LULC value.

Multivariate clustering was implemented by ArcGIS Pro 2.5.0. To assess the optimized clustering strategies, different clusters were generated from 2 to 30 clusters. A Calinski-Harabasz pseudo F-statistic (Caliński & Harabasz, 1974) was applied to rank the top three cluster candidates for further processing. This test generated the minimum spanning tree by calculating the within-group similarity and between-group difference. In other words, spatially contiguous areas were divided into clusters by maximizing the within-group similarities and between-group

differences. Testing showed that clustering IMAR into six clusters was one of the best clustering candidate strategies (Figure 3-4).

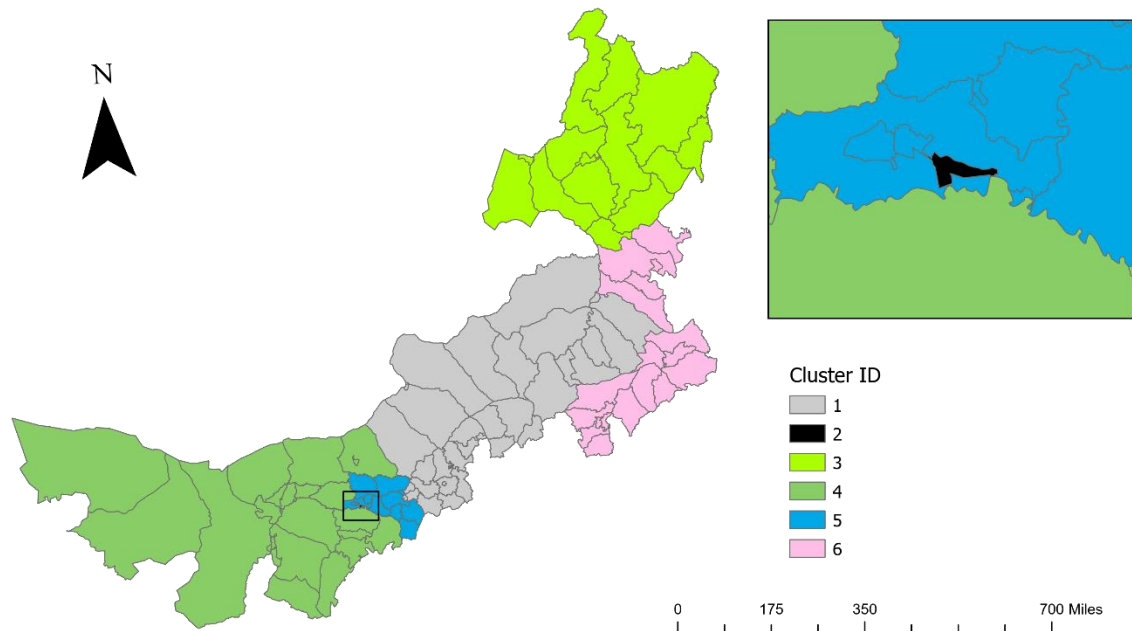


Figure 3-4. Six clusters generated by county-based multivariate geographic k-medoids clustering analysis

In this result, cluster 2, shown in black, was a region in Baotou city with 18,357 people per square kilometers and 5,502 livestock per square kilometer. This resulted in the extreme value here that was clustered into its own cluster. By checking the box-plots chart, the density data for this small area should be treated as outlier. Hence, we merged this area with surrounding areas that also belonged to Baotou City, and treated them as a single county. As a result, the optimized clustering was 5 clusters based on merging cluster 2 into cluster 5. Strategies based on 8 and 10 clusters were also shown to have good performance under the pseudo F-statistic. Using a similar approach, outliers were removed from both the 8-clusters and 10-clusters result, transforming them into 7 and 8 clusters respectively.

Multivariate clustering methods could offer a strategy to cluster the whole study area with several candidates. However, these could not provide the best result directly to support further processing. In this study, to choose and verify which clustering strategy gave the best fit for gap filling, a discriminant analysis (Friedman, 1989) was performed on those three clustered results. In this study, 10,000 random sample points from IMAR were generated. Each point was assigned with values from the LULC classes, DEM, slope, precipitation and temperature. The population density and livestock density were calculated at county scale before being assigned to the sample points. Then, 75% of sample points were randomly sampled as the training dataset, and the remaining 25% were used as the test dataset for discriminant analysis. In this study, a quadratic discriminant analysis (QDA) was applied to avoid the problem where the covariance matrix is not the same for all (McLachlan, 2004). McLachlan also claimed the linear discriminant analysis (LDA) should be used cautiously because it is unlikely that homoscedasticity will hold exactly in real world modeling, even when the preliminary test does not reject the null hypothesis. After running the QDA, confusion matrices for all three scenarios were generated and the overall accuracy was calculated. The overall accuracy for 5 clusters, 7 clusters, and 8 clusters were 92.37%, 89.52% and 90.04% respectively. Hence, the 5-clusters strategy was selected to divide IMAR into 5 sub-regions.

3.4 Implementing Cloud-based Markov Chain - CA Model

The Markov model and CA model are both discrete dynamic models with time and states. When applying a Markov-CA integrated model in LULC change studies, the Markov model relies on a transition probability matrix of LULC change between

pairs of time steps for specific time intervals that are derived from observations. The probability matrix provides an estimate of the probability that each LULC class will be transformed over time to a different class or will remain in its current class. The CA model is used to add spatial features that complement the LULC information handled by the Markov model. By assigning a neighborhood filter and weights to each LULC cell, the integrated approach can be used to predict temporal and spatial changes of LULC for a study area (Fan et al., 2008).

3.4.1 Cloud-based Markov

Using Markov chain processing, the status of each cell at the same coordinates but from different input time steps was compared by overlaying two pair of input LULC datasets: 1) masked 2000 LULC data to masked 2010 LULC data and 2) masked 2010 LULC data to masked 2016 LULC data. The number of cells that convert from one LULC class to a different class was calculated and the transition probability matrix was generated using following equations(Fan et al., 2008):

$$P(X_{m+1} = j | X_m = i, X_{m-1} = i_{m-1}, \dots X_0 = i_0) = P(X_{m+1} = j | X_m = i)$$

$$P_{ij} = P(X_{m+1} = j | X_m = i), m = 0,1,2, \dots$$

Where X_m represented the random process and $m= 0,1,2$ represented cell status at two different time steps; P_{ij} was the transition probability calculated using the equation (Halmy et al., 2015):

$$P_{ij} = \frac{n_{ij}}{n_i}$$

Where n_{ij} represented the number of cells converted from class i to class j ; n_i represented the count of cells converted from class i to all other classes. Assuming there were n classes in this transition period, the values p_{ij} in transition matrix were:

$$0 \leq p_{ij} < 1 \text{ and } \sum_1^n p_{ij} = 1$$

Where i and j were integers ranging from 1 to n . The cloud-based algorithm was implemented by using Apache Spark on a cloud environment using a MapReduce function. (Figure 3-5).

Algorithm 1 Distributed probability matrix calculation algorithm

```

1: def Mapper (Key, Value)
2:   Foreach cell in LULC dataset:
3:     Emit (Cij, 1);
4:
5: def Reducer (Key, int[] Values)
6:   int SUM =0;
7:   Foreach count in values:
8:     SUM += count;
9:   Emit (Cij, SUMij);
10:  Pij = SUMij / SUMTotal
11:  Emit (SUMij, Pij)

```

Figure 3-5. Algorithm of distributed calculating probability matrix using Apache Spark

At Map stage, cells at the same coordinates from two different input LULC datasets were compared. C_{ij} represented the conversion from LULC class i to class j . Each pair was counted as 1. After this step, a list of all conversions was generated. At the Reduce stage, all 1s from all computing nodes were summarized by using C_{ij} as a key to calculate the SUM_{ij} , which represented the number of cells of each conversion pair. In other words, SUM_{ij} also represented the conversion area of each pair because the cell size was fixed. Then, the transition area matrix was generated and the transition probability matrix P_{ij} was calculated using SUM_{ij} divided by SUM_{Total} , returning the total number of input LULC cells.

3.4.2 Transition suitability

The transition matrix calculated by the Markov model provides a purely mathematical direction to simulate LULC change. However, many factors in the real world can affect the actual transition among LULC classes. One practical solution to optimize the simulated results was to apply suitability grid sets. Suitability grid sets contain the suitability values of each cell that may transition to a different LULC class. Suitability grid sets were calculated by the MCE method by assessing each involved constraint and restraint factor. The constraint suitability value was a Boolean type that contains values 0 and 1. Constraint suitability refers to the fact that a specific LULC class was not expected to change to another LULC class during the simulation. Those cells of unchangeable LULC regions were marked as 0 and the other locations in the study area were marked as 1, which represented the potential suitability for land cover change. For example, in IMAR, any type of grassland was highly unlikely to change into a water body over the course of a few decades. While over time, developed land use in an area might see an expansion over time but would be less likely to change back into say, a natural land cover. Hence, in this study, areas of water and human land use were considered as constraint factors.

The restraint suitability value were values in a normalized range calculated by effect of each restraint factor, that represented the potential suitability change level for a specific LULC class from *unsuitable* (small value) to *highly suitable* (large value). In this study, two constraint and nine restraint factors were involved (Table 3-2).

Table 3-2. Suitability factors

Factors	Definition	Function Shape	Control Point (s) / SB	
DEM	Elevation of whole IMAR (m)	MDJ	SR 1	690, 1143
			SR 2	387, 973
			SR 3	1024, 1672
			SR 4	1414, 1969
			SR 5	1316, 1852
SLOPE	Slope calculated by DEM (degree)	MDJ	SR 1	6.1, 20.5
			SR 2	3.9, 17.3
			SR 3	4.1, 14.9
			SR 4	11.1, 27.7
			SR 5	4.5, 15.8
DisRoad	Distance to roads	Linear	N/A	
DisRail	Distance to railways	Linear	N/A	
PPT	Yearly average precipitation (mm)	SS	SR 1	389.35
			SR 2	419.88
			SR 3	276.29
			SR 4	361.9
			SR 5	152.4
TEMP	Yearly average temperature (centigrade)	SS	SR 1	-1.58
			SR 2	5.97
			SR 3	3.28
			SR 4	6.5
			SR 5	8.38
POP	Population density per each county (people per km ²)	MDJ	SR 1	5.1, 37
			SR 2	37.8, 338.1
			SR 3	8.1, 194.5
			SR 4	52.0, 379.4
			SR 5	1.8, 57.5
LS	Livestock density per each county (livestock per km ²)	MDJ	SR 1	11.1, 74.9
			SR 2	115.9, 474.6

			SR 3	36.9, 240.6
			SR 4	116.4, 521.2
			SR 5	14.2, 138.9
CP	Compensation policy	Linear	0,1,2	
WA	Water area	Boolean	N/A	
LU	Human land use	Boolean	N/A	

Restraint factors affected LULC change with specific functions and control points. For example, DisRoad and DisRail represented the distance to roads and railways. The closer roads and railways were (short distances), the less chance there was that the original LULC classes would be converted into grassland classes. In this study, the effect of the distance to road and railways was defined as a linear function. Relating to animal grazing practices in the study area, the compensation policy included three practices: normal grazing (not participated in), balanced grazing, and forbidden grazing. This was a county-based policy that allowed herdsman to get compensation for balanced grazing or to halt grazing for specific time periods. Some counties in IMAR did not participate in this program. In this study, a linear function was defined for this factor by considering that with less grazing, there would be a higher chance that grasslands may recover from other types of LULC classes. Water areas (WA) and human land use areas (LU) were two constraint factors that were not expected to be changed into grassland classes.

Different function shapes were also applied to optimize the suitability value calculations. In this study, four types of functions were applied in MCE: Monotonically decreasing J-shaped function (MDJ), linear, symmetric sigmoidal (SS), and boolean. For example, in SR 1, a monotonically decreasing J-shaped

function was used to calculate suitability values for the DEM layer. Control points defined the position and accurate shape of the function curve in the Cartesian Coordinate System. Shape was used for cases such that, for example, the potential suitability of grassland was similar for areas with less than 690m elevation. Suitability gradually decreased however, as elevation increased from 690m to 1143m. Land was deemed unsuitable for grassland at elevations higher than 1143m. Another example was a symmetric sigmoidal function that was used to calculate the values for the yearly average temperature layer. To simplify calculations, we set -1.58 °C as the uppermost value for average yearly temperature according to the average yearly temperature from 2010 to 2016, which represented that the grassland suitability value was highest at -1.58 °C, and gradually decreased with lower or higher temperatures. After processing all eleven suitability factors, a suitability grid set was generated with assigned weights for each suitability grid. In this study, we applied equal weights to all eleven factors to simplify the modeling and computing.

3.4.3 Cloud-based CA modeling

CA modeling was developed to represent complex scenarios with sets of transition rules over a cell-based pattern. (Pinto and Antunes 2007, Han and Cao 2005). A traditional CA was formed by lattice space, cells, cell states, neighborhood scheme, and transition rules. Von Neumann and Moore are two common neighborhood schemes that are widely used in CA simulators. In this study, a CA with classic 5 x 5 extended Von Neumann neighborhood scheme was applied to undertake three tasks: 1) neighbourhood filtering that assigned different weights to target cells within the designated neighbourhood scheme; 2) computing a final

decision for the transition value for each cell based on the transition probability matrix, transition area matrix and suitability grid sets; and 3) iterations that defined time intervals within given time periods based on the input LULC datasets used in the simulation process.

However, to process a large-scale CA model (i.e., beyond computational capacity of processing on single workstation) efficiently is a challenge. Some researchers have already explored this using big data computing frameworks and high-performance computing resources. For example, Radenski (Radenski, 2013) tested such an algorithm on Amazon's Elastic MR Cloud with a maximum 1.6×10^8 cells in a 2D situation where they employed 1 master node and 16 core nodes in this simulation. Marques and colleagues (Marques et al., 2013) developed a new computational framework based on MapReduce to implement a 1 trillion cell 2D/3D CA simulation on the Microsoft Azure cloud platform. Those large CA approaches show the applicability of CA to big data/big model computing, but also leave room for other investigators to achieve improved solutions with simpler modeling, faster computing and easier data access.

Our approach tackled this problem using a graph-based cellular automata implementation. We considered the graph as a highly suitable framework for big data/big model development as the CA model contains cells and their neighbors. Cells can be modeled by the concept of a vertex, and neighbor relationships can be reflected in an edge pattern. By evaluating the state of the vertices connected with the target vertex, users can easily set up the transition rules to update the state of the target vertex. More importantly, with graph-based CAs, we can make use of suitable

and powerful large-scale graph processing infrastructure such as Apache Giraph in order to achieve massive cell volume and high performance with the CA simulation. Those infrastructures can be usually deployed on cloud platforms such as Amazon Web Services (AWS) (Amazon, 2015), Microsoft Azure (Wilder, 2012), as well as Google’s Compute Engine (Sanderson, 2009), and all of them can offer potentially huge computing resources to support the research processing.

The algorithm to implement cloud-based CA used Apache Giraph, a computational framework based on the Bulk Synchronous Parallel (BSP) model that provides a sound platform for large-scale graph computing (Figure 3-6).

Algorithm 2 Cloud-based CA implemented with Giraph

```

1: class GiraphCA
2:   MaxSuperStep  $\leftarrow$  user defined value
3:   function compute (vertex, messages):
4:     if getSuperstep() == 0 then
5:       sendMessage to all neighbors, assign weight,
6:       compute ( $E_{ij}$  = neighbor weight  $\times$  suitability value)
7:       if getSuperstep() <= MaxSuperStep then
8:         Foreach cell in study area:
9:           While(true):
10:            if  $\max(E_{ij}) >$  area limit then
11:               $\max(E_{ij}) = 0$ 
12:            else
13:              convert i to j;
14:              break;
15:            end if
16:          else
17:            voteToHalt()
18:          end if
19:        end if

```

Figure 3-6. Algorithm of distributed processing Markov CA model with Apache Giraph

This framework contains many supersteps (in terms of Apache Giraph, this represents the timesteps in traditional CA terms) where, in the operation process of each superstep, each computing unit is arranged into a certain number of vertices or edges that ensures parallel computing; each computing unit communicates with others

through message interaction; and when the processing of this unit reaches a barrier, it stops until other cores complete their message interaction. After executing compute-functions in code for certain supersteps or certain halt conditions, users can save the output back to DFS. The message-passing and barrier features are very useful in implementing CA approaches that can support CA cell status changes based on neighbors and updates at each time step. In this study, the neighborhoods of each target cell were calculated at superstep 0 by sending messages with designated edge patterns. Then, neighborhood weighted values were calculated and integrated with the results from MCE. The overall evaluation value list of each cell in the study area (E_{ij}) was captured because each cell could possibly transition from one value/class to another. Once superstep 1 began, Giraph CA started to find the highest transition possibility for each cell by calculating the maximum value in the overall evaluation value list. At the same time, the value was required to not exceed the corresponding value in the area transition matrix to avoid over estimation. After all cell statuses were updated, the job was halted and results of simulated cell values for study area were output.

4. Results and Discussion

4.1 Experiment Environment

Using the same local computational environment as we used in Chapter 2, the local cloud environment tests in this research involved 1008 computing vcores and 5.12 Tb memory available. A 2 Pb HDFS was configured and the block size was 128 M as default (Table 3-3).

Table 3-3. Computing environment

Hardware				Software	
Role	Count	CPU	RAM	Name	Version
Master Node	2	2 x Intel Xeon E5-2680v4 2.4GHz	256 G	Apache Spark	2.2.0
				Apache Giraph	1.2.0
Computing Node	18	2 x Intel Xeon E5-2690v4 2.6GHz	256 G	Centos	6.9
				Java Server VM	1.8.0_152 64Bit
Network	10 Gbps			Cloudera	5.12.0

The approach used in this research was based on YARN, which has been widely used in many current cloud environments (X. Fan et al., 2017).

4.2 Simulation Results and Accuracy Assessment

For this analysis, we used masked LULC maps from 2000, 2010 and 2016 for IMAR as input training datasets to simulate and fill gaps that were present in the 2016 dataset. With the computational capability of our cloud-based Markov-CA simulator, we simulated the whole IMAR region with a single run. After completion, the simulated data for gap areas was mosaiced with existing data on the 2016 LULC map to generate a final estimated 2016 LULC map. Eleven impact factors from the perspective of biophysical, ground features and human-related factors were analyzed using MCE to build the suitability grids for the model.

Table 3-4. Transition matrix

	SR	TMS	TS	TDS	TSD	TD	LM	MM	MA	WA	NG
TMS	SR1	71.87	8.03	N/A	N/A	N/A	2.12	0.34	0.09	0	17.54
	SR2	66.09	0	N/A	N/A	N/A	2.33	0.87	0.02	0.07	30.62
	SR3	58.59	22.35	0	0	N/A	3.01	0.53	0	0.17	15.36
	SR4	9.66	48.6	0	N/A	N/A	0	0	0	0.5	41.24
	SR5	16.91	76.58	0	0.35	0	0	0	0	0	6.16
TS	SR1	9.97	79.16	N/A	N/A	N/A	10.7	0	0.09	0.08	0
	SR2	0	65.43	N/A	N/A	N/A	3.75	0	0	0.26	30.57
	SR3	10.34	34.57	22.63	0	N/A	14.66	0.01	0	0.4	17.39

	SR4	0.24	60.14	0.03	N/A	N/A	0	0	0	2.11	37.48
	SR5	0	73.37	11.71	0	0	0.27	0	0	0.98	13.66
TDS	SR1	N/A									
	SR2	N/A									
	SR3	0	22.76	67.73	7.03	N/A	2.48	0	0	0	0
	SR4	0	10.65	86.34	N/A	N/A	0	0	0	0.32	2.69
	SR5	0	28.95	42.62	15.28	1.92	3.71	0.01	0	0.24	7.26
TSD	SR1	N/A									
	SR2	N/A									
	SR3	0.15	0	33.17	63.43	N/A	3	0	0	0	0.23
	SR4	N/A									
	SR5	0	0	21.89	52.12	25.63	0	0.04	0	0.32	0
TD	SR1	N/A									
	SR2	N/A									
	SR3	N/A									
	SR4	N/A									
	SR5	0	0	0	22.05	31.87	11.45	0	0.58	0	34.05
LM	SR1	1.01	4.51	N/A	N/A	N/A	54.59	1.75	1.18	2.31	34.65
	SR2	1.87	9.23	N/A	N/A	N/A	44.93	0	0.25	5.28	38.44
	SR3	6.62	29.2	2.13	2.46	N/A	50.11	0	0.1	3.24	6.14
	SR4	0	6.79	0	N/A	N/A	17.99	0	0	11.03	64.19
	SR5	0	7.42	16.44	4.41	11.44	30.38	0	0.21	5.99	23.71
MM	SR1	22.28	0	N/A	N/A	N/A	3.97	55.88	0	0	17.87
	SR2	41.08	0	N/A	N/A	N/A	2.83	31.27	0	0	24.82
	SR3	64.1	0	0.6	0.06	N/A	0.23	17.77	0.01	0	17.22
	SR4	27.8	0.82	0	N/A	N/A	0.03	69.52	0	0	1.84
	SR5	0	0	0	0	0	0	74.55	0	0	25.45
MA	SR1	2.48	3.2	N/A	N/A	N/A	47.45	0	43.31	3.56	0
	SR2	6.31	0.35	N/A	N/A	N/A	37.12	0	20.75	0	35.47
	SR3	2.83	0	0	0	N/A	96.91	0	0	0.26	0
	SR4	0	0	0	N/A	N/A	56.7	0	0	41.88	1.42
	SR5	0	0	0	0.8	0	0	0	35.23	59.18	4.79
WA	SR1	0	2.22	N/A	N/A	N/A	19.72	0	0.65	77.41	0
	SR2	0	0	N/A	N/A	N/A	22.13	0	1.15	44.55	32.17
	SR3	0	0.93	0	0	N/A	33.11	0	0.12	65.83	0
	SR4	0	0	0	N/A	N/A	37.8	0	0	47.51	14.68
	SR5	0	1.51	4.13	0	45.03	24.73	0	0.87	23.74	0
NG	SR1	9.9	0	N/A	N/A	N/A	32.12	4.54	0	0.71	52.73
	SR2	18.12	16.43	N/A	N/A	N/A	3.96	0.19	0	1.5	59.81
	SR3	18.07	29.78	0	0	N/A	0.59	0.66	0.02	1.33	49.55
	SR4	0.19	14.34	0.01	N/A	N/A	1.38	0.01	0.13	1.79	82.14
	SR5	0.11	13	12.32	0.34	31.96	6.56	0.05	0	2.41	33.25

The transition matrix for the ten IMAR LULC classes was calculated (Table 3-4).

In this table, the diagonal values for each SR represented the probability that certain LULC classes did not change during the training period. Other values in this table show the probability that certain LULC classes were changed to other types of LULC

classes. The N/A value in this table represented the case where there was no such class in this SR. During the two paired training periods 2000 to 2010, and 2010 to 2016, the most stable LULC classes were water areas (WA) and non-grassland areas (NG). They were considered as Boolean types of constraint factors, which were expected to be fixed and not change during the simulation. Other types of LULC classes can change into those two classes given a certain probability. However, in reality, it was quite rare that grassland classes changed into water areas. On other hand, rapid increases in human activities resulted in grassland classes being changed into NG classes, especially to farmland and human land use (LU). For example, cities and roads were expanding rapidly in IMAR. Overgrazing also happened in some areas during the past few decades. These cases may lead to grassland degradation or even the disappearance of grassland especially in some sensitive areas. By assessing this transition matrix, we found that Temperate Meadow-steppe (TMS), temperate steppe (TS), and Lowland Meadow (LM) were changed into NG with a relatively high probability (typically over 30%) especially in SR 2 and SR 4, which were high population density and high livestock density regions respectively. We also found that some transitions happened between neighboring types of grassland classes. For example, TMS class showed a probability of approximately 76.58% to change into TS in SR 5, which is a very dry area. In other words, TMS class areas that require relatively high moisture environments may be less likely to be stay as dry areas over time.

The transition matrix represented the analysis results from a mathematical perspective. During the simulation process, the results that were closer to an actual

change mechanism were generated by integrating multi-types of impact factors, weighting each of them with different functions and assigning different neighborhood weights.

The actual (observed) classified LULC result of 2016 and the simulated result of 2016 generated by cloud-based Markov-CA were mapped (Figure 3-7). We found the simulated results of three major grassland types in IMAR Meadow, Steppe and Desert grassland were very close to their distribution of the actual LULC map. Especially the TMS at northeastern part of IMAR, and TDS and TD in western part of IMAR shows high similarity based on visual assessment. The Greater Khingan forest, located in the northeast corner of IMAR, was classified into NG class in our study, and the simulated results were also visually similar to the actual LULC case. Three water bodies, Hulun lake, Dalinur lake and Ulansu Lake were clearly represented. Major cities, major roads and other types of human land use were also found to be visually similar. However, some issues were found with the simulated results. In following discussion, we discuss three areas with the most cloud coverage (and the most gaps to fill) to explore further.

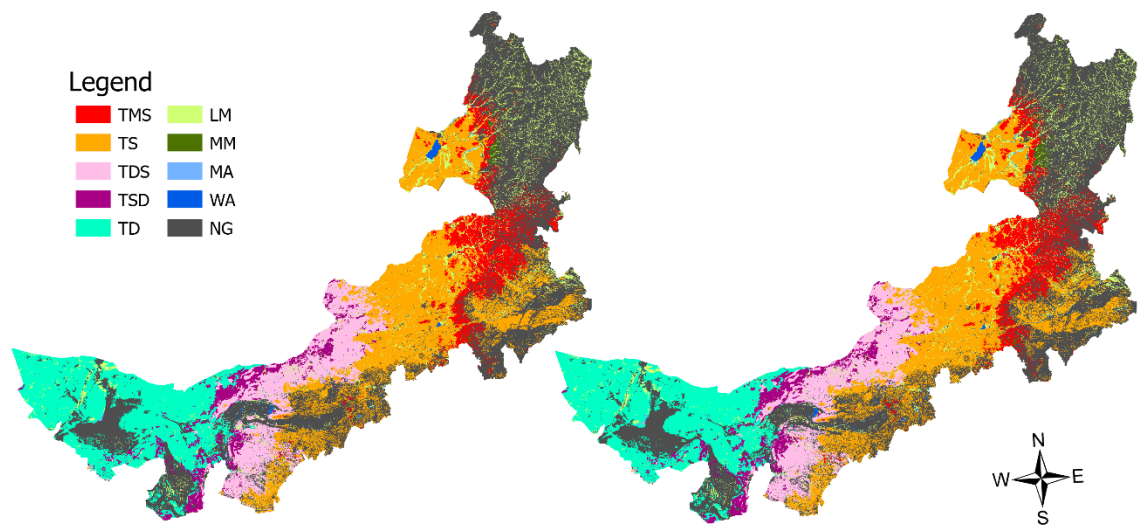


Figure 3-7. Overall results. (a) actual LULC mapping for 2016 and (b) LULC mapping with simulated gap filling for 2016

The overall accuracy assessment results were calculated using two commonly used accuracy assessment methods, a confusion matrix and Kappa statistic (Congalton & Green, 2002). The accuracy assessment results for the model processed without using sub-region strategy (Table 3-5) and using sub-region strategy (Table 3-6) were listed for comparison. By calculating the confusion matrix and Kappa based on 20,000 sample points generated randomly across IMAR, the overall accuracy was found to be 84.11% and the Kappa statistic was 0.79 for the model processed without using sub-region strategy. However, the model processed with sub-region strategy offered 4.05% improvement in overall accuracy and 0.06 Kappa improvement. The major accuracy improved LULC classes are TMS, TS, TDS and MM for the reason that sub-region strategy allowed tailored control process on MCE hence calibrated each sub-model with better fit impact mechanism for each LULC class in different sub-region. These results suggested that the accuracy of the simulated results were

relatively high and the consistency was also high (Viera & Garrett, 2005). The results presented in Table 3-6 show that the simulated accuracy of grassland classes was higher than other types of LULC classes. For example, the producer's accuracy and user's accuracy of TS class were both over 90%. However, the accuracy of Marsh (MA) and WA classes were relatively low (around 30%). One possible reason for this was that the simulated process was lacking supporting evidence, data and mechanism for water body changes, which resulted in water branches near major water bodies not being simulated as successfully. The issue happened most obviously in the Ulansu Lake water area (Figure 3-8(E) and 3-8(F)).

Table 3-5. Overall accuracy assessment without using sub-region strategy

	TMS	TS	TDS	TSD	TD	LM	MM	MA	WA	NG
Producer's Accuracy	82.06	85.22	83.82	87.50	84.50	81.32	81.82	93.75	67.65	84.35
User's Accuracy	84.52	80.67	89.06	82.60	97.00	74.20	80.36	26.79	30.40	85.41
Overall Accuracy	84.11									
Kappa	0.79									

Table 3-6. Overall accuracy assessment by using sub-region strategy

	TMS	TS	TDS	TSD	TD	LM	MM	MA	WA	NG
Producer's Accuracy	87.20	91.88	89.71	89.23	89.51	84.14	86.67	94.12	59.76	85.93
User's Accuracy	89.04	89.13	93.20	84.89	98.05	77.79	85.53	28.07	23.67	86.89
Overall Accuracy	88.16									
Kappa	0.85									

The simulated accuracy for five sub-regions were calculated using confusion matrix and Kappa statistic (Table 3-7). In those five sub-regions, the major grassland classes including TMS in sub-region 1, 3 and 4, TS in sub-region 1,3 and 5 and

TSD/TD in sub-region 3 and 5 were simulated with a high level of accuracy. Especially for TS in sub-region 1 and TDS in sub-region 3 and 5, the accuracy of them were over 99%. In sub-region 2, the accuracy of grassland was not as high (typically around 80%) as them in other sub-region due to the rapid growth of cities and farmland. The Kappa was 0.71 for sub-region 2 and 0.6 for sub-region 4, which represented a substantial level of consistency.

Table 3-7. Sub-regions accuracy assessment

		TMS	TS	TDS	TSD	TD	LM	MM	MA	WA	NG
SR 1	Producer's Accuracy	85.61	98.81	N/A	N/A	N/A	85.71	82.76	100	55.00	96.47
	User's Accuracy	93.70	99.10	N/A	N/A	N/A	87.64	96.00	100	100	93.57
	Overall Accuracy	93.82									
	Kappa	0.89									
SR 2	Producer's Accuracy	81.30	74.88	N/A	N/A	N/A	70.87	100	100	31.25	88.41
	User's Accuracy	81.63	80.26	N/A	N/A	N/A	76.84	80.0	90.0	41.67	85.74
	Overall Accuracy	83.50									
	Kappa	0.71									
SR 3	Producer's Accuracy	93.71	93.55	98.66	95.74	N/A	78.88	100	100	54.55	68.59
	User's Accuracy	90.24	94.44	99.66	100	N/A	92.56	100	90.0	54.55	65.52
	Overall Accuracy	89.78									
	Kappa	0.86									
SR 4	Producer's Accuracy	90.91	71.54	100	N/A	N/A	30.77	90.91	100	15.0	87.34
	User's Accuracy	100	69.96	90.0	N/A	N/A	80.0	100	100	30.0	84.78
	Overall Accuracy	80.84									
	Kappa	0.60									
SR 5	Producer's Accuracy	100	72.12	95.42	95.43	98.39	79.17	100	90.91	40.0	78.90
	User's Accuracy	20.0	84.40	88.56	99.40	95.10	75.0	70.0	100	54.55	83.88
	Overall Accuracy	89.74									
	Kappa	0.86									

The RS images were commonly affected by cloud and haze especially for the eastern and northern parts of IMAR, due to relatively high levels of moisture being present. To further assess the gap filling results for IMAR, we focused and zoomed to three heavy cloud coverage subareas with large numbers of gaps (over 85%) that were needing to be filled in the Hinggan League region in eastern IMAR, the Xilingol League region in the central part of the region, and the Bayan Nur region in the west, where each subarea covers ~100,000 square kilometers (Figure 3-8). Another reason for choosing those three areas was because they were covered by three major grassland types: Meadow grassland in Hinggan League area, Steppe grassland in Xilingol League area and Desert grassland in Bayan Nur area, which could better represent the gap filling performance for major grasslands in IMAR.

The first grassland subarea represented the Hinggan region (Figure 3-8a) with the actual LULC map of this area for 2016 (Figure 3-8b). In this area, TMS and TS were two major grassland classes. The simulated results were similar to the actual surveyed grassland distribution in this area. However, the TS class was represented as over simulation to some extent, which may result from the weight assignment and function control points setting of precipitation factors during the construction of the suitability grids. The spatial resolution for precipitation data (1km) were lower than the spatial resolution of LULC mapping (30m). Hence the control points for precipitation were generated by a statistical method, which might not have been able to capture the difference of precipitation in each specific area of this subarea. TS grassland might be sensitive to this environmental factor, which may result in the

simulated TS grassland class expanding faster in this area than it actually did (i.e., it is over simulated in the final result).

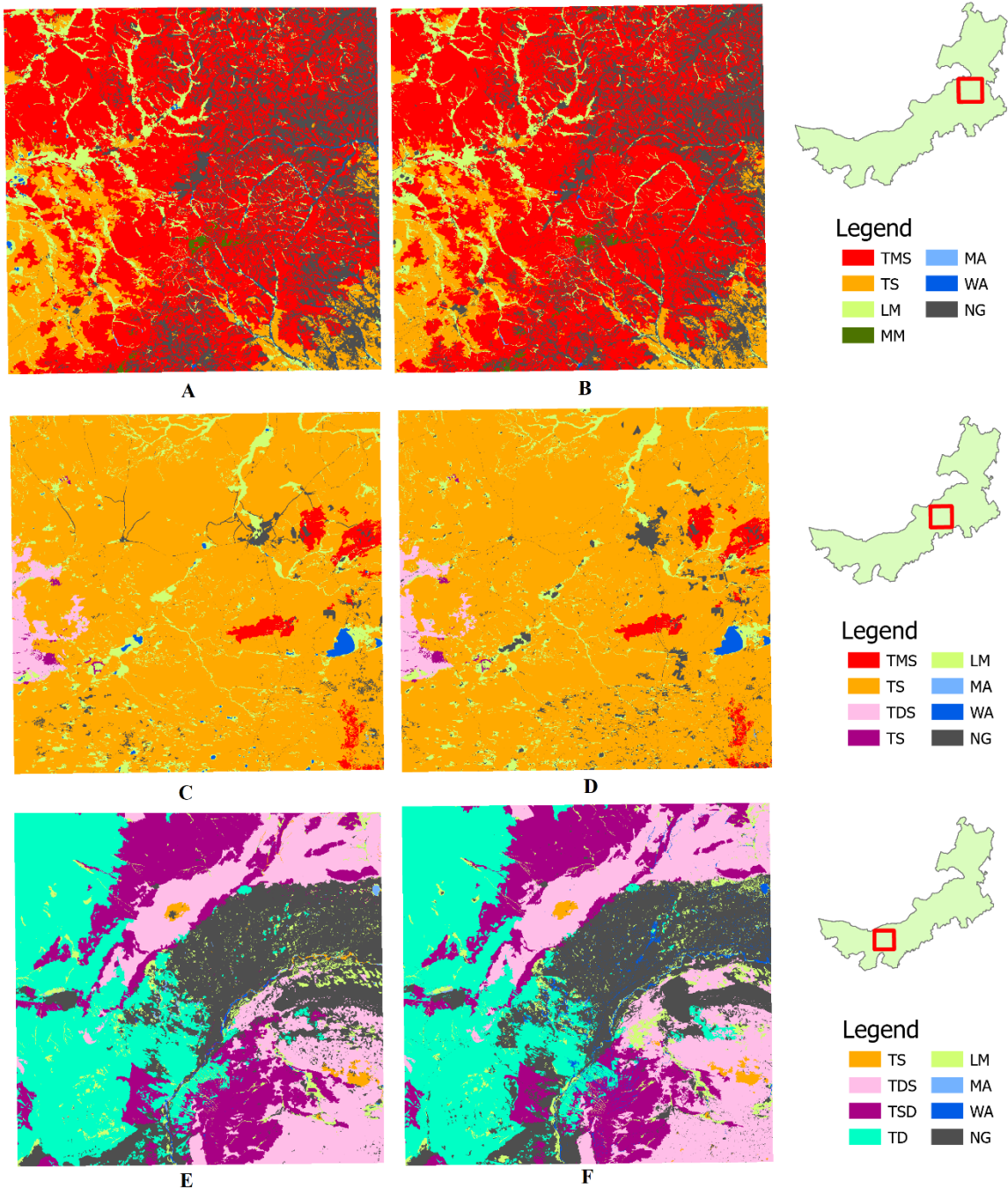


Figure 3-8. LULC maps of three grassland areas: (A) simulated Hinggan League area (meadow grassland), (B) real Hinggan League area, (C) simulated Xilingol League area (steppe grassland), (D) real Xilingol League area, (E) simulated Bayan Nur area (desert grassland), and (F) real Bayan Nur area

There was a lack of detailed simulation for the water body class that was also notable in this area. We found more WA class pixels existing as shown in Figure 3-8a, but these pixels were changed into NG or other types as shown in Figure 3-8b. After checking the training data, the water body classes that were shown in simulated results were existing in the 2010 LULC map. However, they were not classified as WA again in 2016 due to drying up or a lack of water because they were small lakes or rivers that were vulnerable to environmental change. The same thing happened in Bayan Nur region where the simulated results for Bayan Nur showed the WA class was less represented (Figure 3-8e) than it was shown in the actual LULC map of the same subarea (Figure 3-8f).

Human land use in IMAR was found to have a tendency to be under simulated. This was found among the three subareas especially in the Xilingol League area where the simulated Xilingol League region showed more NG (Figure 3-8c) while the actual LULC map showed less NG (Figure 3-8d) in this area. For Xilingol, the major grassland classes were accurately simulated except for parts that were occupied by human land use such as expanded cities and newly developed farmland. For example, the continuous NG area in the central part of Xilingol League was the Xilinhot City. It was shown that the simulated result of Xilinhot City was smaller than actual coverage. Also, some rectangles like NG area were shown in the southern part of this area, which were new farmland developed in the past few years, and were not successfully simulated in this study. This was mainly due to the fact that although socio-economic impact factors such as human density, livestock density and compensation policy were integrated into the simulation process, further research on

actual impact mechanisms that show how those factors affect a population's land use and the best weight for each factor will be required. For this reason, simulating human land use especially farmland was not as accurately simulated in IMAR as other land use types.

5. Conclusions

This study presented a one-stop solution that was able to implement a cloud-based LULC data gap filling framework using a distributed Markov Chain CA model. We designed and implemented a Spark-based in-memory distributed Markov Chain and suitability processing algorithm on the cloud, and integrated it with a self-designed Giraph-based distributed Cellular Automata to fill LULC gaps even for a very large area. The advantages of this framework were: 1) it can be easily integrated with existing cloud-based RS imagery processing tools. The output of those image processing tools can output the LULC dataset directly to the distributed file system on the cloud as input for our LULC gap filling tool without any additional data transfer; 2) It can access existing LULC and auxiliary data sets in a cloud data warehouse without downloading data to local machines.; 3) it can exploit a large scale cloud-based computing framework to accelerate the processing speed; and 4) like all other cloud-based frameworks, it is able to gain unlimited computing resource in theory to support very large processing tasks through commercial cloud computing platforms.

In this study, we successfully tested LULC gap filling processing for the entire IMAR region with a cloud-based Markov Chain CA framework. Masked LULC maps from 2000, 2010 and 2016 were applied as an input training dataset with 11 different constraint and restraint factors involved in the MCE. After running this framework on

a local cloud environment, we successfully filled the gaps resulting from cloud coverage in the 2016 LULC map using simulation, and generate a complete LULC map for 2016. We also ran accuracy assessments that returned an overall 88.16% accuracy.

Further research is needed to improve the accuracy and performance of this framework. For example, in IMAR, water bodies and human land use were not simulated as accurately due to a lack of supporting data and studies on the impact mechanisms for these LULC classes. Also, weight assignments and function design require further study and need to be elaborated for each LULC class to bring the final results closer to the actual values. In the future, Analytic Hierarchy Process, a commonly used approach to assess the weight for each impact factor will be integrated into this framework. The neighborhood scheme could also be adjusted and tested. In this study, the classic 5 x 5 extended Von Neumann neighborhood scheme was applied, however, different neighborhoods could also be used along with an accompanying re-design of the computing partitioning of the Graph-based CA to ensure the best computing performance.

Chapter 4 : Massive Voxel Cellular Automata Using Giraph: A Use Case of Air Pollutant Particles Dispersal

1. Introduction

Cellular automata(CA) was originally conceived to support computing needed for complex scenarios (Von Neumann, 1951). Cells, acting as an automaton (A), were entrusted as representative media for housing states (S), and transition-rule machines (f) were tasked with determining changes to those states over time ($t \rightarrow t+1$), based on input garnered from the cell itself and from neighboring cell-states (N). When arranged in a lattice or matrix space (L) of dimension d, CAs can collectively be used to represent a variety of phenomena, as well as structures and the processes that determine their dynamics (Wolfram, 2018), principally by trading the unique local value of state information within the systematic context of the larger lattice.

$$A = (L, d, S, N, f), f : S_t \rightarrow S_{t+1}$$

In 1970, John Conway designed a computer game named “Game of Life” (GoL) (Conway, 1970) that became the best-known CA and attracted the attention of scientists. It has been used as a standard testbed for performance comparisons among different CA approaches. Following on this work, Stephen Wolfram examined large number of possible classifications of automata in the 1980s. He published the seminal book, *A New Kind of Science* (Wolfram, 2002b) and introduced four classes (Stable, Period, Chaotic and Complexity) by which cellular automata and several other simple computational models could be categorized depending on their behaviors.

CAs are popularly used as computational media for modeling and simulation of complex systems in varied scientific arenas. Much of the appeal of CAs lies in their potential to support huge numbers of interacting components, which has made them the medium for addressing unwieldy complex systems in simulation. While they are theoretically scalable to fantastic sizes, the vast majority of CA models have been built to support applied simulations. For example, CAs have been applied to solve problems in biology (Ermentrout & Edelstein-Keshet, 1993), chemistry (Mendes et al., 2001), medicine (S. H. White, Del Rey, & Sánchez, 2007), physics (Zhao, Billings, & Coca, 2009), astronomy (Isliker, Anastasiadis, Vassiliadis, & Vlahos, 1998) and economics (Qiu et al., 2007) among other areas.

CAs have also been used for geographical domains. For example, the "sand pile CA" was a simple 2-D CA that represented a pile of sand grains. This idea emanated from physicist Per Bak and his colleagues in 1987 and used CAs to explore complexity in physical phenomena (Bak, Tang, & Wiesenfeld, 1987). Following Bak *et al.*'s study, researchers like Dattilo and Spezzano (Dattilo & Spezzano, 2003) and Iovine *et al.* (Iovine, D'Ambrosio, & Di Gregorio, 2005) pushed CA modeling forward in their studies that used CAs to study for example, mass movement of Curti–Sarno debris flow in Southern Italy. In recent years, Li and his colleagues applied CAs to simulate the flood of Hunhe River, China using dynamic observations (Y. Li et al., 2015). Dahal and Chow simulated urban growth in San Marcos, Texas using an irregular CA model (Dahal & Chow, 2015).

Until now, most of the CA models in geographical sciences have been built in two-dimensional (2D) space. However, many environmental processes operate in

three-dimensional (3D) space including time. To model these complex systems, 3D spatial contexts need to be supported in order to offer enough details to analyze the dynamics of these systems in all three dimensions. However, modeling geographical applications with 3D CAs is a significant research challenge because it will require the sophisticated re-design of each part in the CA to successfully support the simulation process. In other words, the modeling and computational complexity will be highly increased when extending 2D CA models into 3D CAs. In a few studies of 3D CA in geography, researchers have developed voxel CAs for geographical processes, such as dune movement and formation (Narteau, Zhang, Rozier, & Claudin, 2009), snow avalanche structures interaction (Avolio, Errera, Lupiano, Mazzanti, & Di Gregorio, 2017; Dahal & Chow, 2015), and wind field estimation (Salcido & Celada, 2010). However, these 3D models were all limited with respect to the number of cells (less than 100,000 voxels) and either applied small scale study areas or used coarse spatial resolution in their experiments. And even when a CA model has the potential to support huge numbers of interacting components in a simulation, most existing 2D and 3D CA models are run on a single desktop computer. Limited by the computing capacity of a single machine, researchers are constrained in the detail and intricacy with which they can model phenomena relative to the richness and complexity that they observe in the real world, especially in 3D space. Moreover, the range of questions that can be posed in simulation are often constrained by excessive computing times for single-CPU CA, which means that the native ability of CA to sweep through the parameter-space of complex what-if problems is often under-utilized in these applied studies. With the arrival of the big

data era, those issues are becoming more prominent as CAs have the potential to serve as a big model due to being highly scalable, and their ability to consume big data as input and convert the data into meaningful results, helping to answer scientific questions. However, big data and big models inevitably also lead to big computational burdens in real life processing. Even by applying CAs to model complex, 3D scenarios, the volume of the data that needs to be processed may not be significantly reduced (in some cases, it could increase). Hence, the requirement of improving the capability of CA, especially for 3D modeling, and to fit large scale simulation is becoming increasingly important.

As computing hardware and big data silos have progressed, many in the CA modeling community have been slow to adopt technologies that could significantly broaden the reach of their CA models. Use of high-performance computing (HPC) clusters and cloud computing platforms have been taking hold in applied CA modeling. Compared to traditional supercomputer and HPC computing clusters, cloud computing resources offer a low access threshold for many researchers, that could help to greatly reduce the cost of large-scale scientific computing. Indeed, a number of supportive software frameworks are already well developed and available for use. For example, Apache Hadoop has been widely leveraged in a diverse range of applications to reduce the developmental overhead in parallelizing computing (Shvachko, Kuang, Radia, & Chansler, 2010). Related schemes based on Apache Giraph and Spark are also garnering a popular following in scientific research (Han & Daudjee, 2015).

In this study, we introduce a method that uses a graph-based algorithm to implement a very large (up to 1 trillion cells in our experiments) voxel CA. To accomplish this, we fully re-think and rebuild the voxel CA with respect to modeling, computational implementation, and computing resource access. We rely on Apache Giraph, a Bulk Synchronous Parallel (BSP) model-based framework designed for iterative large-scale graphics processing, which affords users a platform to build and implement large graphs to support massive-scale voxel CAs with efficiency. To demonstrate the value of this approach, we discuss two test simulations: one as a 1 trillion-cell 3D GoL CA, and another as a 1 billion-cell CA-based simulation of air pollutant particle dispersal. We will demonstrate how our graph processing approach was successful for gaining significant processing efficiency, with particular performance gains over different cloud platforms.

2. Big Voxel CA Implementation

Although the CA model is widely regarded as a powerful tool and can be used to simulate any complex phenomena in real life by applying simple transition rules, it is not easy to design and implement a suitable CA model to accurately reproduce phenomena in the real world, especially using a 3D spatial perspective. At issue here, first and foremost, are some complications involving how voxel CAs are mapped to geographical systems in simulation. This has been discussed with respect to properly designing neighborhood relationships and cell shape is a challenge to implement big CA with big data (Fonstad, 2006). These two aspects of CA design are critical in HPC in particular, because they determine the exchange of information across the lattice, and govern what is computed and updated, where, and when. For example,

most 2D CA are formed on a square-grid lattice, while 3D cellular automata are based on a cube grid. However, in most practical situations, the grid cell shape departs significantly from the actionable shapes relevant to the applied problem that the CA is tasked with modeling. This creates what is called a Modifiable Areal Unit Problem (MAUP) in geography (Openshaw, 1984). Flexible schemes for handling cell shapes may be required, particularly if different scales of interaction and mixed-mode CA are desired in the same simulation. For example, Frisch and colleagues pointed out that "lattice-gas" CA models usually need hexagonal cell-shapes (Frisch, Hasslacher, & Pomeau, 1986).

Furthermore, transition rules can be designed in two different ways: a top-down approach makes the CA transition rules change during processing based on specific phenomena in different scenarios, while piecewise transition rules may lead to more complex rule sets, but also result in difficulties in integrating these rules in the simulation (Gobron, Çöltekin, Bonafos, & Thalmann, 2011). Additionally, the cell states of a discrete CA such as the GoL contains only 2 states, dead or alive. However, the cell states of continuous CAs could be represented using float values from 0 to 1 that enable the transition rules to be much more flexible in appropriate cases.

In addition to neighborhood scheme and transition-rule design, another critical issue in implementing a voxel CA model that fits a big model, is the computational design: CAs are, at their heart, computers. With the exception of some lattice-gas CA models, most applied CA models are built on raster-grid based computer graphics processing schemes using a single processor, with a serial strategy for updating states

according to transition rules. Yet, CAs are inherently suited to parallel processing on multi-core, multi-processor, or multi-computing node platforms. For some large-size CA simulations, such as those developed by Crave and Davy (Crave & Davy, 2001), researchers divide the whole computing process into many time-steps or time-slices instead of developing a real-time simulator. To conquer this problem, many researchers are seeking different solutions to develop CA models that can handle big models and also, simplify the coding and deploy complexity.

Researchers have demonstrated significant success in implementing CA models with HPC algorithms and cloud computing for simple CAs. Conway's GoL (Conway, 1970), in both 2D and 3D form, is a standard for performance testing in this regard, because it is simply specified but yields significant dynamics, including Wolfram Class 4 complexity (Wolfram, 2002a).

2.1 HPC Solutions

A lot of attention has been paid to general-purpose processing using a graphics processor unit (GPGPU) on local high performance workstations (Gobron et al., 2011). As one of commonly used HPC frameworks, GPGPU platform provides a large number of cores (cost-effectively) for CA computing within the single desktop model that is commonly used in applied modeling as compared to the normal CPU environment. Specifically, GPUs can handle more split tasks in a certain time period than many CPUs. Using this idea, We have built a GPGPU-based voxel GoL CA and tested it on a single workstation (with two NVIDIA Quadro 5000 cards providing 352 cores) and we were able to reach 10 million-cell simulation in real-time for a voxel implementation of the GoL. We abutted hard memory limitations at lattice sizes

beyond that, however, and this would appear to be a generic limitation for GPGPU approaches.

Message Passing Interface (MPI) is another commonly used HPC approach. For the GoL, for example, researchers have shown that MPI can speed-up transition calculations significantly, e.g., for a 1000 x 1000 lattice, a processing time of 3.365 seconds per 100 time steps can be achieved (Weeden, 2013). MPI has some limitations, however. First, MPI is focused on memory efficiency: this eats away at the limits of GPGPUs that we mentioned, but it is at odds with data-intensive modeling tasks. In GIScience applications, for example, CA models must often ingest huge amounts of sensor data, at initialization as well as during run-time, and these must often be reconciled, queried, and analyzed during computing. Researchers have also had difficulty implementing MPI on CAs in some cases. For example, (Gropp & Lusk, 2007) implemented a 2D CA model with MPI-2 but had to create a buffer for boundary cells, divide the whole grid into submatrices, and run them in parallel. This workaround is less extensible to the scenarios that we are requiring: the relationship among boundary cells in voxel-space is much more complicated than in 2D, and this can lead to many challenges in dealing with the boundary cell buffer and partitioning of the lattice, particularly causing problems with resource balancing. However, there is some implied potential in using dedicated big data management schemes, such as Hadoop and Spark, to handle this data-wrangling outside or alongside MPI.

2.2 Cloud Solutions

Use of cloud-based clusters of computers is another possible solution to support massive voxel CAs, offering particular advantages for running distributed programs

atop them. Key here is the potential for cloud resources to allocate and share CA software and hardware resources on-demand. CAs could support distributed partitioning because of the way that transition-through neighborhoods work on a local-to-global scale within CA lattices. Cloud resources are well-suited to chunking large computational tasks into many small parallel tasks and assigning them to computing instances, offering both efficiency and load-balancing. After all nodes have completed the assigned tasks, results are aggregated and returned to the users' local database, and here the network can infuse additional efficiency in reconciling updates between partitioned jobs. In this way, cloud distribution can reach millions (or even billions) of instructions in a few seconds. Indeed, MapReduce and Hadoop have emerged to provide this functionality with considerable efficiency (T. White, 2012). Radenski introduced a cloud-based GoL implementation using MapReduce Streaming (MRS), tested for both distributed and continuous simulation (Radenski, 2013). He made use of single and multiple text files as the state information for cells: each line in the text was used to represent a cell. Initial states for the GoL were stored in a Distributed File System (DFS) before being run through MapReduce for processing starts (Figure 4-1).

Algorithm 4-1 *GoL* with 16×10^7 cells on Radenski's Hadoop framework

```

1: class Mapper
2:   cell(row,col) ← function Parse(line)      ▷ parse input test files
3:   function hash(neighbors)
4:   function Emit(cell, tag)
5: class Reducer
6:   function compute(cell, tag)
7:     if last-cell != None then
8:       if Next-State-Is Alive(last cell) then  ▷ Conway's transition rules
9:         function Emit(cell, tag)
10:      end if
11:    end if

```

Figure 4-1. Radenski's cloud-based 2D GoL algorithm

Radenski tested this algorithm on Amazon’s Elastic MR Cloud with a maximum of 16×10^7 cells, using one master node and 16 slave nodes. The shortest processing time without data initialization was 4 mins per single CA timestep. Radenski’s use of MRS afforded some performance improvement for the GoL, but only to an extent: each single MRS job can just process one timestep for the GoL. Extra tools are required if multi-step simulation is required. iMapReduce (Y. Zhang, Gao, Gao, & Wang, 2012), or Apache Oozie (a workflow scheduler) may be possible solutions to this. Another cloud-based GoL was presented by Marques et al. (Marques et al., 2013) (Figure 4-2). Rather than targeting processing speed, they instead focused on leveraging MapReduce to optimize the matrix data size, which helps to overcome the limitation of data size on input and intermediate results storage, and on messaging communications among cloud computing nodes. This leads to an iterative algorithm, in which intermediate results are stored in linked list cells and, after all transition calculations are finished, the final results are aggregated, compressed, and stored as final results. In their study, a 1-trillion cell 2D or 3D CA was presented by using pointers in sparse matrices (e.g. 3 pointers for each cell in 3D scenario) to locate the dependent neighborhood. Using up to 350 instances with 8 cores on each at Microsoft Azure cloud platform, it was able to process a single step in approximately 748 minutes.

Algorithm 4-2 2D/3D *GOL* with 1-trillion-cells on Marques’s MapReduce framework

```

1: class GameofLife
2:   submatrix  $\leftarrow$  function Partition(neighborhood_radius)
3:   Superstep_number  $\leftarrow$  time index
4:   cells  $\leftarrow$  function add(function g(submatrix))    ▷ intermediate results dataset
5:   result  $\leftarrow$  function compress_list(cells)        ▷ final results

```

Figure 4-2. Marques’s 2D/3D GoL algorithm on MapReduce

The key point of comparison with Radenski's and Marques's approach is the latter has ability to handle multi-step cellular automata simulation without using third party software for iterative operations. The MRS approach does not group same key-value pairs at intermediate processing (and this is partially limited by the rigidity of the default Java API). Marques's framework applies sparse matrix and circular linked lists to significantly compress the data. It also splits the overall computing task into smaller subtasks while compressing submatrix datasets. Moreover, Marques's scheme is relatively easy-to-use: users not accustomed to cloud computing can straightforwardly build custom CAs by declaring a master function to dictate transition rules and the neighborhood scheme. However, there are some limitations on partitioning. Although users provide the neighborhood radius for state look-up in the CA to control the segmentation of the sub-matrix and its expansion, this framework may not be able to split the whole input data into a sub-matrix accurately when dealing with irregular neighborhood schemes. In some cases, this could lead to passing a large number of unnecessary messages among the whole system during processing.

After reviewing parallelized voxel CA approaches implemented with GPGPU, MPI and cloud computing techniques, the limitations in computing voxel CA across large lattice-spaces can be better overcome with some re-thinking of the data structures underpinning CA models in simulations. Overcoming these limitations, which are computational rather than theoretical, would allow CAs to be deployed more flexibly and authentically in particular for applied scenarios in geographical sciences, where CAs could become much more useful as media for exploring

computation-intensive what-if scenarios, and for animating evolving silos of big data from geographical science observation platforms.

3. Giraph-based Cellular Automata

3.1 Rethink CA in Giraph

MapReduce and MPI are not always ideal for large-size CA model processing. Consider, for example, that at each iteration of a CA state-update, the model needs to submit one MapReduce job, which includes reading data files from DFS, parsing the input file, initializing cells' status, computing each cell, finishing cell status evolution, reducing the results of computing, and writing this single step result back to DFS. MapReduce also relies on key-value pairs to achieve data processing: this is problematic for CAs, which contain not only the cell-state, but also the cells' neighbor relationship(s). For high-dimensional CA models (e.g., 3D voxel lattices), the traditional key-value approach needs to be flawlessly designed to implement the storage of the cells' initial states, and this also makes computing functions more difficult to implement and code, and more complex to understand and wield.

We mentioned that re-thinking the data structures that support CA models may help overcome those issues. In graph theory (Bondy & Murty, 1976), a graph is an ordered or unordered pair $G = (V, E)$. It is a 2-element tuple that contains a set V of vertices and a set E of edges. Graph theory has been applied in many scientific areas to represent relations, schemas, and structures of physical, biological or information systems. Graphs are naturally suited to CAs. Considering this, a CA lattice can be represented by a graph: cells can be expressed as a vertex, and the neighbor

relationship can be reflected in an edge. By judging the state of the vertices connected with the target vertex, users can easily set up the transition rules to update the state of the target vertex.

Building cellular automata in graph form provides many advantages. For example, for spatial systems, concepts such as centrality, clustering, connectivity and accessibility, can be easily implemented using graph theory and related techniques. The concept of centrality can be used to measure the most central nodes in a network. The clustering concept in graph theories can be used to assess which nodes in networks can be clustered together. A comprehensive review of those applications has been done by Lin (Lin, 2012). Connectivity represents the robustness of a network (J. Wang, Kwan, & Ma, 2014) and accessibility addresses interaction opportunities among various nodes in a network (Hansen, 1959). The algorithmic implementation of this approach is straightforward: graphs and computers have played nicely together for many years. Nevertheless, it remains important to find a suitable and powerful graph processing infrastructure to achieve the massive array sizes and interactivity that are needed for building a very big CA in 3D.

Developed by Valiant (Valiant, 1990), the Bulk Synchronous Parallel (BSP) model is a promising candidate for graph processing of big models. BSP accommodates many super-steps. Within the operation process of a super-step, each computing unit is arranged into a certain amount of vertices or edges, which ensures parallel computing; each computing unit communicates with others through interactive messaging; and when the computing of this unit reaches a barrier, it will stop until other cores complete their message interaction (Figure 4-3). BSP may be

implemented in Apache Giraph, a vertex-centric model that is based on Google's Pregel. Facebook, for example, have used Giraph to analyze 1 trillion graph edges, using 200 machines, in four minutes (Ching, 2013). Giraph lies on top of Hadoop, so it may be deployed and implemented relatively easily on any HDFS-based computing platform. Allied to Giraph, ZooKeeper helps to coordinate computation: when a Giraph job is submitted, one worker is elected as a master by ZooKeeper. At the same time, the input graph will be loaded and vertices or edges will be partitioned and assigned to workers. After executing compute functions in code for certain supersteps or certain halt conditions, workers will save the output back to DFS.

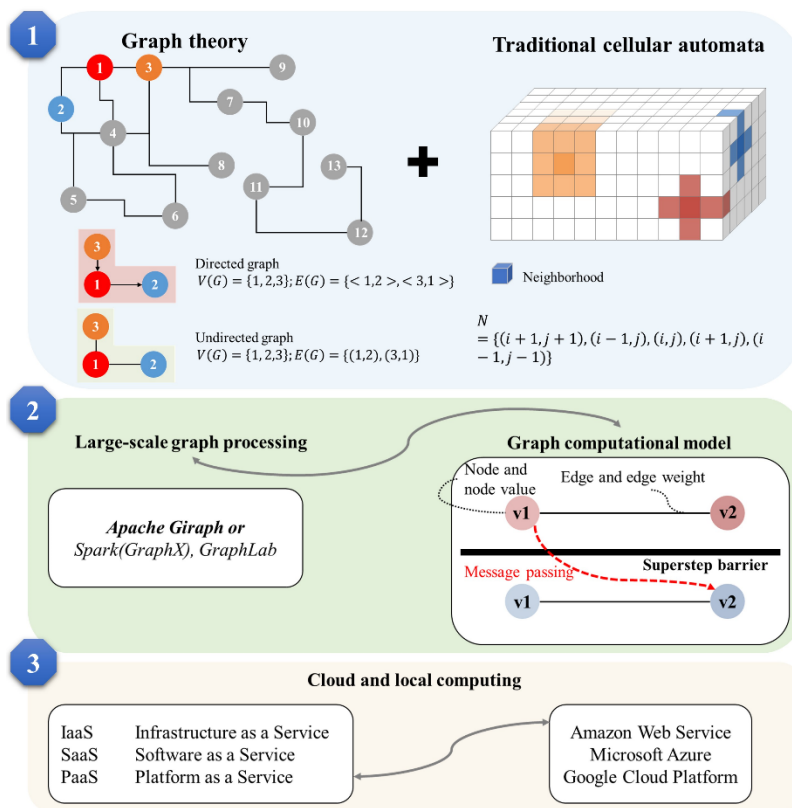


Figure 4-3. workflow to accelerate voxel CA models

3.2 Implementing Massive GoL CA with Giraph

For this research a Giraph-based scheme was built to accommodate a large voxel CA and the feasibility and performance of this framework was tested using the GoL. Most GoL implementations rely on Von Neumann (4 neighbors and the cell) or Moore (8 neighbors and the cell) neighborhood schemes. In a 3D situation, a first-order Von Neumann scheme for a cell has six (extra-cellular) neighbors (Figure 4-4).

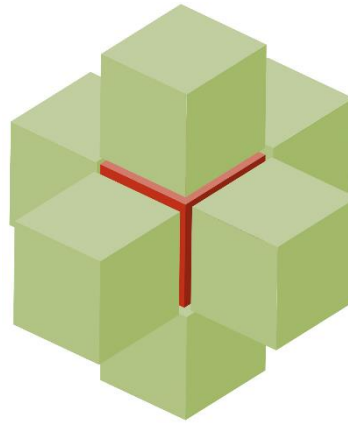


Figure 4-4. A 3D Von Neumann neighborhood scheme

GoL evolution is initially determined by seed states for the CA lattice (no other state data is read-in from outside the model once run). States are binary: alive or dead. Transition rules are straightforward and well-covered in Conway's study (Conway, 1970). For this research, GoL was run with JSON input files, e.g., as `[1,0,[[2,1],[3,0],[4,0]]]`. As a single line in a JSON input text files, this represents a given cell with ID 1 surrounded by another three cells with IDs 2, 3, and 4. The state of cells 1 to 4 in the GoL are dead, alive, dead, and dead respectively. By using JSON input, Giraph can easily load all cell states and neighborhood schemes at the initialization step. This makes coding much easier because in implementing the computing function, we no longer need to consider the neighborhood, and only the

transition rules need to be written into the computing function. When the user defines a certain timestep number in the pseudo-code (Figure 4-5), the MaxSuperstep is set to control the whole running process. At super-step 0, all values include the cells initial states and the neighborhood scheme, and the neighborhood cell-states are all loaded and partitioned to workers.

Algorithm 4-3 Voxel *GoL* on Giraph

```

class GameofLife
1:  MaxSuperStep ← user defined value
2:  function compute(vertex, messages)
3:  if getSuperstep() == 0 then
4:    sendMessage to all neighbors
5:  if getSuperstep() <= MaxSuperStep then
6:    count total live neighbors
7:    if Convey's rule
8:      setVertexValue()
9:    end if
10: else
11:   voteToHalt()
12: end if
13: end if

```

Figure 4-5. Voxel GoL CA implementation on Giraph

3.3 Performance Tests and Discussion

A pressure test was performed on a 115-node Hadoop cluster was used as the platform on AWS to monitor the computational capacity and performance of our voxel Giraph CA (Table 4-1). Approximately 1 trillion vertex and 6 trillion edges were involved, and 500 workers were employed.

Table 4-1. Computing environment for 1 trillion cells GoL experiment

Hardware					Software	
Type	Count	CPU	RAM	Storage	Name	Version
Memory optimized x1.32xlarge instances	115	Intel Xeon E7-8880 v3 128 virtual cores	1952 G	2 * 1920 G SSD	Apache Spark	2.0.2
					Apache Giraph	1.1.0
					Centos	6

We illustrate the resulting performance results over an eleven super-step experiment (Figure 4-6). As shown in this figure, each single super-step can be finished on average in 71 minutes with default hash partitioning. Step 0 is for initialization and step 11 is added by Giraph to the procedure `voteToHalt()`. As shown in Figure 2, the consumed time varies on each step because the cells states change over the processing. For instance, under the GoL transition rules, if a cell status is 1 (alive), there are two "IF" rules to decide whether it will turn 0 (dead). But, if cell status is 0, there is only one IF rule to make it alive. Different numbers of cell status with 1 or 0 will affect the computing time at each super-step.

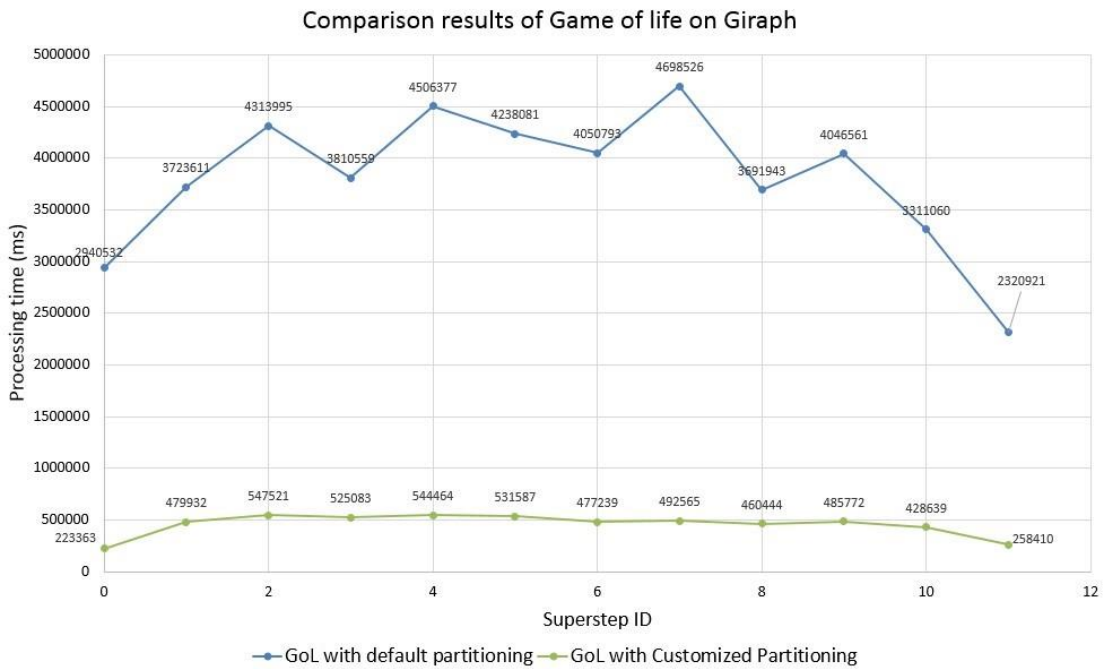


Figure 4-6. Comparison results of GoL: default hash vs. customized partitioning

Using Giraph can significantly improve the computational efficiency of a 3D GoL CA model. Furthermore, by manipulating the data partitioning scheme, this could help achieve better results in some cases. The default method of Giraph partitioning is based on the hash algorithm. This is a general method that could be

applied in many practical simulations such as Shortest Path search and PageRank. Giraph is also an open source graph algorithm computing framework, which affords some flexibility in how users might modify the partitioning scheme. Based on the characteristics of the CA model, each cell has a relatively fixed pattern of neighbors, which ensures that cells do not interact with other cells that are not their neighbors. Hence, if related cells can be partitioned for target cells on the same computing node as much as possible, cross-node communications can be greatly reduced, and this leads to a significant reduction of their drain on processing performance through network communication latency. This solution could introduce significant efficiency, particularly when a large number of computing nodes are used or when the network latency is relatively high. For example, using customized partitioning on a cloud-based platform usually shows better performance than on a local computing cluster.

Patterns for partitioning are clearly important to the scheme described above, and specific neighborhood schemes in different circumstances should be considered. The pseudo-code for customized partitioning for the GoL on Giraph is presented in Figure 4-7. The only difference between this and the default method is that a user needs to apply the `getPartition()` and `getWorker()` functions before the super-step starts. Function `getPartition()` could be used to return the place bundle that corresponds to a specific part of a vertex, and `getWorker()` returns worker indications belonging to the partitions. (For our voxel GoL implementation, each cell has von Neumann six neighbors, except for cells located on the boundary.)

Algorithm 4-4 Voxel *GoL* on Giraph with Customized Partitioning

```
class GameofLifewithCP
1: function getPartition(id,partitionCount) ▷customized partitioning
2: function getWorker(partition) ▷return worker
3: MaxSuperStep ← user defined value
4: function compute(vertex, messages)
5:   if getSuperstep() == 0 then
6:     sendMessage to all neighbors
7:     if getSuperstep() <= MaxSuperStep then
8:       count total live neighbors
9:       if Convey's rule
10:        setVertexValue()
11:     end if
12:   else
13:     voteToHalt()
14:   end if
15: end if
```

Figure 4-7. Giraph-based GoL with customized partitioning

For example, as shown in Figure 4-8(A), for an 81-cell lattice of 3 x 3 x 9 cells, neighbors of cell 14 will be in indices/positions 5, 11, 13, 15, 17, and 23. Assuming the CA is run with only three computing nodes of identical hardware specifications (Figure 4-8(B)), and with hash partitioning, those cells that are related to cell 14 may be partitioned to two or three computing nodes. Hence, when cell 14 is tasked to update its state, it has to call two other computing nodes to ask for dependent information and this generates an unnecessary network overhead. Here part of the voxel GoL lattice was visualized to monitor the status of the tested voxel GoL visually (Figure 4-8(C)).

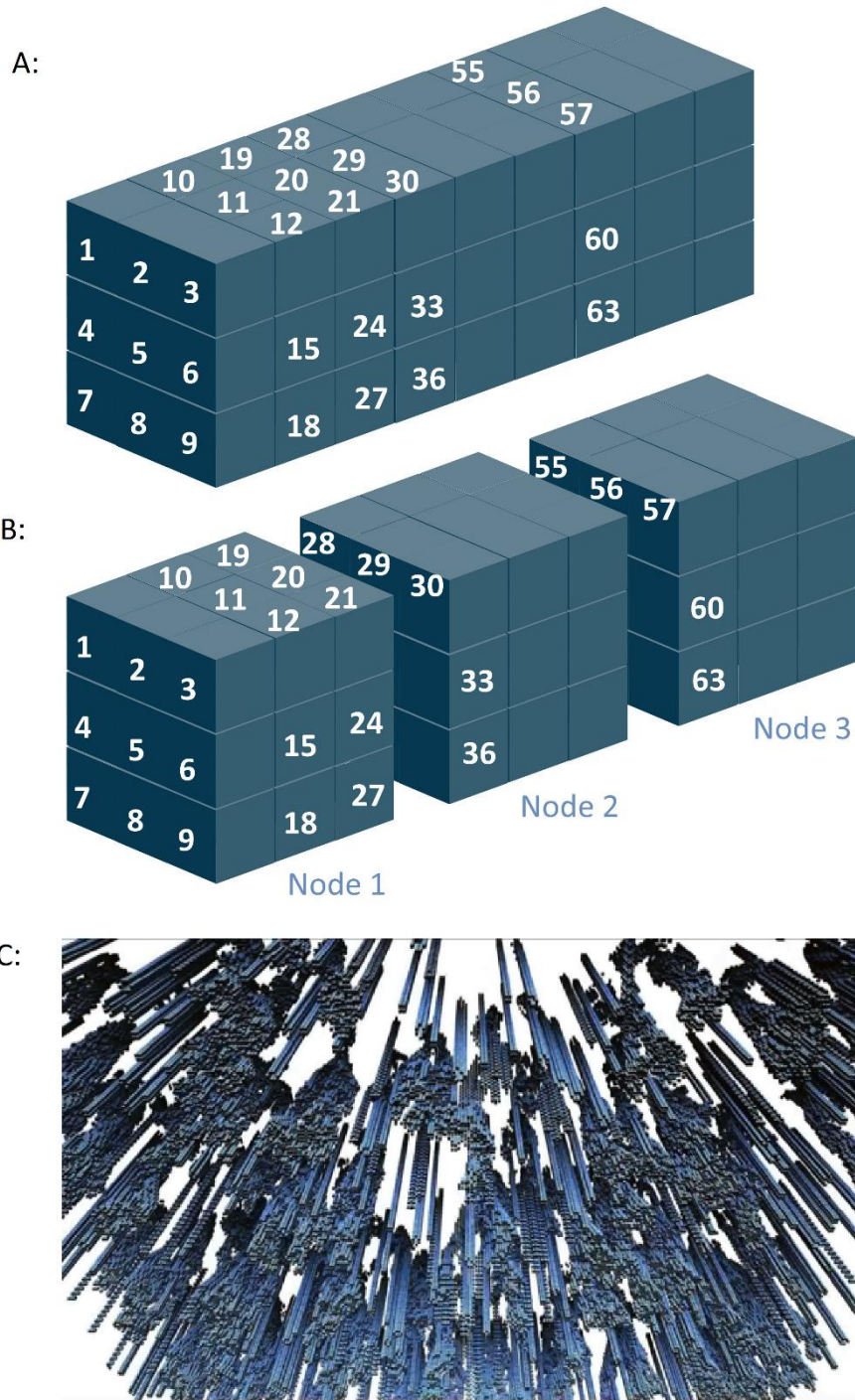


Figure 4-8. Partitioning strategy of a voxel GoL CA

Alternatively, a customized partitioning method could be used such that the initial data might be split into three $3 \times 3 \times 3$ partitions, then assign each of these to three computing nodes. In this alternative scheme, when transitions for cell 14 needs

to be calculated, all related information for that calculation could be acquired within the node where it is located. We show comparison results for the custom partitioning method and a default partitioning for a GoL CA with identical testing environment and input data (Figure 4-6). Customized partitioning proves to be approximately eight times more efficient than the default partitioning method. Here, the computing capacity and performance of our Giraph-based voxel CA was proved. In comparison to same scale voxel GoL in literature, our approach represented up to ~90 times faster in performance. The key reason for this result could be our Giraph-based voxel GoL was performed fully in memory but Marques's scheme can only support disk-based data exchange, which was highly limited by the throughput of disk bandwidth.

Another reason was the neighborhood scheme and initial cell status were written into the input file of our Giraph-GoL, which was needed to be calculated in Marques's approach during each timestep. This will require extra processing time for each timestep to seek neighbors of each cell. The last reason could be HDFS-based framework replicated input data on distributed file systems (3 replicas in our case), which could save data transfer time especially under network latency within cloud computing clusters.

However, GoL is only a toy model that was used for performance tests in recent research. The feasibility and performance of applying Giraph-based voxel CA on real geographical models have not been verified. In following section, an actual geographical model - air pollutant particle dispersal, will be implemented in our voxel CA in massive scale.

4. Massive Scale Voxel Air Pollutant Particle Dispersal Simulation

In this section, a billion cells voxel air pollutant particle dispersal simulation will be demonstrated to represent the capability and performance of our Giraph-based CA model in handling massive scale actual 3D geo-simulation applications.

The dispersal of air pollutant particles is a major factor that affects the health of all lives on earth and it has been a global concern that has drawn much attention in recent years (Rao et al., 2013). The mechanism of air pollutant particles dispersal mainly results from buoyancy-driven air flow (Hajra, 2014). Heavy air pollutant particles with negative buoyancy move downward to the ground due to gravity, which makes their movement less influenced by surrounding air flows and wind. However, air pollutant particles with neutral or positive buoyancy will stay afloat and are easily affected by the airflow system to further transport and dilute.

Wind speed and direction has played an important role in airflow systems It is a major factor in the transportation patterns of air pollutant particle dispersal that mix with surrounding airflow systems. With greater velocity of wind speed, the faster the air pollutant particles move from one place to another. More specifically, the wind and surrounding airflow affect the advection and diffusion of air pollutant particle dispersal. Advection was defined as the transportation of air pollutant particles within the airflow that they are located in, while diffusion refers to a continuous process where air pollutant particles move automatically from high pollutant concentration areas to low pollutant concentration areas without being affected by other forces until the pollutant concentration reaches an equilibrium (Jjumba & Dragicevic, 2015). Turbulence may also cause additional diffusion due to complex airflow conditions

among neighborhoods of specific group of air pollutant particles. However, turbulence will not be considered in this study for the purpose of simplifying the physical model being simulated.

Many current 3D air pollutant dispersal models have been developed using differential equations to represent the air pollutant transportation pattern in urban street canyons or rural areas (Fang et al., 2014; Tomlin, Ghorai, Hart, & Berzins, 1999; Zahran, Smith, & Bennett, 2013). Voxel-based particle applications have also been developed by researchers based on stochastic Lagrangian particle models (Molnar Jr, Szakaly, Meszaros, & Lagzi, 2010). In this study, a simplified voxel particle-based physical model (Jjumba & Dragicevic, 2015) has been applied to represent the mechanism of air pollutant particles transport in 3D space along time.

4.1 Physical Model

We adopted the model designed by Jjumba and colleagues, which included two main components: advection and diffusion (Jjumba & Dragicevic, 2015). The reason to apply this simplified physical model is to 1) focus on demonstrating our cloud-based voxel graph CA computing framework that could be applied in a spatiotemporal simulation of actual 3D phenomena; and 2) prove our framework could help to significantly improve the computational capacity of this simulation model thus potentially offer detailed and large-scale simulation for a large study area. This would overcome a problem mentioned by Jjumba et al. (2015) in their study where they had to use coarse spatial resolution over a small study area for their simulation, and ran for short computational durations to improve computational efficiency with the series-based MATLAB simulation program.

In this physical model, a synthetic study area was developed to assess air pollutant particles advection and diffusion. The whole study area was treated as a set of uniform cubes. Air pollutant particles, fresh air and non-air objects such as ground and buildings were all contained in those uniform cubes. In other words, a group of air pollutant particles in a same voxel was considered as a single object during simulation. For each voxel, 26 voxels that were contiguous to it were considered as its neighbors according to a 3D Moore neighborhood scheme. Using this design, the effects of advection and diffusion from a specific voxel during a single time step will not affect voxels other than those 26 voxels. Another design aspect of this physical model was the further the distance between a voxel and its neighbor, the lesser effects of air pollutant particles advection and diffusion. More specifically, a 3 x 3 x 3 voxels group was used as an example, where air pollutant particles moved from a center voxel to all downwind 3 x 3 voxels due to advection. However, not all the downwind voxels received the same amount of pollutant based on the theory of distance decay (Fotheringham, 1981). Hence, three classes of the neighbors were defined as face position neighbors, diagonal position neighbors and double diagonal position neighbors with distance 1, $\sqrt{2}$, and $\sqrt{3}$. The diffusion process was also affected by the distance of the target voxel and its neighbors. During a single timestep, more air pollutant particles moved to the nearer neighbors than moved to farther neighbors if the pollutant concentration of the target voxel was higher than for other neighbors.

The overall transportation process was represented as the equation below:

$$C^{t+1} = C^t + C_{diffusion} - C_{advection} = C^t - r \frac{\Delta C}{\Delta d} - v \frac{C}{\Delta d}$$

Here, C^t and C^{t+1} represented the concentration of a specific voxel at time step t and $t+1$. The change of the concentration of a specific voxel during processing was due to diffusion $C_{diffusion}$ and advection $C_{advection}$. The concentration change due to diffusion was calculated by using a dispersion coefficient r multiplied by the concentration difference between the target voxel and its neighbors (ΔC) and then divided by the distance Δd . If the value of the concentration change was positive, the pollutant concentration in the target voxel was reduced and *vice versa*. In other words, the pollutant concentration of all voxels was potentially increased or decreased during the diffusion process. Different from diffusion, air pollutant particles moved from the center voxel to all 9 neighbors under wind-driven advection. The amount of the pollutant concentration changes from a specific voxel to one of its neighbors was calculated by using wind velocity v and the distance between this voxel and its neighbor Δd .

4.2 Graph-Based Voxel Air Pollutant CA

On the basis of the physical model, as described above, a computational model was designed in extensible form as a CA, which was tasked with animating air pollutant particles dispersal dynamics according to the processes outlined in the physical model. Within the lattice, each voxel was represented dynamically, as an individual and autonomous CA. For the simulations discussed in this research, the study area was specified as a box lattice. In other words, the entire study area was considered as being composed of same-size voxels, each represented at unit 1 in volume, which is consistent with the explanatory concepts of the physical model. While states are allowed to transition through each voxel in the lattice, the cells in the

CA were not allowed to move their locations or to be deleted (although they may change state). Three types of CA were considered: air pollutant particles, fresh air and non-air objects. At the initial state in simulation, the fresh air was specified as air voxels. The status/concentration of air pollutant particles voxels that transit during each time step. The status of non-air voxels was never changed. To fit the continuous air pollutant concentration changing in the physical model, a continuous CA model was applied by using cell state values to represent air pollutant concentration. All air pollutant source voxels were initialized as 1. During advection and diffusion processing, the concentration will reduce until a value of 0 if the pollutant source was set as temporary pollutant source such as a vehicle exhaust in an application of detailed urban simulation. It could be set as one at beginning of each time step to represent a permanent or long-term pollutant source such as a factory. Synthetic start conditions were designed in the examples that will be shown in this study, but seed conditions could alternatively be specified with real data if available. The resolution that our CA model offers in simulation matches the needs of synthetic data simulation. If finer or courser resolutions be required, they could easily be accommodated by simply altering the specification of CA dimensions in the CA model.

Neighborhoods for state exchange are a critical component of CA. According to the air pollutant particles dispersal physical model, target cells are only communicated by those cells at first-order Moore Neighborhood positions in the lattice (Figure 4-9(A)). Three types of neighbors were defined in CA model to fit the

physical model. However, with the graph-based CA, the absolute physical spatial relationship among each voxel will not need to be considered.

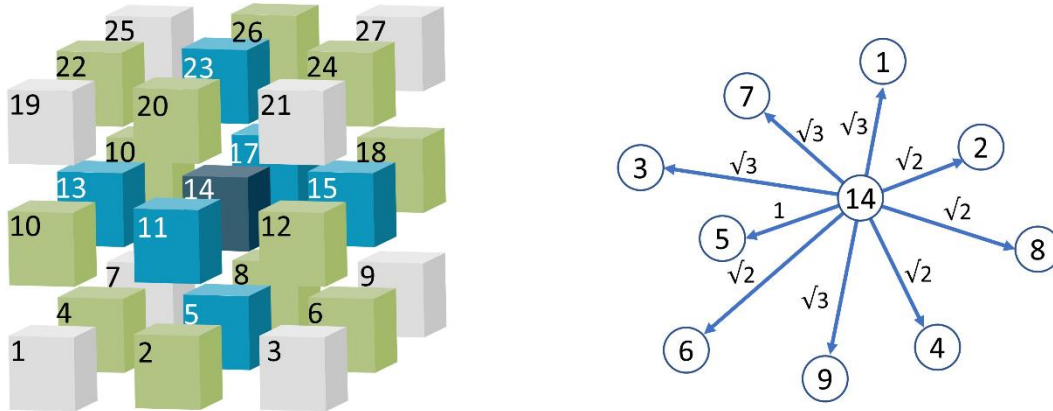


Figure 4-9. Traditional voxel air CA vs. Graph-based voxel air CA

The neighborhood scheme and the distance among each voxel were the major focus for designing the graph-based approach. A directed graph was applied to represent the voxels by nodes and the neighborhood scheme by using edges (Figure 4-9(B)). Using neighbors of voxel #14 that labeled from 1 to 9 as an example, the weight of each edge represented the distance between the linked two voxels. A JSON file was loaded as input at the initial stage as below:

```
[targetvoxel, (0, status), [[neighbor1, (distance, status)], ... [neighborn, (distance, status)]]]
```

This JSON fit for default input format of Apache Giraph. The 0 represented the distance from the target voxel to itself. The rest of the file represented all neighbors of this target voxel and the corresponding distance between them as well as the status of those neighbors. In this CA model, the overall transition rules were represented as:

$$C^{t+1} = C^t - r \frac{\Delta C_k}{\Delta d_k} + \sum \frac{(v_i C_i - v_j C)}{\Delta d}$$

Here, with given wind direction i for influx wind direction and j for outflux wind direction, the change of air pollutant concentration for a specific voxel, will be equal

to the summary of influx air pollutant $v_i C_i$ from any direction i and outflux $v_j C$ to any direction j with considering the neighbors distance Δd and self-diffusion to all surrounding lower concentration voxel k .

4.3 Experiment Results and Discussion

In this section, two experiments were designed to test the feasibility and performance of the Giraph-based voxel CA framework for the air pollutant particles dispersal simulation. The first experiment was developed to verify the framework could handle the exact same simulation as the experiments developed by (Jjumba & Dragicevic, 2015). In this experiment, we replicated the single source and multi-source air pollutant particles diffusion with the same parameters setting as used in the original study. In addition, to further explore the computational capability of our framework, another experiment was designed using 1 billion voxels, which was much larger than the 64000 voxels in the literature, to test the computational performance with a massive simulation. In this experiment, we applied a continuous single source air pollutant simulation with fixed direct wind for 1000 timesteps, which was much longer than the 90 timestep in the previous study. We then did a comparison between these tests and the previous research from the perspective of modeling, implementation, and performance at different scales.

The implementation of the CA simulation in large-scale graph processing and graph-based computing on Giraph with customized partitioning was shown in Figure 4-10 (Figure 4-10). The partitioning operation was performed before supersteps begin. When the simulation task was submitted to the computing cluster, the simulation user may define the number of supersteps to represent a desired time

period, as well as the corresponding relations between supersteps and real simulation time. Initialization was performed before superstep 0. As we discussed, this step was straightforward in easing specification and parameterization of the model. As per the computational design of the CA model, if necessary, different situations (different parameters, different transition rules, different neighborhoods) could be applied to explain air pollutant particles in different time periods. From superstep 1, the simulation automatically applied the transition rules, depending on the values of each parameter set.

Algorithm 4-5 Air pollutant particles dispersal on Giraph with Customized Partitioning

```

1: getPartition(id,partitionCount) ▷partitioning for each 3 x 3 x 3 voxel group
2: getWorker(partition) ▷return worker
3: MaxSuperstep ← user defined simulated time period
4: Compute (vertex, messages)
5:   if getSuperstep() == 0 then
6:     sendMessage along edges to detect neighbors
7:   if getSuperstep() <= MaxSuperstep then
8:     Foreach cell in CA:
9:        $C^{t+1} = C^t + C_{diffusion} - C_{advection}$ 
10:    Emit ( $C^{t+1}$ )
11:  else
12:    voteToHalt()
13:  end if
14: end if

```

Figure 4-10. Air pollutant particles dispersal CA on Giraph

According to the physical model design, there was no air pollutant particles transfer reach outside the range of a 3D Moore neighborhood during a single timestep. Hence, the neighbors of each cell in the lattice were only located inside a 3 x 3 x 3 positions. In other words, the whole CA model could be considered as a bundle of 3 x 3 x 3 voxel groups. Cells on each voxel groups can only evolve inside their own voxel groups during single timestep, rather than influencing cells in other voxel groups. This was an important point to make, as it suggested some

computational schemes that can be injected into the simulation to allow for computing efficiency in ways that ally to known physics. Specifically, because each voxel group was independent, simulation users could perform partitioning for specific numbers of those voxel groups, depending on the computing capacities of each node in any available computing cluster. For example, if there were a total of nine-voxel-groups to be resolved in simulation, and three corresponding computing nodes, a user could arrange each node with three voxel groups to achieve balanced computing performance. The computational environment for air pollutant particles simulation involved 1008 computing vcores and 5.12 Tb memory at local cloud environment. A 2 Pb HDFS was configured and the whole resource are managed under YARN (Table 4-2).

Table 4-2. Computing environment for air pollutant CA experiments

Hardware				Software	
Role	Count	CPU	RAM	Name	Version
Master Node	2	2 x Intel Xeon E5-2680v4 2.4GHz	256 G	Apache Spark	2.2.0
				Apache Giraph	1.2.0
Computing Node	18	2 x Intel Xeon E5-2690v4 2.6GHz	256 G	Centos	6.9
				Java Server VM	1.8.0_152 64Bit

The first experiment was a natural air pollutant particles dispersal with continuous pollutant source under no wind environment. We applied 0.009 for diffusion coefficient (Jjumba & Dragicevic, 2015) and 0 for wind speed. Cell status of the pollutant source was set as 1 before each timestep started. 50 timesteps in totally was pre-configured at the initialization stage of simulator. Status of each cell was recorded during whole processing. For example, after zoomed into the cells

around pollutant source after first timestep finished, we can clearly see the change of each cell around the pollutant source due to diffusion process (Figure 4-11).

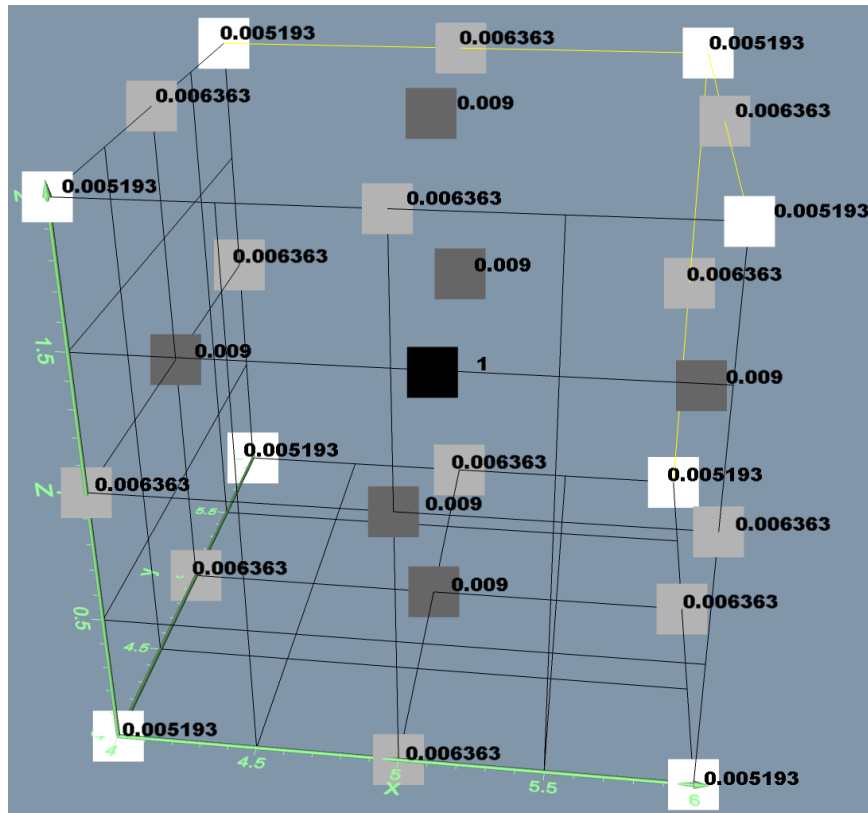


Figure 4-11. Visualization of first timestep of single source air pollutant diffusion simulation

With different neighborhood distances, the air pollutant concentration in each cell was different due to the transition rules. The cells status of the whole system was visualized by each 10 timesteps to display the overall air pollutant concentration distribution pattern (Figure 4-12).

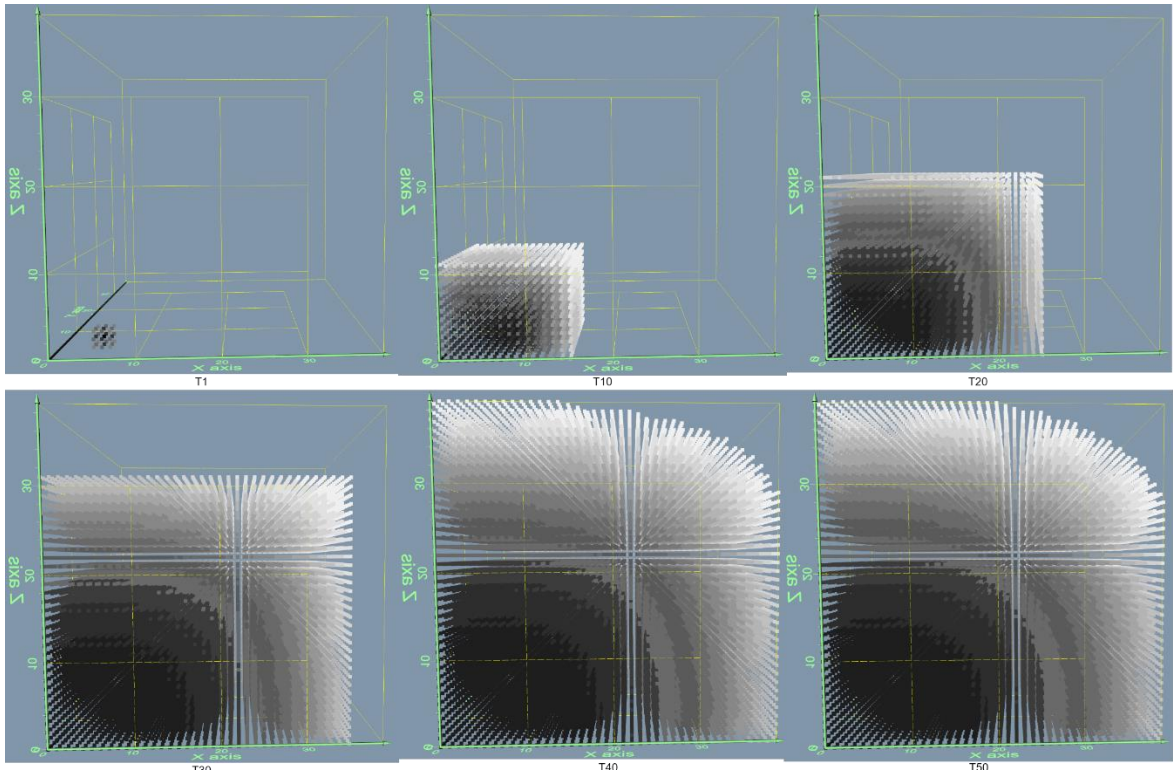


Figure 4-12. Single source air pollutant natural diffusion simulation for 50 timesteps

Under the same parameter configurations, a multi-source air pollutant particles dispersal simulation was deployed (Figure 4-13). In this experiment, three continuous pollutant sources were applied at the initialization stage. This simulator was run for 50 timesteps and the results were visualized at each of 10 timesteps. From those visualization results, the pattern of how the air pollutant particles were transported from the pollutant source to the whole study area could be seen. For example, there was no significant pattern change from timestep 40 to timestep 50 under this visualization. In other words, air voxels that were far from the pollutant source would be affected by air pollutants very slowly due to the long transportation distance, neighborhood transportation chain and the fixed diffusion rate of the pollutant sources.

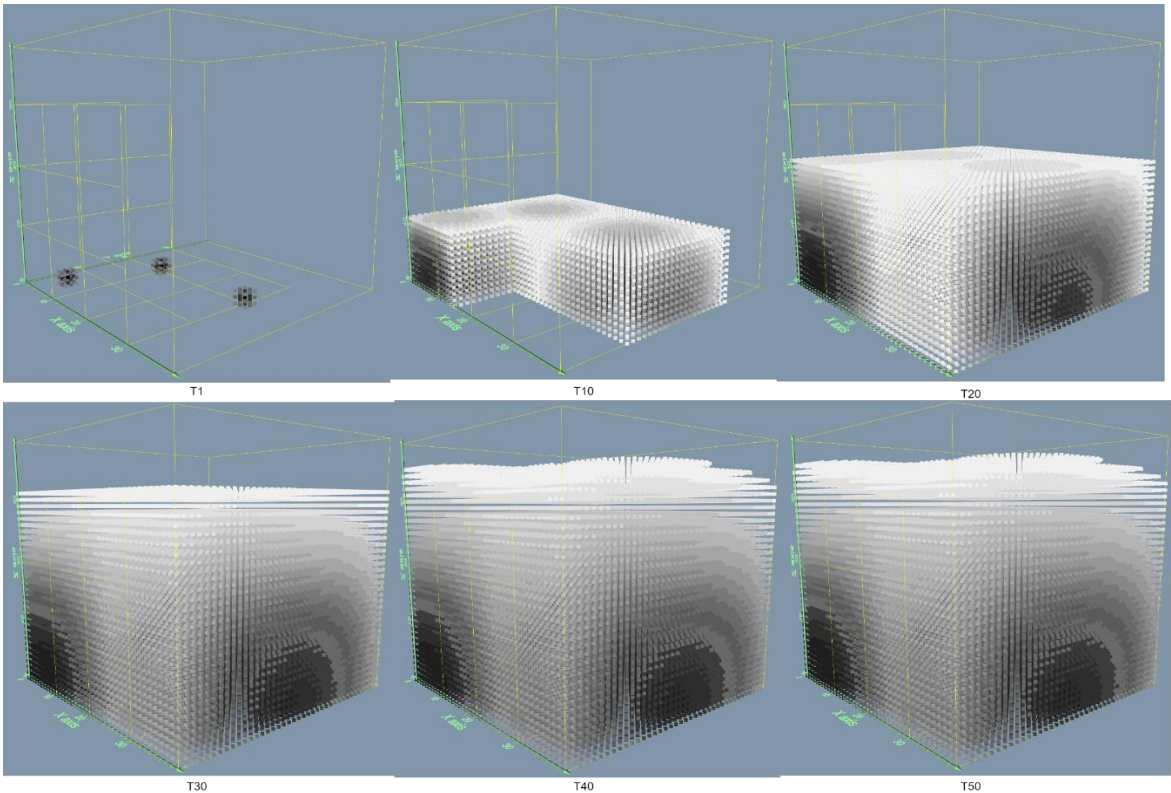


Figure 4-13. Multi-sources air pollutant natural diffusion simulation

In addition, to replicate the original study's simulations under our cloud-based computing framework, the second experiment was designed with the same parameter configuration as in the previous experiment, but with a much larger scale. In this experiment, an air pollutant particles dispersal simulation was performed with single changeable pollutant source (continuous for first 125 timesteps) under fixed wind environment (Figure 4-14). 1 billion voxels were involved during this simulation and 1000 timesteps was set for entire processing. Wind speed in this experiment was setup as 4 along the Y axis. By visualizing each 125 timesteps, it was clearly to find that the air pollutant was dispersed to the entire study area at timestep 125. Then, all the air pollutant particles were transported along the Y axis and the concentration of them was gradually decreased due to advection and diffusion. At timestep 1000, the

concentration of air pollutant in all visible voxels was very low and some non-pollutant air voxels could be even found among them.

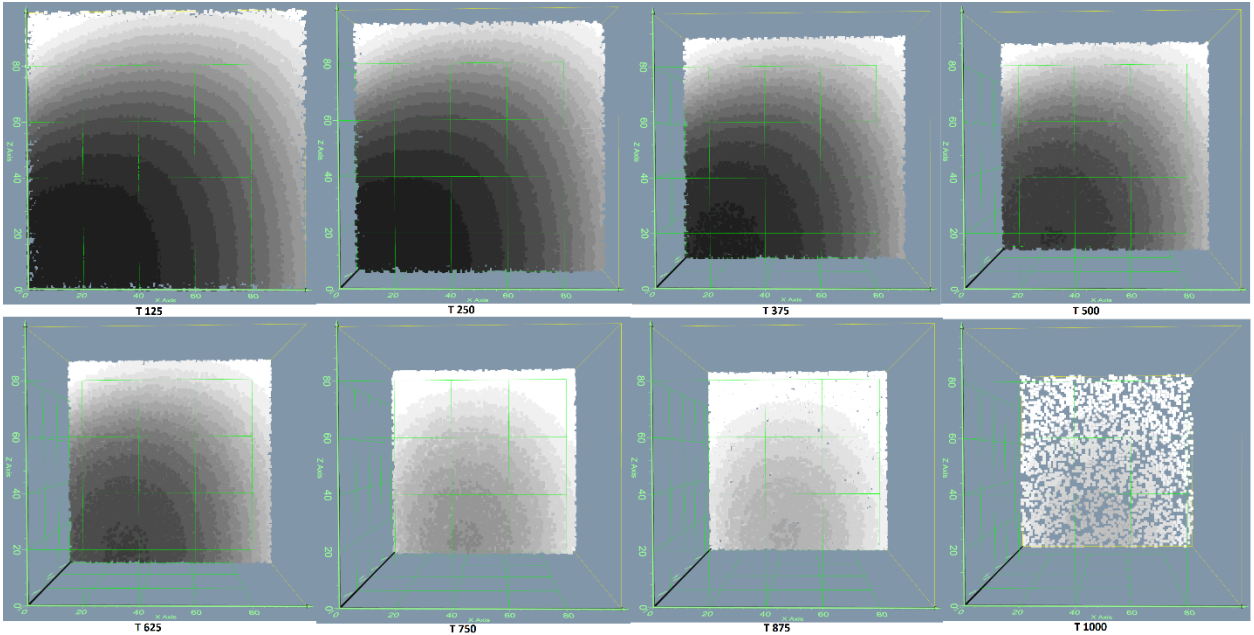


Figure 4-14. One-billion-cells single source air pollutant dispersal simulation for 1000 timesteps

In this experiment, the simulation was run under our computing framework with customized partitioning to further improve computing performance. However, it should be pointed out that voxel visualization performance was a bottleneck with this simulation. Hence, to focus on assessing the computing performance of our framework, we recorded processing time for each timestep. At the same time, we visualized the results with a resampling approach. Hence, 1 billion cells were calculated but only 1 million cells were displayed due to the limitation of visualization performance.

A comparison between our framework and that of the original study was performed from the perspective of model design, model implementation, and computing performance (Table 4-3). Both studies were adopting voxel CA as a media

to simulate the air pollutant particles dispersal with synthetic data. In our study, we applied the simplified physical model with same parameters as in the earlier work as a benchmark to test the computing performance of our framework. In their study, they applied traditional voxel CA to model air advection and diffusion in 3D space. By using series programming in MATLAB software on a single workstation, they simulated air pollutant dispersal with 64000 voxels and up to 90 timesteps, which was limited by the computational capacity to achieve a higher resolution simulation with more timesteps. With the same experimental environment and implementation approach, their simulation was re-tested by upscaling to 1 million cells, which represented about 4.7 hours processing time per each timestep. For 1 billion cells simulation, the model did not finish a single timestep after 24 hours. The same simulation was then tested in our experiment under our cloud-based computing framework, where this framework used 16.2 seconds processing time for 1 million cells and 47.3 seconds per timestep for 1 bill-cell simulation, which successfully overcame the computational capacity limitation shown in the original study. We also successfully tested our simulator with 1000 timesteps.

Table 4-3. A comparison of Jjumba’s study and ours

	(Jjumba & Dragicevic, 2015)	Our study
Topic	Voxel CA - Air particles dispersal	Same
Physical model	Advection and Diffusion	Same
Modeling	Traditional CA	Graph-based CA
Implementation	MATLAB (series)	Apache Giraph (distributed)
Test env	Single Workstation	Cloud Env
Scale	64000 cells	Up to 1 trillion cells as tested. Unlimited in theory

Performance @ 1million cells	4.7 hours per timestep	16.2 seconds per timestep
Performance @ 1billion cells	Over 24 hours per timestep	47.3 seconds per timestep
Time steps	Up to 90	Up to 1000 as tested. Unlimited in theory

In the above two experiments, an in-memory Giraph computing was applied, which consumed approximately 13 times the input graph size memory. Giraph can also be configured in out-of-core computing mode, which can help to solve this memory limitation issue to some extent. In other words, it can help to support larger graph computing that requires a certain memory size. For example, in the air pollutant particles dispersal experiment, it was possible to run out-of-core computing to simulate 1 trillion-cells for an even larger coverage study area or to increase the resolution of air pollutant voxels. However, due to a limitation with the bandwidth of disks, it would take a longer time to finish the task.

These two air pollutant dispersal experiments addressed the task of adapting a Giraph voxel CA framework to a scientific problem beyond the well-understood GoL test scenario. A massive scale voxel graph CA was built that depended on an air pollutant particle dispersal physical model. These efforts shown how a graph-based voxel CA was able to support GIScience research. This novel and powerful simulation tool has the potential to be widely used in many areas with theoretically unlimited computing power (only limited by the specific computing framework and budget available for computing resources).

5. Conclusions

The research objective of this study has been to develop a novel method for computing using a very large voxel CA with high dimensions with efficiency that goes beyond existing desktop CPU, GPGPU, MPI, and cloud schemes, with the view that bigger and better CA models with high dimensions can provide researchers in applied geographical contexts with more detail and processing power to better represent phenomena of interest to them. And begin to build such a model in a way that can take advantage of ever-emerging big data resources that might be available.

In this research, we have demonstrated that we were able to obtain significantly more efficiency by employing Giraph-based voxel CA models, and that this efficiency can be achieved on both local clusters and cloud platforms. Using 1 trillion voxel GoL as a testbed, our framework successfully processed on AWS from ~ 8 minutes per timestep, which was about 90 times faster than the same scale simulation in literature. In addition to assess the feasibility and performance of our framework with actual geographical applications, a 3D air pollutant particles dispersal model was adopted. In comparison with the upscaled original simulations that implemented in series programming via MATLAB, which cannot be done in at least 24 hours. Our approach represented a billion cells air advection and diffusion in 47.3 seconds per timestep, which overcame the computational limitation in original study. Furthermore, CAs based on graph specifications could help to make model-design more intuitive, especially in high dimensions. By simply modifying the input files, the model can be transformed from 2D into 3D without modifying the whole structure

of the compute function and where changes to only a small number of parameters would be required in some cases.

Further improvements in performance could be generated by optimizing the input and output functions of the Giraph scheme, which would in turn help to compress the size of data and reduce the overhead of large data partitioning, passing, and storage. At the same time, many computing clusters are composed of multiple computing nodes with different performance capabilities. Allocating different sub-tasks to those nodes according to the actual performance of those nodes may become another effective method to further improve the processing performance.

Chapter 5 : Conclusions and Future Work

1. Conclusions and Limitations

Big data offers new opportunities for geographical scientific research. With an ever-increasing volume of velocity, veracity, and variety of data, it is possible to provide significant computational tools to support ongoing scientific research and generate more detailed, accurate and trust-worthy results for research. However, it is a grand challenge to mine those data with models and transfer model results into valuable information, further supporting geographical research. More specifically, to process big datasets requires a correspondingly big computational capacity to allow those processes to be successfully finished in a reasonable time (e.g. in hours, not days). In geographical science, the rapid expansion of geospatial datasets, e.g., RS imagery datasets, provide a solid foundation for geographical researchers to undertake scientific research at large scales or in greater detail. However, limited by computing resources as well as computational capacity, many geographical researchers have not benefitted from this big data era, and have not been able to successfully undertake big geographical research in a reasonable time. This dissertation contributes a set of methods for geographical researchers, especially for those who do not have access to traditional HPC cluster computing resources, to lift the block relating to a lack of big computational capacity and thus be able to use big geospatial data to contribute to geographical sciences. More importantly, this work contributes to the operational use of RS data where a shorter turnaround time to get analysis results is necessary and a benefit for many different applications. For

example, it could potentially utilize a big RS observation dataset to monitor, analysis, and predict natural hazards such as flooding and forest fire spread and evacuations in a near real-time or even real-time manner. This dissertation presents three studies that utilize different perspectives to demonstrate how an open source computing framework can be integrated with cutting edge cloud computing technologies to implement large scale geospatial data processing and complex geographical model simulation.

RS imagery dataset classification was selected in the first study in this dissertation to represent how to design and implement a universal scalable computing framework by coupled with existing RS imagery classification approaches to classify multi-sourced RS imagery datasets on cloud in efficiency. In this study, we exploited Apache Spark to build a multi-sourced RS imagery classification framework and successfully deployed and tested it in both commercial cloud platform and local cloud environment. This study further explored the feasibility and performance of this framework under two different study areas, with different sources of RS imagery datasets, by using supervised and unsupervised classification methods. Our framework suggests several promising advantages: 1) It has the flexibility to process RS datasets in multi-spatial, multispectral or multi-temporal cases with slight parameters adjustment, thereby significantly saving the time cost of reprogramming brand new toolkits for different purposes; 2) It is possible to exploit the benefits of cloud platforms to gain unlimited computing resources theoretically and provide highly efficient performance; 3) It is highly accessible to multisource data storage, even in the cloud, to reduce the data transformation cost.

This study is the first step in this direction and significant improvements can still be made to the framework that has been developed. Currently, only two classification methods were tested using this framework. However, it is insufficient to rely on those two classification approaches to process all RS imagery classification tasks in actual research. This framework should be used with additional classifiers to make the framework a practical tool to support a broad range of geographical studies. More importantly, the classifiers used with this framework in this dissertation were both pixel-wise RS imagery processing functions. Some RS imagery classifiers with computer vision technologies (e.g., objected oriented, or deep learning based) could fit better for other types of geographical studies. Integrating those classifiers into this framework could enrich the functions to support RS imagery classification under more scenarios. The current partitioning may not fit for computer vision-based classifiers to keep them perfectly embedded into our framework with high computational efficiency and so alternative methods may need further investigation.

Another issue remaining from the first study is how to handle missing data, as this is a common issue when mining RS imagery datasets especially for large areas, which hinders researcher from obtaining complete LULC information to further support their research. The second study aims to contribute a solution to this issue by designing and implementing a Spark-based in-memory distributed Markov Chain and suitability processing algorithm on the cloud, and integrating this algorithm with a self-designed Giraph-based distributed CA on the cloud to fill LULC gaps. A comprehensive workflow is introduced starting with integrating existing RS imagery classification framework to access the input data, and continue to apply a machine

learning approach to perform multivariable clustering for a big study area (with diverse ground features and significant different environmental and social-economics conditions) for calibrating the training models, include potential impact factors that are available to drive the gap filling model, and finally fill the LULC gaps using a simulation process. In this study, we successfully tested LULC gap filling processing for the entire IMAR region using 11 different auxiliary datasets and the developed framework with an overall accuracy of 88.16%.

The accuracy assessment results represent for some LULC types, e.g., water body and human land use area, the accuracy is not satisfaction. The main reason is detailed transition mechanism of those LULC types along time series is unclear though related auxiliary datasets were added into the model training, especially for human land use, which could be possibly affected by real-time policies. A possible solution for this is to gather more local information such as social media data, government reports, news and policies to support describe the mechanism of those specific LULC types. Ground truth points, if available, could be considered to add in to further calibrate the simulation results. Another limitation of this study is for now, a classic 5 x 5 2D CA neighborhood schema was applied into this study to drive the CA model to rank and select the highest possibility of each cell transit from one class to another. With different neighborhood schema, the result could be possibly different, especially for the edge smoothing for simulated results of each SR, the coverage and schema of the neighborhood should be carefully considered.

In addition to accelerate the data processing and mending, the third study in this dissertation support boosting a theoretical universal geographical model - CA which

could potential consume big geographical data to serve as a media for big geographical complex system simulation and prediction. By fully rethinking traditional CA from perspective of modeling, computation and computing resource accessing, a Giraph-based voxel CA computing framework was developed to support massive scale geographical complex systems simulation and prediction in high dimensional space. This framework was tested on two different cloud platforms by reaching up to 1 trillion cells in memory computing capacity in voxel space. With comparing with same model and scale approach in literature, our framework appears about 90 times faster in computing efficiency. This study further explored this framework with a real geographical model --- air pollutant particle dispersal and successfully simulated the air pollutant particles diffusion and advection with synthetic data.

One of the limitations in this study is real observation-based data was not be used in the air pollutant particles simulation due to unavailability. Hence, we can only test the computational performance in comparison with same model that implemented with series programming in literature rather than further test the simulation accuracy. From the computational perspective, another limitation is this study is purely in memory, which does fit the concept in cloud computing that “trade space for time”. However, in actual research, the budget probably cannot support the computing resource for pure in memory processing. In this study, we introduced the “out of core” mechanism to use part of disk space as an alternative to solve the memory deficiency. But it will reduce the computational performance due to the low throughput of disk (comparing with memory). Hence, compressing input and output

data by removing the redundancy information and makes them fit for the memory size as much as possible could be a better solution though the computational complexity may be increased in some cases.

2. Significant contributions

The goal of this dissertation is to advance our understanding of using cloud computing to support geographical researchers to benefit from big geospatial data and further contribute to geographical sciences. The major findings and significant contribution of studies in this dissertation are summarized in the following paragraphs.

Contribution 1: In the first study (Chapter 2), an in-memory Spark-based distributed cloud computing framework for multi-sourced RS imagery classification was developed. This framework has been tested and deployed on different cloud platforms to process RS imagery dataset from different sources using alterable classifiers with only minor parameter adjustments. An onsite visualization approach to convert text-based output as human readable figures was implemented to produce ready-to-use results in a distributed manner on the cloud. The results of this study show this framework could be a robust solution to potentially support big RS imagery data classification (with sufficient computing resources) in big geographical research such as LULC classification tasks.

Contribution 2: One of the classification approaches applied in the first study (Chapter 2) is SVM that is originally designed as a model for linear-based binary classification purposes. To cater to RS imagery classification tasks that require multiple classes and non-linear processes, we extended a distributed cloud based

SVM in Spark MLlib with a kernel trick and a *one-against-all* computing strategy to make it fit very well for RS imagery classification on the cloud, which could be seamlessly embedded into our framework. Using Landsat 8 imagery data for the entire IMAR region as a test input, we were able to successfully classify ~ 600 images in a single run with 2284 seconds, which was much faster than processed it on single workstation (over 24 hours).

Contribution 3: In the second study (Chapter 3), a comprehensive solution was introduced that be able to integrate with existing cloud-based RS image processing framework, for example, our first study, in order to deal with LULC data gap filling. We used a commonly used prediction model, Markov Chain CA to do the gap filling with existing observation data and 11 different auxiliary datasets in biophysical and socio-economic. We finally successfully tested this framework with entire IMAR as study area on cloud with overall 88.16% accuracy. We also found with machine learning based clustering, the gap filling model could be better trained and the overall accuracy increase about 4% as compared to treating the whole study area as a single region.

Contribution 4: Markov Chain - CA model is a classic LULC simulation model, but it has been never tested on LULC studies with large computing clusters in the literature. In Chapter 3, we described the design of a Spark-based in-memory distributed Markov Chain and Suitability processing algorithm integrated with a Giraph-based distributed Cellular Automata algorithm on the cloud with a customized partitioning strategy to implement big gap filling tasks that could not be done with a single workstation. The gap filling tasks (~18 billion cells in total included LULC

data for 3 years and 11 auxiliary datasets with 1.28 billion cells per layer) were also been successfully tested with our framework.

Contribution 5: Considering big models need to be accelerated in addition to handling big data processing, we extended the computational model presented in Chapter 3 to develop a cloud-based distributed voxel CA model to support massive scale geographical complex systems simulation in 3D. In this study (Chapter 4), we re-thought the traditional CA and reconstructed traditional CA with graph theory and extended it in voxel space. We also implemented a voxel graph CA with Apache Giraph, a cloud-based graph processing framework on cloud and tested it with a standard testbed GoL on commercial cloud platform in order to demonstrate the computational capacity and performance for very large CA lattices (up to 1 trillion cells in this study). We found our approach could reach up to about 90 times faster than same scale studies in literature with customized partitioning computing strategy.

Contribution 6: Using the framework presented in Chapter 4, we further assessed this framework with an actual geographical model in 3D. By adopting an air pollutant particles dispersal model, we successfully simulated the air pollutant particles diffusion and advection at massive scale (up to 1 billion cells with 1000 timesteps). In comparison to the original series MATLAB based implementation using the same physical model, we proved our framework could run the same simulation with significant better computational performance that we achieved 47.3 seconds of processing time per timestep for 1 billion cells simulation, which took over 24 hours with original approaches in literature.

3. Future work

With advancing RS observation platforms and computing engineering technologies, massive volume of geographical datasets has been published in recent years. Big geographical research that applied large scale and fine resolution datasets is drawing attention from geographical researchers who was suffering from limited computing capacity and unavailable big datasets. In this dissertation, three studies are conducted with three selected major challenges of geographical research in this big data era, to discuss several possible solutions for geographical researchers nowadays to undertake big geographical studies. However, limitations of our studies are mentioned in previous discussion. To overcome those issues, some tasks are proposed to be accomplished in the future studies.

For the first study, we are considering making it as a comprehensive RS imagery datasets processing framework to support geographical studies in various scenarios. In addition to add more RS imagery classification functions e.g. random forest (Liaw & Wiener, 2002), ISODATA (Ball & Hall, 1965) and some customized classifiers by encouraging community to contribute, into this framework, we can also add RS imagery preprocessing functions such as cloud-removal, atmospheric correction, and mosaicking to allow users to start with raw RS datasets to generate LULC information. Furthermore, an automatic accuracy assessment function could be built in to allow user to upload ground truth data as validate and report the classification accuracy in real time. Besides, we could develop a user interface that allow users to apply the basic functions of it with a couple of clicks, which may benefit for many geographical researchers without coding experience.

In the second study, increasing the modeling accuracy for specific LULC in study by digging into the transition mechanism of them could be tested. On the other hand, we could integrate the classification framework in the first study and publish them as web service, which could allow users to select their study areas on cloud, choose the RS dataset they need, upload their auxiliary datasets and setup customized MCE parameters. After a single run, it could provide users a complete LULC map for their selected study area without further coding and deployment.

Air pollutant particle dispersal was successfully simulated with our framework in the third study. However, we only applied synthetic data with the physical model to test the voxel CA. Using some complex models in geographical sciences, e.g., lattice-Bozeman (Wolf-Gladrow, 2000) based models that convert models built on partial differential equations into particle models would be useful, as well as further involving more variables into the model simulation process for specific case studies. Using air pollutant particle dispersal simulation as an example, although observation datasets with very high resolution (e.g., 1 meter in 3D) that could fit into this model were unavailable for testing for this research study, we could expect datasets would be available in the future to support such testing. In the meantime, the simulation in this study was implemented in a synthetic study area. A further study involving processing data in an actual scenario could be undertaken to achieve a more realistic use case. For example, by using building height data from OpenStreetMap (OSM) (Haklay & Weber, 2008), this model could be potentially applied to simulate and predict air pollution distribution patterns in Manhattan, New York City, NY.

Bibliography

- Absardi, Z. N., & Javidan, R. (2017). *Classification of big satellite images using hadoop clusters for land cover recognition*. Paper presented at the 2017 IEEE 4th International Conference on Knowledge-Based Engineering and Innovation (KBEI).
- Amazon, E. (2015). Amazon web services. Available in: <http://aws.amazon.com/es/ec2/>(November 2012).
- Amazon Web Services Inc. (2018a). Landsat on AWS. Retrieved from <https://aws.amazon.com/public-datasets/landsat/>
- Amazon Web Services Inc. (2018b). MODIS on AWS. Retrieved from <https://docs.opendata.aws/modis-pds/readme.html>
- Apache Software Foundation. (2018a). Apache Spark. Retrieved from <https://spark.apache.org/>
- Apache Software Foundation. (2018b). Tuning Spark. Retrieved from <http://spark.apache.org/docs/latest/tuning.html#tuning-spark>
- Avolio, M. V., Errera, A., Lupiano, V., Mazzanti, P., & Di Gregorio, S. (2017). VALANCA: A Cellular Automata Model for Simulating Snow Avalanches. *Journal of Cellular Automata*, 12(5).
- Bai, Y., Han, X., Wu, J., Chen, Z., & Li, L. (2004). Ecosystem stability and compensatory effects in the Inner Mongolia grassland. *Nature*, 431(7005), 181-184.
- Bak, P., Tang, C., & Wiesenfeld, K. (1987). Self-organized criticality: An explanation of the 1/f noise. *Physical review letters*, 59(4), 381.
- Baker, W. L. (1989). A review of models of landscape change. *Landscape ecology*, 2(2), 111-133.
- Ball, G. H., & Hall, D. J. (1965). *ISODATA, a novel method of data analysis and pattern classification*. Retrieved from
- Ballester, C., Bertalmio, M., Caselles, V., Sapiro, G., & Verdera, J. (2000). Filling-in by joint interpolation of vector fields and gray levels.
- Bell, E. J., & Hinojosa, R. (1977). Markov analysis of land use change: continuous time and stationary processes. *Socio-Economic Planning Sciences*, 11(1), 13-17.
- Bondy, J. A., & Murty, U. S. R. (1976). *Graph theory with applications* (Vol. 290): Citeseer.
- Borthakur, D. (2007). The hadoop distributed file system: Architecture and design. *Hadoop Project Website*, 11(2007), 21.
- Calera, A., Campos, I., Osann, A., D'Urso, G., & Menenti, M. (2017). Remote sensing for crop water management: from ET modelling to services for the end users. *Sensors*, 17(5), 1104.
- Caliński, T., & Harabasz, J. (1974). A dendrite method for cluster analysis. *Communications in Statistics-theory and Methods*, 3(1), 1-27.
- Cavallaro, G., Riedel, M., Bodenstern, C., Glock, P., Richerzhagen, M., Goetz, M., & Benediktsson, J. A. (2015). *Scalable developments for big data analytics in remote sensing*. Paper presented at the Geoscience and Remote Sensing Symposium (IGARSS), 2015 IEEE International.

- Cavallaro, G., Riedel, M., Richerzhagen, M., Benediktsson, J. A., & Plaza, A. (2015). On understanding big data impacts in remotely sensed image classification using support vector machine methods. *IEEE Journal of Selected Topics in Applied Earth Observations and Remote Sensing*, 8(10), 4634-4646.
- Chang, Y.-W., Hsieh, C.-J., Chang, K.-W., Ringgaard, M., & Lin, C.-J. (2010). Training and testing low-degree polynomial data mappings via linear SVM. *Journal of Machine Learning Research*, 11(Apr), 1471-1490.
- Chavez, P. S. (1996). Image-based atmospheric corrections-revisited and improved. *Photogrammetric engineering and remote sensing*, 62(9), 1025-1035.
- Chen, X., & Zhou, L. (2015). *The remote sensing image segmentation mean shift algorithm parallel processing based on MapReduce*. Paper presented at the International Conference on Intelligent Earth Observing and Applications 2015.
- Ching, A. (2013). Scaling apache giraph to a trillion edges. *Facebook Engineering blog*, 25.
- Chuang, C.-C., Su, S.-F., Jeng, J.-T., & Hsiao, C.-C. (2002). Robust support vector regression networks for function approximation with outliers. *IEEE Transactions on Neural Networks*, 13(6), 1322-1330.
- CMDC. (2020). Retrieved from <http://data.cma.cn/data/>
- Congalton, R. G., & Green, K. (2002). *Assessing the accuracy of remotely sensed data: principles and practices*: CRC press.
- Conway, J. (1970). The game of life. *Scientific American*, 223(4), 4.
- Cortes, C., & Vapnik, V. (1995). Support-vector networks. *Machine learning*, 20(3), 273-297.
- Crave, A., & Davy, P. (2001). A stochastic “precipiton” model for simulating erosion/sedimentation dynamics. *Computers & Geosciences*, 27(7), 815-827.
- Dahal, K. R., & Chow, T. E. (2015). Characterization of neighborhood sensitivity of an irregular cellular automata model of urban growth. *International Journal of Geographical Information Science*, 29(3), 475-497.
- Dattilo, G., & Spezzano, G. (2003). Simulation of a cellular landslide model with CAMELOT on high performance computers. *Parallel Computing*, 29(10), 1403-1418.
- Dean, J., & Ghemawat, S. (2008). MapReduce: simplified data processing on large clusters. *Communications of the ACM*, 51(1), 107-113.
- Dewan, A. M., & Yamaguchi, Y. (2009). Land use and land cover change in Greater Dhaka, Bangladesh: Using remote sensing to promote sustainable urbanization. *Applied geography*, 29(3), 390-401.
- Di Gregorio, S., Kongo, R., Siciliano, C., Sorriso-Valvo, M., & Spataro, W. (1999). Mount Ontake landslide simulation by the cellular automata model SCIDDICA-3. *Physics and Chemistry of the Earth, Part A: Solid Earth and Geodesy*, 24(2), 131-137.
- Douglas, E. M., Niyogi, D., Frohling, S., Yeluripati, J. B., Pielke Sr, R. A., Niyogi, N., . . . Mohanty, U. (2006). Changes in moisture and energy fluxes due to agricultural land use and irrigation in the Indian Monsoon Belt. *Geophysical Research Letters*, 33(14).

- Ermentrout, G. B., & Edelstein-Keshet, L. (1993). Cellular automata approaches to biological modeling. *Journal of theoretical Biology*, *160*(1), 97-133.
- Fan, Wang, Y., & Wang, Z. (2008). Temporal and spatial change detecting (1998–2003) and predicting of land use and land cover in Core corridor of Pearl River Delta (China) by using TM and ETM+ images. *Environmental monitoring and assessment*, *137*(1-3), 127.
- Fan, X., Lang, B., Zhou, Y., & Zang, T. (2017). *Adding network bandwidth resource management to Hadoop YARN*. Paper presented at the Information Science and Technology (ICIST), 2017 Seventh International Conference on.
- Fang, F., Zhang, T., Pavlidis, D., Pain, C., Buchan, A., & Navon, I. (2014). Reduced order modelling of an unstructured mesh air pollution model and application in 2D/3D urban street canyons. *Atmospheric Environment*, *96*, 96-106.
- Feddema, J. J., Oleson, K. W., Bonan, G. B., Mearns, L. O., Buja, L. E., Meehl, G. A., & Washington, W. M. (2005). The importance of land-cover change in simulating future climates. *science*, *310*(5754), 1674-1678.
- Ferro-Famil, L., Pottier, E., & Lee, J. (2001). *Unsupervised classification and analysis of natural scenes from polarimetric interferometric SAR data*. Paper presented at the IGARSS 2001. Scanning the Present and Resolving the Future. Proceedings. IEEE 2001 International Geoscience and Remote Sensing Symposium (Cat. No. 01CH37217).
- Foley, J. A., DeFries, R., Asner, G. P., Barford, C., Bonan, G., Carpenter, S. R., . . . Gibbs, H. K. (2005). Global consequences of land use. *science*, *309*(5734), 570-574.
- Fonstad, M. A. (2006). Cellular automata as analysis and synthesis engines at the geomorphology–ecology interface. *Geomorphology*, *77*(3-4), 217-234.
- Fotheringham, A. S. (1981). Spatial structure and distance-decay parameters. *Annals of the Association of American Geographers*, *71*(3), 425-436.
- Frery, A. C., & Perciano, T. (2013). Image data formats and color representation. In *Introduction to Image Processing Using R* (pp. 21-29): Springer.
- Friedl, M. A., McIver, D. K., Hodges, J. C., Zhang, X., Muchoney, D., Strahler, A. H., . . . Cooper, A. (2002). Global land cover mapping from MODIS: algorithms and early results. *Remote Sensing of Environment*, *83*(1-2), 287-302.
- Friedman, J. H. (1989). Regularized discriminant analysis. *Journal of the American statistical association*, *84*(405), 165-175.
- Frisch, U., Hasslacher, B., & Pomeau, Y. (1986). Lattice-gas automata for the Navier-Stokes equation. *Physical review letters*, *56*(14), 1505.
- Gafurov, A., & Bárdossy, A. (2009). Cloud removal methodology from MODIS snow cover product. *Hydrology and Earth System Sciences*, *13*(7), 1361-1373.
- Giachetta, R. (2015). A framework for processing large scale geospatial and remote sensing data in MapReduce environment. *Computers & Graphics*, *49*, 37-46.
- Gladkova, I., Grossberg, M. D., Shahriar, F., Bonev, G., & Romanov, P. (2012). Quantitative restoration for MODIS band 6 on Aqua. *IEEE Transactions on Geoscience and Remote Sensing*, *50*(6), 2409-2416.

- Gobron, S., Çöltekin, A., Bonafos, H., & Thalmann, D. (2011). GPGPU computation and visualization of three-dimensional cellular automata. *The Visual Computer*, 27(1), 67-81.
- Goodchild, M. F., Guo, H., Annoni, A., Bian, L., de Bie, K., Campbell, F., . . . Jackson, D. (2012). Next-generation digital earth. *Proceedings of the National Academy of Sciences*, 109(28), 11088-11094.
- Gorelick, N., Hancher, M., Dixon, M., Ilyushchenko, S., Thau, D., & Moore, R. (2017). Google Earth Engine: Planetary-scale geospatial analysis for everyone. *Remote Sensing of Environment*, 202, 18-27.
- Gropp, W. D., & Lusk, E. (2007). *Using MPI-2: A problem-based approach*. Paper presented at the European Parallel Virtual Machine/Message Passing Interface Users' Group Meeting.
- Guan, D., Li, H., Inohae, T., Su, W., Nagaie, T., & Hokao, K. (2011). Modeling urban land use change by the integration of cellular automaton and Markov model. *Ecological modelling*, 222(20-22), 3761-3772.
- Guller, M. (2015). *Big data analytics with Spark: A practitioner's guide to using Spark for large scale data analysis*: Springer.
- Haasdonk, B. (2005). Feature space interpretation of SVMs with indefinite kernels. *IEEE Transactions on pattern analysis and machine intelligence*, 27(4), 482-492.
- Hajra, B. (2014). A review of some recent studies on buoyancy driven flows in an urban environment. *International Journal of Atmospheric Sciences*, 2014.
- Haklay, M., & Weber, P. (2008). Openstreetmap: User-generated street maps. *IEEE Pervasive Computing*, 7(4), 12-18.
- Halmy, M. W. A., Gessler, P. E., Hicke, J. A., & Salem, B. B. (2015). Land use/land cover change detection and prediction in the north-western coastal desert of Egypt using Markov-CA. *Applied geography*, 63, 101-112.
- Han, M., & Daudjee, K. (2015). Giraph unchained: Barrierless asynchronous parallel execution in pregel-like graph processing systems. *Proceedings of the VLDB Endowment*, 8(9), 950-961.
- Hansen, W. G. (1959). How accessibility shapes land use. *Journal of the American Institute of planners*, 25(2), 73-76.
- He, C., Okada, N., Zhang, Q., Shi, P., & Li, J. (2008). Modelling dynamic urban expansion processes incorporating a potential model with cellular automata. *Landscape and Urban Planning*, 86(1), 79-91.
- He, Y., Lee, E., & Warner, T. A. (2017). A time series of annual land use and land cover maps of China from 1982 to 2013 generated using AVHRR GIMMS NDVI3g data. *Remote Sensing of Environment*, 199, 201-217.
- Heppenstall, A. J., Crooks, A. T., See, L. M., & Batty, M. (2011). *Agent-based models of geographical systems*: Springer Science & Business Media.
- Huang, Q., Yang, C., Benedict, K., Chen, S., Rezgui, A., & Xie, J. (2013). Utilize cloud computing to support dust storm forecasting. *International Journal of Digital Earth*, 6(4), 338-355.
- Huang, W., Meng, L., Zhang, D., & Zhang, W. (2017). In-memory parallel processing of massive remotely sensed data using an apache spark on hadoop

- yarn model. *IEEE Journal of Selected Topics in Applied Earth Observations and Remote Sensing*, 10(1), 3-19.
- Hutchinson, M. F., & Xu, T. (2004). Anusplin version 4.2 user guide. *Centre for Resource and Environmental Studies, The Australian National University, Canberra*, 54.
- Iovine, G., D'Ambrosio, D., & Di Gregorio, S. (2005). Applying genetic algorithms for calibrating a hexagonal cellular automata model for the simulation of debris flows characterised by strong inertial effects. *Geomorphology*, 66(1-4), 287-303.
- Ippoliti, C., Candeloro, L., Gilbert, M., Goffredo, M., Mancini, G., Curci, G., . . . Quaglia, M. (2019). Defining ecological regions in Italy based on a multivariate clustering approach: A first step towards a targeted vector borne disease surveillance. *PloS one*, 14(7).
- Islaker, H., Anastasiadis, A., Vassiliadis, D., & Vlahos, L. (1998). Solar flare cellular automata interpreted as discretized MHD equations. *Astronomy and Astrophysics*, 335, 1085-1092.
- Jjumba, A., & Dragicevic, S. (2015). Integrating GIS - based geo - atom theory and voxel automata to simulate the dispersal of airborne pollutants. *Transactions in GIS*, 19(4), 582-603.
- Kaufmann, R. K., & Stern, D. I. (1997). Evidence for human influence on climate from hemispheric temperature relations. *Nature*, 388(6637), 39.
- Kononova, M. a. M. (1961). Soil organic matter, its nature, its role in soil formation and in soil fertility. *Soil organic matter, its nature, its role in soil formation and in soil fertility*.
- Lambin, E. F., Turner, B. L., Geist, H. J., Agbola, S. B., Angelsen, A., Bruce, J. W., . . . Folke, C. (2001). The causes of land-use and land-cover change: moving beyond the myths. *Global environmental change*, 11(4), 261-269.
- Lan, H., & Xie, Y. (2013). A semi-ellipsoid-model based fuzzy classifier to map grassland in Inner Mongolia, China. *ISPRS journal of photogrammetry and remote sensing*, 85, 21-31.
- Lan, H., Zheng, X., & Torrens, P. M. (2018). Spark Sensing: A Cloud Computing Framework to Unfold Processing Efficiencies for Large and Multiscale Remotely Sensed Data, with Examples on Landsat 8 and MODIS Data. *Journal of Sensors*, 2018.
- Li, M., Zang, S., Zhang, B., Li, S., & Wu, C. (2014). A review of remote sensing image classification techniques: The role of spatio-contextual information. *European Journal of Remote Sensing*, 47(1), 389-411.
- Li, X., Shen, H., Li, H., & Zhang, L. (2016). Patch matching-based multitemporal group sparse representation for the missing information reconstruction of remote-sensing images. *IEEE Journal of Selected Topics in Applied Earth Observations and Remote Sensing*, 9(8), 3629-3641.
- Li, Y., Gong, J., Song, Y., Liu, Z., Ma, T., Liu, H., . . . Yu, Y. (2015). Design and key techniques of a collaborative virtual flood experiment that integrates cellular automata and dynamic observations. *Environmental Earth Sciences*, 74(10), 7059-7067.

- Liaw, A., & Wiener, M. (2002). Classification and regression by randomForest. *R news*, 2(3), 18-22.
- Lin, J. (2012). Network analysis of China's aviation system, statistical and spatial structure. *Journal of Transport Geography*, 22, 109-117.
- Liu, J., Kuang, W., Zhang, Z., Xu, X., Qin, Y., Ning, J., . . . Yan, C. (2014). Spatiotemporal characteristics, patterns, and causes of land-use changes in China since the late 1980s. *Journal of Geographical Sciences*, 24(2), 195-210.
- Liu, Y., & Zheng, Y. F. (2005). *One-against-all multi-class SVM classification using reliability measures*. Paper presented at the Proceedings. 2005 IEEE International Joint Conference on Neural Networks, 2005.
- Lunga, D., Gerrand, J., Yang, L., Layton, C., & Stewart, R. (2020). Apache Spark Accelerated Deep Learning Inference for Large Scale Satellite Image Analytics. *IEEE Journal of Selected Topics in Applied Earth Observations and Remote Sensing*, 13, 271-283.
- Lynch, C. (2008). Big data: How do your data grow? *Nature*, 455(7209), 28.
- Ma, L., Li, M., Ma, X., Cheng, L., Du, P., & Liu, Y. (2017). A review of supervised object-based land-cover image classification. *ISPRS Journal of Photogrammetry and Remote Sensing*, 130, 277-293.
- Ma, Y., Wu, H., Wang, L., Huang, B., Ranjan, R., Zomaya, A., & Jie, W. (2015). Remote sensing big data computing: Challenges and opportunities. *Future Generation Computer Systems*, 51, 47-60.
- Marques, R., Feijo, B., Breitman, K., Gomes, T., Ferracioli, L., & Lopes, H. (2013). A cloud computing based framework for general 2D and 3D cellular automata simulation. *Advances in Engineering Software*, 65, 78-89.
- McFeeters, S. K. (1996). The use of the Normalized Difference Water Index (NDWI) in the delineation of open water features. *International journal of remote sensing*, 17(7), 1425-1432.
- McGuffie, K., Henderson-Sellers, A., Zhang, H., Durbidge, T., & Pitman, A. (1995). Global climate sensitivity to tropical deforestation. *Global and Planetary change*, 10(1-4), 97-128.
- McLachlan, G. J. (2004). *Discriminant analysis and statistical pattern recognition* (Vol. 544): John Wiley & Sons.
- Melgani, F. (2006). Contextual reconstruction of cloud-contaminated multitemporal multispectral images. *IEEE Transactions on Geoscience and Remote Sensing*, 44(2), 442-455.
- Mendes, R. L., Santos, A. A., Martins, M., & Vilela, M. (2001). Cluster size distribution of cell aggregates in culture. *Physica A: Statistical Mechanics and its Applications*, 298(3-4), 471-487.
- Meng, X., Bradley, J., Yavuz, B., Sparks, E., Venkataraman, S., Liu, D., . . . Owen, S. (2016). Mllib: Machine learning in apache spark. *The Journal of Machine Learning Research*, 17(1), 1235-1241.
- Mineter, M. J., Dowers, S., & Gittings, B. M. (2000). Towards a HPC framework for integrated processing of geographical data: encapsulating the complexity of parallel algorithms. *Transactions in GIS*, 4(3), 245-261.

- Mitsova, D., Shuster, W., & Wang, X. (2011). A cellular automata model of land cover change to integrate urban growth with open space conservation. *Landscape and Urban Planning*, 99(2), 141-153.
- Molnar Jr, F., Szakaly, T., Meszaros, R., & Lagzi, I. (2010). Air pollution modelling using a Graphics Processing Unit with CUDA. *Computer Physics Communications*, 181(1), 105-112.
- Narteau, C., Zhang, D., Rozier, O., & Claudin, P. (2009). Setting the length and time scales of a cellular automaton dune model from the analysis of superimposed bed forms. *Journal of Geophysical Research: Earth Surface*, 114(F3).
- Openshaw, S. (1984). *The Modifiable Areal Unit problem*. CATMOG 38. Norwich.
- OpenStreetMap. (2020). Retrieved from <https://www.openstreetmap.org/>
- Overmars, K. d., De Koning, G., & Veldkamp, A. (2003). Spatial autocorrelation in multi-scale land use models. *Ecological modelling*, 164(2-3), 257-270.
- Pielke Sr, R. A., Pitman, A., Niyogi, D., Mahmood, R., McAlpine, C., Hossain, F., . . . Fall, S. (2011). Land use/land cover changes and climate: modeling analysis and observational evidence. *Wiley Interdisciplinary Reviews: Climate Change*, 2(6), 828-850.
- Pontius Jr, R. G., & Chen, H. (2006). GEOMOD modeling. *Clark University*.
- Qin, C. Z., Zhan, L. J., & Zhu, A. (2014). How to apply the geospatial data abstraction library (GDAL) properly to parallel geospatial raster I/O? *Transactions in GIS*, 18(6), 950-957.
- Qiu, G., Kandhai, D., & Sloot, P. (2007). Understanding the complex dynamics of stock markets through cellular automata. *Physical Review E*, 75(4), 046116.
- Radenski, A. (2013). *Using MapReduce streaming for distributed life simulation on the cloud*. Paper presented at the Artificial Life Conference Proceedings 13.
- Rao, S., Pachauri, S., Dentener, F., Kinney, P., Klimont, Z., Riahi, K., & Schoepp, W. (2013). Better air for better health: Forging synergies in policies for energy access, climate change and air pollution. *Global Environmental Change*, 23(5), 1122-1130.
- Rouse Jr, J. W., Haas, R., Schell, J., & Deering, D. (1974). Monitoring vegetation systems in the Great Plains with ERTS.
- Salcido, A., & Celada, A. (2010). A lattice gas approach to the Mexico City wind field estimation problem. *Modelling, Simulation and Optimization*, 385-416.
- Salvaris, M., Dean, D., & Tok, W. H. (2018). Microsoft AI Platform. In *Deep Learning with Azure* (pp. 79-98): Springer.
- Sanderson, D. (2009). *Programming google app engine: build and run scalable web apps on google's infrastructure*: " O'Reilly Media, Inc."
- Satellite Imaging Corporation. (2017a). QuickBird Satellite Sensor. Retrieved from <https://www.satimagingcorp.com/satellite-sensors/quickbird/>
- Satellite Imaging Corporation. (2017b). WorldView-4 Satellite Image Gallery. Retrieved from <https://www.satimagingcorp.com/gallery/worldview-4/>
- Shangguan, B., & Yue, P. (2018). *SPARK Processing of Computing-Intensive Classification of Remote Sensing Images: The Case on K-Means Clustering Algorithm*. Paper presented at the 2018 26th International Conference on Geoinformatics.

- Shen, H., Li, X., Cheng, Q., Zeng, C., Yang, G., Li, H., & Zhang, L. (2015). Missing information reconstruction of remote sensing data: A technical review. *IEEE Geoscience and Remote Sensing Magazine*, 3(3), 61-85.
- Shen, H., Zeng, C., & Zhang, L. (2011). Recovering reflectance of AQUA MODIS band 6 based on within-class local fitting. *IEEE Journal of Selected Topics in Applied Earth Observations and Remote Sensing*, 4(1), 185-192.
- Shvachko, K., Kuang, H., Radia, S., & Chansler, R. (2010). *The hadoop distributed file system*. Paper presented at the 2010 IEEE 26th symposium on mass storage systems and technologies (MSST).
- Singh, S. K., Mustak, S., Srivastava, P. K., Szabó, S., & Islam, T. (2015). Predicting spatial and decadal LULC changes through cellular automata Markov chain models using earth observation datasets and geo-information. *Environmental Processes*, 2(1), 61-78.
- Suez Canal Authority. (2013). New Suez Canal. Retrieved from <https://www.suezcanal.gov.eg/English/About/SuezCanal/Pages/NewSuezCanal.aspx>
- Sun, Z., Chen, F., Chi, M., & Zhu, Y. (2015). *A spark-based big data platform for massive remote sensing data processing*. Paper presented at the International Conference on Data Science.
- Theobald, D. M., & Hobbs, N. T. (1998). Forecasting rural land-use change: a comparison of regression-and spatial transition-based models. *Geographical and Environmental Modelling*, 2, 65-82.
- Tomlin, A. S., Ghorai, S., Hart, G., & Berzins, M. (1999). 3D adaptive unstructured meshes for air pollution modelling. *Environmental Management and Health*.
- Tong, C., Wu, J., Yong, S.-p., Yang, J., & Yong, W. (2004). A landscape-scale assessment of steppe degradation in the Xilin River Basin, Inner Mongolia, China. *Journal of Arid Environments*, 59(1), 133-149.
- Townshend, J., Justice, C., Li, W., Gurney, C., & McManus, J. (1991). Global land cover classification by remote sensing: present capabilities and future possibilities. *Remote Sensing of Environment*, 35(2-3), 243-255.
- Tseng, D.-C., Tseng, H.-T., & Chien, C.-L. (2008). Automatic cloud removal from multi-temporal SPOT images. *Applied Mathematics and Computation*, 205(2), 584-600.
- Turner, W., Spector, S., Gardiner, N., Fladeland, M., Sterling, E., & Steininger, M. (2003). Remote sensing for biodiversity science and conservation. *Trends in ecology & evolution*, 18(6), 306-314.
- U.S. Department of the Interior, & U.S. Geological Survey. (2018, 03/28/18). Landsat 8. Retrieved from <https://landsat.usgs.gov/landsat-8>
- U.S. Geological Survey. (2011, December 13, 2011). Sensors - Hyperion. Retrieved from <https://eo1.usgs.gov/sensors/hyperion>
- Valiant, L. G. (1990). A bridging model for parallel computation. *Communications of the ACM*, 33(8), 103-111.
- Viera, A. J., & Garrett, J. M. (2005). Understanding interobserver agreement: the kappa statistic. *Fam med*, 37(5), 360-363.
- Von Neumann, J. (1951). The general and logical theory of automata. *1951*, 1-41.

- Wang, J., Kwan, M.-P., & Ma, L. (2014). Delimiting service area using adaptive crystal-growth Voronoi diagrams based on weighted planes: a case study in Haizhu District of Guangzhou in China. *Applied Geography*, *50*, 108-119.
- Wang, L., Ma, Y., Yan, J., Chang, V., & Zomaya, A. Y. (2018). pipsCloud: High performance cloud computing for remote sensing big data management and processing. *Future Generation Computer Systems*, *78*, 353-368.
- Wang, N., Chen, F., Yu, B., & Qin, Y. (2020). Segmentation of large-scale remotely sensed images on a Spark platform: A strategy for handling massive image tiles with the MapReduce model. *ISPRS journal of photogrammetry and remote sensing*, *162*, 137-147.
- Wang, Z., Zhong, J., Lan, H., Wang, Z., & Sha, Z. (2017). Association analysis between spatiotemporal variation of net primary productivity and its driving factors in inner mongolia, china during 1994–2013. *Ecological Indicators*.
- Wang, Z., Zhong, J., Lan, H., Wang, Z., & Sha, Z. (2019). Association analysis between spatiotemporal variation of net primary productivity and its driving factors in inner mongolia, china during 1994–2013. *Ecological indicators*, *105*, 355-364.
- Weeden, A. (2013). Parallelization: Conway's game of life. In: Online.
- Weng, Q. (2002). Land use change analysis in the Zhujiang Delta of China using satellite remote sensing, GIS and stochastic modelling. *Journal of environmental management*, *64*(3), 273-284.
- White, S. H., Del Rey, A. M., & Sánchez, G. R. (2007). Modeling epidemics using cellular automata. *Applied Mathematics and Computation*, *186*(1), 193-202.
- White, T. (2012). *Hadoop: The definitive guide*: " O'Reilly Media, Inc."
- Wilder, B. (2012). *Cloud architecture patterns: using microsoft azure*: " O'Reilly Media, Inc."
- Wolf-Gladrow, D. A. (2000). 5. Lattice Boltzmann Models. In *Lattice Gas Cellular Automata and Lattice Boltzmann Models* (pp. 159-246): Springer.
- Wolfram, S. (2002a). *A new kind of science* (Vol. 5): Wolfram media Champaign, IL.
- Wolfram, S. (2002b). *A new kind of science*, vol. 5. *Wolfram media Champaign*, *80*.
- Wolfram, S. (2018). *Cellular automata and complexity: collected papers*: CRC Press.
- Wulder, M. A., & Coops, N. C. (2014). Make Earth observations open access: freely available satellite imagery will improve science and environmental-monitoring products. *Nature*, *513*(7516), 30-32.
- Xie, Y., Zhang, Y., Lan, H., Mao, L., Zeng, S., & Chen, Y. (2018). Investigating long-term trends of climate change and their spatial variations caused by regional and local environments through data mining. *Journal of Geographical Sciences*, *28*(6), 802-818.
- Yang, C., Yu, M., Hu, F., Jiang, Y., & Li, Y. (2017). Utilizing cloud computing to address big geospatial data challenges. *Computers, Environment and Urban Systems*, *61*, 120-128.
- Ye, B., & Bai, Z. (2007). *Simulating land use/cover changes of Nenjiang County based on CA-Markov model*. Paper presented at the International Conference on Computer and Computing Technologies in Agriculture.

- Zahran, E.-S. M., Smith, M. J., & Bennett, L. D. (2013). 3D visualization of traffic-induced air pollution impacts of urban transport schemes. *Journal of computing in civil engineering*, 27(5), 452-465.
- Zeng, C., Shen, H., & Zhang, L. (2013). Recovering missing pixels for Landsat ETM+ SLC-off imagery using multi-temporal regression analysis and a regularization method. *Remote Sensing of Environment*, 131, 182-194.
- Zhang, C., Li, W., & Travis, D. (2007). Gaps - fill of SLC - off Landsat ETM+ satellite image using a geostatistical approach. *International journal of remote sensing*, 28(22), 5103-5122.
- Zhang, J., Clayton, M. K., & Townsend, P. A. (2011). Functional concurrent linear regression model for spatial images. *Journal of Agricultural, Biological, and Environmental Statistics*, 16(1), 105-130.
- Zhang, Y., Gao, Q., Gao, L., & Wang, C. (2012). imapreduce: A distributed computing framework for iterative computation. *Journal of Grid Computing*, 10(1), 47-68.
- Zhao, Y., Billings, S. A., & Coca, D. (2009). Cellular automata modelling of dendritic crystal growth based on Moore and von Neumann neighbourhoods. *International Journal of Modelling, Identification and Control*, 6(2), 119-125.
- Zhao, Y., Billings, S. A., Coca, D., Ristic, R., & DeMatos, L. (2009). Identification of the transition rule in a modified cellular automata model: the case of dendritic NH₄Br crystal growth. *International Journal of Bifurcation and Chaos*, 19(07), 2295-2305.
- Zhu, X. (2012). *The impact of agricultural irrigation on land surface characteristics and near surface climate in China*.