Cloud Computing and Its Applications in GIS

Cao Kang

May 2011

A DISSERTATION

Submitted to the faculty of Clark University, Worcester, Massachusetts, in partial fulfillment of the requirements for The degree of Doctor of Philosophy in the Department of Geography

And accepted on the recommendation of

J. Ronald Eastman, PhD
Chief Instructor
ACADEMIC HISTORY

Name (in Full): Kang, Cao
Date: May, 2011

Baccalaureate Degree: Information Engineering, L.L.B.
Source: Wuhan University
Date: July, 2002

Other degrees, with dates and sources:
N/A

Occupation and Academic Connection since date of baccalaureate degree:
Ph.D. candidate in Geography Department of Clark University
DISSERTATION COMMITTEE

J. Ronald Eastman, Ph.D.
Chief Instructor

Samuel Ratick, Ph.D.
Committee Member

Robert Gilmore Pontius Jr., Ph.D.
Committee Member

Craig Wills, Ph.D.
Committee Member
ACKNOWLEDGEMENTS

I wish to thank Professor J. Ronald Eastman, for his support of this research and for his patience and guidance through my graduate study. I also wish to thank Professor Samuel Ratick, Professor Robert Gilmore Pontius Jr., Professor Craig Wills, Professor B. L. Turner, and Professor Anthony Bebbington for their support and advice.
# TABLE OF CONTENTS

<table>
<thead>
<tr>
<th>Section</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>List of Figures</td>
<td>vi</td>
</tr>
<tr>
<td>List of Tables</td>
<td>viii</td>
</tr>
<tr>
<td>Introduction</td>
<td>1</td>
</tr>
<tr>
<td>Chapter 1</td>
<td>8</td>
</tr>
<tr>
<td>Chapter 2</td>
<td>37</td>
</tr>
<tr>
<td>Chapter 3</td>
<td>60</td>
</tr>
<tr>
<td>Conclusion</td>
<td>84</td>
</tr>
<tr>
<td>Reference</td>
<td>91</td>
</tr>
<tr>
<td>Figure</td>
<td>Page</td>
</tr>
<tr>
<td>--------</td>
<td>------</td>
</tr>
<tr>
<td>Figure 1</td>
<td>3</td>
</tr>
<tr>
<td>Figure 2</td>
<td>15</td>
</tr>
<tr>
<td>Figure 3</td>
<td>26</td>
</tr>
<tr>
<td>Figure 4</td>
<td>30</td>
</tr>
<tr>
<td>Figure 5</td>
<td>31</td>
</tr>
<tr>
<td>Figure 6</td>
<td>33</td>
</tr>
<tr>
<td>Figure 7</td>
<td>39</td>
</tr>
<tr>
<td>Figure 8</td>
<td>41</td>
</tr>
<tr>
<td>Figure 9</td>
<td>42</td>
</tr>
<tr>
<td>Figure 10</td>
<td>44</td>
</tr>
<tr>
<td>Figure 11</td>
<td>47</td>
</tr>
<tr>
<td>Figure 12</td>
<td>48</td>
</tr>
<tr>
<td>Figure 13</td>
<td>50</td>
</tr>
<tr>
<td>Figure 14</td>
<td>51</td>
</tr>
<tr>
<td>Figure 15</td>
<td>52</td>
</tr>
<tr>
<td>Figure 16</td>
<td>55</td>
</tr>
<tr>
<td>Figure 17</td>
<td>56</td>
</tr>
<tr>
<td>Figure 18</td>
<td>58</td>
</tr>
<tr>
<td>Figure 19</td>
<td>66</td>
</tr>
<tr>
<td>Figure 20</td>
<td>71</td>
</tr>
<tr>
<td>Figure 21</td>
<td>73</td>
</tr>
<tr>
<td>Figure 22</td>
<td>74</td>
</tr>
</tbody>
</table>
LIST OF TABLES

Table 1 33
Table 2 65
Table 3 66
Introduction

The emergence of cloud computing provides many opportunities for academia, the information technology (IT) industry and the global economy as an information technology revolution. Compared to other distributed computing paradigms such as Grid computing and High Performance Computing (HPC), cloud computing provides broader interoperabilities over the world-wide web networks. As IT industry leaders such as Google, IBM and Amazon are striving to promote this innovative computing paradigm, it is reasonable to expect that cloud computing will bring profound changes to every aspect of the IT industry and to various sectors of the economy.

GIS, as a powerful spatial analysis tool, has being successfully used in many fields. However, today’s Geographic Information Systems also face many challenges. Among these challenges, there are two issues that need special attention for the further development of GIS applications: 1. data acquisitions, which are usually expensive and time-consuming; 2. software and hardware needed to run GIS applications, which are usually expensive and require professional knowledge to setup and use. The process of acquiring, storing and pre-processing data are time consuming, expensive and often lead to unnecessary data replications. Spatial analyses often require expensive software and computer systems. Furthermore, for the general public, it is not practical for a non-professional occasional user to acquire several gigabytes of data and to spend thousands of dollars on professional GIS software. Therefore, the question is, is there a solution to solve these two problems?
Fortunately, the development of cloud computing provides us with a solution to host large volumes of data and to provide powerful computing services in the cloud computing infrastructures.

Cloud computing, is an innovative network-based distributed computing paradigm that promises us that “we won’t compute on local computers, but on centralized facilities operated by third-party compute and storage utilities” [2] Compared to other distributed computing paradigms over the network, cloud computing has a few advantages that distinguish it from other similar paradigms

“Cloud computing offers a scalable virtual infrastructure to users and developers. It gives the illusion of an unlimited resource for computation and data storage. Since it allows users to start small and increase computation or storage resources only when they need, it provides users with access to large amounts of “hardware resources” in a short time interval without requiring the users’ fixed investments or maintenance costs for expensive hardware.” [5]

The cloud is formed and used in a loose bundled architecture with certain abstractions the end users do not need to learn or understand how to install software or gather data to run applications by using cloud computing based web services, software developers do not need to worry about the limitations or the hardware specifications of the “computers” they are working on because these “computers” have a unified interface and specification provided by cloud computing hosting companies.

A typical cloud is comprised of four layers (see Figure 1) a fabric layer, a unified resource layer, a platform layer, and an application layer. The fabric layer is the physical hardware such as CPUs, memories, hard drives, etc. The second layer is the unified resource layer that wraps the
underlying physical hardware layer. By abstracting the physical hardware into resources units, cloud computing providers can provide more flexible and robust computing and storage services to users. Depending on a provider’s service type, the third layer of cloud computing can either be a platform layer or an application layer. The platform layer is built on top of the unified resource layer. The platform layer usually is comprised of virtual machines with pre-installed operating systems. Applications built on top of the platform layer have fewer controls over the supporting hardware, but the platform layer will provide isolation between applications. This isolation can provide better security and prevent the whole system from crashing. The last layer is the applications layer. All the end-user applications are built on this layer. The applications layer is also the only layer that the end users will be able to see. This four layer structure provides a robust and flexible environment for developers and users [2].

![Cloud Computing Architecture](image_url)

**Figure 1:** Cloud Computing architecture from Foster, Zhao, Raicu, etc, 2008. [2].
Cloud computing has many significant advantages over other types of computing paradigms. For example, cloud computing uses hardware resources more efficiently and effectively: cloud computing allows different users to use resources of the cloud infrastructure at different times, which leads to less system idle time, i.e., less waste of resources; cloud computing providers will be able to charge users for the resources they used, and the rate might be different at different times of a day. For example, the rate can be higher during the day time office hours and cheaper at night during off-office hours when there are fewer user demands. This practice may encourage the users to use the cloud computing services at different times to satisfy different requirements and to cut costs. Adoption of a business model like this will lead to the maximal use of the cloud computing resources, which will allow cloud computing providers to further cut costs due to economies of scale, and to offer lower service rates consequently.

Cloud computing can also provide parallel computation support to applications. By adopting parallel computing methods such as MPI and Map/Reduce, applications built on top of cloud computing are able to process large amounts of data in parallel [14].

Because of these advantages, there have been several experiments of hosting scientific research and computation over the cloud computing infrastructures [14] [22] [23] [26]. These early experiments indicate that: despite some technical issues such as long latency in message passing over the network, it is promising to host this type of research on cloud computing infrastructure to replace expensive high-performance supercomputers [14].

The cloud architecture offers an easy way to share data and computing power over the internet. By providing various web services to end users, anybody with an internet-connected computer
can query, edit and manipulate large volumes of data stored in servers hosted by different parties without buying expensive professional GIS software or facing a steep learning curve to understand how to use it. By avoiding the downloading, storing and pre-processing duplicated data, efficiencies can be obtained in the sharing of data over the cloud. From the perspective of a GIS service developer or distributor (including data and operations), all data and software used are only stored on servers, it is easier and more efficient to revise the program and the data on the server without providing data revision or releasing software patches in a traditional medium such as CD-ROM.

There are several challenges that, however, need to be solved before a GIS system can fully benefit from the adoption of cloud computing as its underlying computing infrastructure. The first challenge is how to re-design and migrate truly-spatial GIS algorithms to cloud computing to take advantages of the parallel computing mechanisms provided by cloud computing infrastructures. The second challenge is how to host large volumes of GIS data in cloud computing infrastructures to provide efficient and reliable data services such as queries. These questions will be addressed in the three articles described below.

The format of this dissertation is one consisting of three papers suitable for submission to professional journals. These three papers are developed to answer one general research question: how to develop a cloud based GIS system? Due to the large scope of this topic, raster GIS will be the primary research topic and each article will selectively focus on one of three key components of the above general question: 1. what cloud computing is and what cloud computing based GIS systems can offer; 2. challenges of re-designing truly spatial GIS algorithms for cloud computing; 3.
how to efficiently store large volumes of GIS data in a cloud computing infrastructure. Although
the focus has been set on raster GIS, many of the insights gained during the research process will
also be valuable for the development of cloud computing based vector GIS systems.

The first article serves as the review and introduction component of the dissertation. This article
will address what cloud computing is and why it is beneficial to build cloud computing based GIS
systems. It will review cloud computing technology and compare it with similar distributed
computing technologies such as Grid computing. Advantages and disadvantages of each
distributed computing technology will be analyzed. Potential applications of cloud computing in
GIS will be discussed. The implications of building cloud computing based GIS systems will be
proposed. Three initial architectural designs for cloud computing based GIS systems will also be
introduced.

The second article focuses on algorithm-related issues for cloud computing based GIS systems. It
will examine the problems of migrating stand-alone truly spatial GIS algorithms to distributed
computing infrastructures by designing a cloud computing compatible parallel minimum
Euclidean distance calculation algorithm, which is a commonly used truly spatial algorithm. The
purposes of this article are to understand: 1. the mechanisms and limitations of traditional stand­
alone minimum Euclidean distance calculation algorithms; 2. the possible advantages of the
distributed computing based distance algorithm; 3. the challenges and possible solutions of
parallelizing traditional truly spatial algorithms. The lessons learned in this process will not only
help to build a cloud computing compatible minimum Euclidean distance algorithm, but also will
help to develop other cloud computing based truly spatial GIS algorithms.
The third article will tackle data storage related issues for cloud computing based GIS systems. Rather than using traditional Relational Database Management Systems (RDBMS), most cloud computing infrastructures adopted distributed computing compatible NoSQL Distributed Database Management Systems (NDDBMS). How to store GIS data in this type of cloud computing compatible database so that these data can be served quickly and efficiently is challenging. An innovative NDDBMS GIS data storage schema will be a vital piece to solve this puzzle. Due to the large extent of this topic, the third article will focus on designing a data storage schema for storing only raster GIS data in NDDBMS. Commonly used data will be analyzed. Data storage models will be presented. Indexing strategies will be introduced. Some optimization will also be discussed.
Chapter 1: Cloud Computing and Its Applications in GIS

Abstract
Cloud Computing has become increasingly popular in recent years. As a novel computing paradigm, Cloud Computing can fundamentally change the ways that data are stored and shared and how computing is conducted. More and more applications and services are being created based on Cloud Computing technology every day. On the other hand, Cloud Computing is still unfamiliar to most members of the GIS community. Its capabilities and potential applications related to GIS have not been extensively discussed. This article introduces Cloud Computing to the GIS community by examining the definition and architecture of Cloud Computing. Advantages of Cloud Computing such as low entry barriers for public users, friendly to large volumes of data, high availability, and strong scalability are discussed in detail. Potential issues of Cloud Computing such as privacy and data security, inflexibility to switch, and data lock-in are investigated. This article consequently proposes a definition of Cloud Computing based GIS systems. Features of Cloud Computing based GIS systems are further analyzed. Three cloud-based GIS system architectures, public cloud-based GIS, private cloud-based GIS, and hybrid cloud-based GIS, are also proposed.

Keywords
Cloud Computing, GIS, Web GIS, Internet GIS, Grid Computing, Web Services
1. Introduction

Cloud Computing is a novel computing term that has recently emerged. Strongly supported by Google, Sun, IBM, Amazon and many other information technology industry leaders, it brings many innovative concepts and substantial potential to the information technology industry. Cloud Computing offers an interoperable way of providing and sharing services such as computing and data storage over the internet. Unlike many similar yet different distributed computing paradigms such as: Grid Computing [2] which applies the resources of many loosely coupled, heterogeneous, and geographically dispersed computers in a network to a single problem at the same time but usually implements strict admission and access control policies over its member users, and High Performance Computing (HPC) [14] which usually uses expensive supercomputers and computer clusters to solve advanced computation problems, Cloud Computing takes advantage of economy of scale by using massive numbers of commodity computers to distribute computing power and data storage over networks, offering easy access to these resources by massive numbers of public users. With flexible configurations, Cloud Computing allows users to dynamically increase or decrease hardware resources allocated for various services depending on the user’s real-time demands. Adopting the concept of Utility Computing [16] which delivers computing resources such as computational power and storage space to users, as a metered service similar to traditional public utilities such as water or electricity, Cloud Computing also provides a business model for monitoring and billing parties who use services provided by the cloud providers. Further, hardware virtualization built inside of the cloud infrastructure makes Cloud Computing viable from a business perspective.
Geographic Information System (GIS) applications often involve acquiring and processing data from multiple sources followed by intensive spatial computations provided by expensive computer systems. The exact same data are hosted in different locations and need to be processed the same way many times when used by different parties; and in many cases, in order to process or conduct spatial analysis over these data would require expensive investments in hardware, software and training of personnel. The network-based nature of Cloud Computing may provide a better solution to share resources and cut unnecessary costs and time over many GIS applications.

Certain Cloud Computing based map services have also appeared in the market. Google Maps and Google Earth are two popular cloud-based map products. They provide fast and user friendly map query and display functionalities to massive numbers of public users. The backbones of these two systems are supported by Google’s cloud infrastructure, namely Google File System [6] and Google Big Table [20]. This is an important and innovative step to bring Cloud Computing to GIS. However, according to DeMers’ definition of GIS, a GIS system should include “a data manipulation and analysis subsystem that performs tasks on the data, aggregates and disaggregates, estimates parameters and constraints, and performs modeling functions” [24], which Google Maps and Google Earth clearly lack. Thus they are not true GIS systems. A well-built cloud-based GIS system should: 1. take the benefits of Cloud Computing such as being friendly to massive amounts of data, parallelized high performance computing, ease of sharing data and the capability of supporting large numbers of users; 2. provide full functionality sets that a GIS system should have such as customizable spatial analysis and modeling. A cloud based GIS should be able
to provide not only functionalities or services such as data display and query, but also data editing and spatial analysis.

Despite these potential advantages, GIS presents a challenge for a Cloud Computing based platform. The extremely large data volumes could make data display and data downloading/uploading between the cloud infrastructure and end users extremely slow. In addition, the algorithmic nature of truly spatial processes presents potential challenges for the scalability of resources. There were some earlier discussions and research focusing on GIS in similar distributed computing paradigms [12] [13] [15] [19], but there are major differences among them. The applications of these earlier distributed computing-based GIS systems such as Grid-based GIS systems or HPC-based systems are limited due to many factors, such as very expensive hardware or overly strict user policies. Conversely, this paper focuses on Cloud Computing which offers alternatives to provide high performance using inexpensive hardware and offers easy access to massive numbers of public users.

The absence of true cloud-based GIS systems and development of Cloud Computing technology has brought a great opportunity to the GIS community. To better understand Cloud Computing and the possible developments it may bring to the GIS community, this paper will focus on reviewing Cloud Computing technologies and propose some prototype Cloud Computing-based GIS system designs that can be used as the basis for an evaluation of the feasibility of running a cloud-based GIS. This article will include: an introduction to Cloud Computing in section 2; the concept and pros and cons of cloud-based GIS in section 3; architecture design for a cloud-based GIS system in section 4 and potential challenges in section 5.
2. A New Era of Cloud Computing

2.1. The Definition of Cloud Computing

Although the basic idea of Cloud Computing is not really new [5], considering the relatively young age of the new term Cloud Computing, it is not surprising to see little consensus among experts on its definition [1]. Foster and Zhao define Cloud Computing as “A large-scale distributed computing paradigm that is driven by economies of scale, in which a pool of abstracted, virtualized, dynamically-scalable, managed computing power, storage, platforms, and services are delivered on demand to external customers over the Internet” [2]. Wang and Lazewski’s definition is “A computing Cloud is a set of network enabled services, providing scalable, Quality of Service (QoS) QoS guaranteed, normally personalized, inexpensive computing platforms on demand, which could be accessed in a simple and pervasive way” [21]. Armbrust, Fox and Griffith define Cloud Computing as follows:

“Cloud Computing refers to both the applications delivered as services over the Internet and the hardware and systems software in the datacenters that provide those services. The services themselves have long been referred to as Software as a Service (SaaS), so we use that term. The datacenter hardware and software is what we will call a Cloud” [5].

An additional definition might be a network shared scalable resource pool that can provide services such as computing power and/or storage space in abstracted and encapsulated forms, which are delivered through networks with proper levels of isolation, dynamic resource allocation and usage monitoring, to authorized users based on users’ demands.

Several key attributes need to be noticed:

- First, Cloud Computing is delivered in the form of abstracted and encapsulated services. That is to say, the users will not know nor need to know which computer in the cloud
infrastructure is providing services to them. Users will not know nor need to know technical details of these services such as what programming language these services were written in or which version they are. Users will only need to know what services they want and where to get them. Unlike similar network shared computing paradigms, the services delivered by Cloud Computing provide certain amounts of isolation by adopting virtualization [2] [4]. For example, virtualization may create multiple operating systems and storage spaces on the same piece of hardware. Each user of these virtualized operating systems or storage space will not even notice other users are using the same hardware. This provides security, application and fault isolations among virtualization users.

- Second, the resource pool that Cloud Computing is built upon is dynamically scalable. This means that a cloud user can easily increase or decrease the resources he/she needs based on his/her demand and the cloud infrastructure itself can be expanded or deflated easily. The popular usage of commodity computers in building up a cloud infrastructure decreases the cost of Cloud Computing and thus makes it more affordable to public users.

- Third, Cloud Computing based services are delivered through networks, including internet, intranet, or both.

- Fourth, adopting the concept of Utility Computing [16] [17] [18], Cloud Computing providers are able to monitor the usage of each cloud user and charge the user based on service usage. Unlike Grid Computing which consists of heterogeneous administrative domains and complicated access verification mechanisms, Cloud Computing adopts
homogeneous administrative management systems and unified access verifying mechanisms that make Cloud Computing easy to monitor user usages. This feature enables Cloud Computing with a larger and more flexible user management system compared to Grid Computing and other network shared computing paradigms [2].

2.2. Components and Architecture of Cloud Computing
A number of Cloud Computing providers have emerged in recent years. Amazon’s Elastic Cloud Computing (EC2) and Simple Storage System (S3), Google App Engine, Microsoft’s Azure and SalesForce.com are all examples of how popular this new computing model is becoming. Even though these cloud suppliers all provide Cloud Computing services to users, they in fact represent different cloud architectures. Meanwhile, although the architectures, technologies and setups used by these cloud providers are different in many ways, they all share some common components.

2.2.1. Architectures of Cloud Computing
There are various versions of Cloud Computing architecture schematics [2] [5]. One widely discussed architecture is that of Foster et al [2], which consists of four layers. These are: 1. the Fabric layer, 2. the Unified resource layer, 3. the Platform layer, and 4. the Application layer.
The fabric layer represents the actual raw hardware which composes the entire Cloud Computing infrastructure. Nowadays, many cloud suppliers choose commodity computers as their major hardware form. The logic behind the popular adoption of commodity computers is to offer the best performance per cost unit [6].

The unified resource layer is the first service layer above the fabric layer. It provides cloud users with abstracted/encapsulated resources such as unified file systems, virtual computers and databases so that they can provide services to upper service layers or end users.

The platform layer builds on top of the unified resource layer. It provides tools, middleware and services to developers for development and deployment. The services that the platform layer provides can also be considered to be an Application Program Interface (API) for the cloud in most cases.
The application layer is the top layer in the Cloud Computing architecture. This layer provides applications and services which can be executed in the cloud by end users. The applications and services containing in this layer are built based on lower layers such as the platform layer or the unified resource layer.

2.3. Cloud service categories

Today, Cloud Computing offers many different services ranging from simple web log processing to heavy scientific computation. In general, most Cloud Computing services fall into one of the following three categories [2][3]:

**Infrastructure as a Service (IaaS):** Provides computing infrastructure, such as encapsulated virtual servers, virtual desktops, storage units and network resources and are delivered as services over networks. One example is Amazon’s Elastic Cloud Computing (EC2) Service and Simple Storage Service (S3) which started their services in 2006 for the public. A typical EC2 service to the user is creating and hosting virtual machines over the cloud. Once the VM (virtual machine) is created, a user can perform operations with it like a normal desktop computer [2][4].

**Platform as a Service (PaaS):** Provides an API for users so they can develop and deploy applications and/or services based on cloud provider platforms. A typical example is Google App Engine. Users can build and deploy applications and services by using the Google App Engine API. Google App Engine may scale users’ applications or services up and down automatically based on certain criteria such as the intensity of user demands.

**Software as a Service (SaaS):** Delivers software over networks to end users. Since this type of service provides features of specific software, SaaS usually only provides functionalities to satisfy
specific needs. Salesforce is an example of a SaaS provider. It provides CRM (Customer Relationship Management) software and services to end users based on a usage-based pricing model.

2.3. Cloud Computing: Advantages and Disadvantages
There are both advantages and disadvantages of adopting Cloud Computing.

2.3.1. Advantages of Cloud Computing
There are many advantages of Cloud Computing. The following ones are especially noteworthy:

- Scalability: Users are able to scale resources allocated for applications up or down dynamically and flexibly. Rather than acquiring additional infrastructure such as hardware and software to support applications when demand is high, users can cut the lead time by quickly scaling up in a cloud environment while avoiding the risk of idling the infrastructure when demand is low. Consider what happens in a non-cloud based scenario – a business just releases a service that only 50 people are aware of and it is not sure if this service is going to become popular. Hence, it would not make much sense for it to invest in a high-performance server to host this service. When the service turns out to be a success, more and more users are going to use this service. So this business has to keep up with the users’ pace by upgrading its servers/clusters. But, after a while, other competitors of this business may release similar yet better services, and now all users are leaving to those new comers. This business now has expensive servers/clusters sitting in the server room doing nothing but consuming electricity. The waste is obvious. What would happen if this business used a cloud computing infrastructure? Taking the scalability advantage of a cloud infrastructure, this business could scale up and down its
service capability dynamically to accommodate its demand without any hardware investment or waste.

- **Availability**: Cloud Computing is built on a massive shared infrastructure which typically resides in multiple physical locations [7]. One faulty part in a cloud infrastructure can be reset or replaced quickly by backup hardware. This makes cloud based services highly available, effectively cutting costly downtime and recovery.

- **Reliability**: Since major cloud providers redundantly back up data stored in their cloud infrastructure [7] [8] [9], cloud based services are highly reliable compared to in house clusters. For example, Microsoft Azure keeps backup copies in different data centers. Should the main copy be destroyed, this backup remains accessible. Other cloud providers such as Google even keep three copies of data in the infrastructure [6].

- **Parallelization**: A user may create hundreds of services / applications in a massive cloud infrastructure undertaking parallel computing as in a HPC (High Performance Computer). There have been meaningful experiments of running scientific computations in the cloud [14] [22] [23] [26]. This type of computing usually requires expensive HPC’s, but running these types of computing in the cloud can significantly decrease the cost of acquiring and maintaining expensive HPC’s.

- **Economy of Scale**: Compared to in-house computing infrastructures such as data center clusters or HPC’s, Cloud Computing offers a better economy of scale because of its massive scale. Discounted utility, discounted hardware/software purchase prices and
dedicated administration personnel are the benefits of this economy of scale. Considering all the maintenance costs, utility costs and upgrade costs, a user may end up spending much more if he/she chooses an in-house data center compared to cloud based services.

- More efficient application/data update model: In most Desktop systems, software updating is not an easy task. Update patches have to be created, distributed and installed by end users or IT administrators. The updating process could take days or even weeks. In a cloud-based GIS system, users will not even notice the update of applications in most cases. Everything is done on the cloud server side. This makes updating in Cloud-based GIS a breeze compared to the Desktop-based GIS system. When it comes to data updating, Cloud-based GIS only needs to update data on the cloud server side and all users will have the most up-to-date data.

- Easy to share/distribute: A while ago, the most popular way to share or distribute map data through the internet was through ArcIMS. Today, many new tools such as MapServer and GeoServer have emerged to share data through an intranet or the internet. But none of them can provide the massive network throughput like a cloud infrastructure which focuses on serving massive numbers of public internet users from the very beginning. The Cloud infrastructure is pushing the ease of sharing and distributing data further thanks to its network-based nature.

- Data volume-friendly: Cloud Computing is built on a massive shared infrastructure which is capable of handling petabytes of data. Many cloud-oriented computing frameworks have also been created to handle large amounts of data [11] [25].
Better use of resources: In a traditional business scenario, only certain computers in a company run at high demand; the rest of them only get high demand occasionally such as during business hours while idling most of the time after business hours. This is not only a waste of electricity, but also a waste of hardware and software. By offering different prices during different time periods in a day, some users may prefer to use computing powers in the cloud to perform time-insensitive processing after peak business hours for its lower prices [5]. The ideal result is that most computers in a cloud infrastructure can run at high demand all the time.

The following disadvantages of Cloud Computing are those most frequently cited [5] [10]:

- Privacy and data security: This would be a major concern to anyone who is considering putting their data into a third-party storage space. After all, users will not know exactly where their data are going to be stored and who would have the privilege to access it inside the cloud infrastructure. Some Cloud Computing providers may allocate their data centers in different countries. The data stored in these countries may face different regulations and are thus protected or unprotected according to the local governments.

- Performance stabilities: Although, in most cases, performance in a cloud is quite stable, there are times when the cloud infrastructure is under heavy loads. This may be observed at certain times in a business day typically during the business hours. The performance for an application in a cloud environment will depend on network traffic and the resources other virtual machines running on the same physical machine as the application's virtual
machine are taking. It can be difficult to repeat the performance demonstrated from one run to another [14].

- Inflexibility to switch: Due to the fact that different cloud providers may offer different levels of services (e.g. Google App Engine provides PaaS, Amazon EC2 provides IaaS, Salesforce provides SaaS) and APIs, it may be difficult for a user to switch from one cloud provider to another. This is also known as the “lock-in” problem in other literatures [5]. In general, IaaS provides easier possibilities of switching than PaaS while SaaS is the most difficult one to switch to or from.

- Network speed: There are two network speeds that need to be mentioned here: one is intra-cloud network speed and the other is inter-cloud speed. Intra-cloud network speed is the network transfer speed inside the cloud infrastructure. It is usually limited by switches in the cloud infrastructure network. Depending on the cloud infrastructure network setup, this intra-cloud network speed may vary. For some data or message passing intensive applications, the latency might not be acceptable [14]. Inter-cloud speed refers to the network speed between the cloud provider and the user or among different cloud providers. This network speed is limited largely by public internet speed. This speed may affect the users’ experiences and interactivities between cloud providers and users.

- Data lock-in: Taking GIS applications as an example, a typical GIS application usually involves a great deal of data. The data format and how data are stored may produce a large performance difference to the GIS application. Furthermore, data import and export in a GIS application may be very important in some cases. As was mentioned in section 2,
certain cloud providers tend to provide a lock-in data structure by providing non-transparent exclusive data storage methods to users. This may cause serious inconveniences in GIS systems particularly.

3. Cloud Computing Based GIS
The internet development has brought many possibilities and opportunities to traditional GIS. Many GIS systems have released various levels of internet oriented GIS systems such as ESRI's ArcGIS Server, Google Earth and Bing Map. Today, the emergence of Cloud Computing has further introduced more exciting opportunities to the GIS community.

3.1. Definition of Cloud-based GIS
Before we can define what a cloud-based GIS is, it is of value to review the general nature of current GIS systems. Based on how the GIS systems deploy/distribute their services and user bases, we can classify the GIS systems into the following three categories: Desktop GIS, Client/Server GIS and Public Internet GIS.

3.1.1. Types of GIS
**Desktop GIS:** Computing and storage units are located in the end users’ computers. ESRI’s ArcGIS Desktop 9.3, Clark Labs IDRISI and MapInfo can all be classified into this category. No network is necessary for this type of GIS system.

**Client/Server GIS:** Most storage space resides on the server side while computing can happen on both the server and/or the client depending on the operations. This type of GIS usually requires at least an intranet. The users are from a specific group that is authorized to use resources on a specific server. Users use client machines to display data and perform queries and other types of
editing. When this is completed, users submit the changes back to the server. Typical examples are ArcIMS with ArcGIS Desktop and ArcGIS Server with ArcGIS Explorer. Even though this type of GIS system can sometimes deliver data and services via the internet, it is not built for operation by massive numbers of public users due to its hardware and software constraints.

**Public Internet GIS:** Both computing and storage units reside on the server side. GIS systems in this category often only provide data visualization and limited operations such as various queries. This type of GIS system usually focuses on massive numbers of public users. Google Maps, Google Earth and Bing Map are examples of public internet systems that support some of the functions of a GIS, though they lack the necessary analytical components.

**Distributed GIS:** GIS systems that implement any type of distributed computing paradigm can be called distributed GIS systems. Such computing paradigms may include Grid Computing, peer-to-peer computing, Cloud Computing and High Performance Computing [12]. However, a truly distributed GIS system still does not exist yet.

By understanding these different systems, a tentative definition of Cloud-based GIS can be postulated:

* A GIS system that builds on top of a Cloud Computing infrastructure, using the cloud infrastructure to dynamically scale its computing and/or storage capabilities, providing parallelized services that are able to serve various user bases including authorized users and/or massive numbers of public users; these services should include basic GIS related functionalities such as reclass, overlay and so forth, as well as a comprehensive set of truly spatial GIS analysis functionalities such as cost distance, watershed, runoff, etc.
The services mentioned in the definition above include not only data display and data query, but also data editing and spatial analysis.

Although the definitions of different types of GIS above are quite self-explanatory, it is necessary to make some distinctions among these definitions. Cloud-based GIS is a subset of distributed GIS. The major difference between a cloud-based GIS and other types of distributed GIS such as GRID-based GIS and HPC-based GIS lie in the hardware and protocols they built on top of, as well as user bases. HPC uses expensive hardware whereas Cloud Computing uses inexpensive commodity computers; GRID uses more strict access control and internal protocols whereas Cloud Computing uses relatively loose access control and public protocols [2]; both HPC and GRID based GIS are not designed to serve massive numbers of public users while cloud-based GIS is.

The differences between a public internet GIS and a cloud-based GIS mostly reside in the function sets and the underlying infrastructures they use. A cloud-based GIS system uses a distributed cloud computing infrastructure which offers more data throughput compared to stand-alone web servers. In general, cloud-based GIS systems are able to offer more data and computationally intensive services compared to non-cloud-based internet GIS systems due to infrastructural differences. In fact, some public internet mapping systems are already cloud-based such as Google Maps and Google Earth.

Landscape of current cloud-based GIS systems

With the development of cloud computing, some GIS systems have been released and are claimed to be cloud-based GIS systems. Examples of these systems include Google Earth Engine, cloud-
based ArcGIS Server and giscloud.com. However, as the time of writing, none of these GIS systems
can really be called cloud-based GIS systems according to the definition proposed in Section 3.1.2.

Currently, Google Earth Engine can provide a huge amount of distributively stored GIS data
collections. It also provides certain levels of operations for satellite imagery data. But there is no
concrete evidence showing that it is capable of executing truly spatial GIS algorithms such as cost
distance calculation due to the fact that it does not provide the type of file system accesses
needed for those algorithms.

ESRI provides several cloud-based GIS solutions: ArcGIS Server on Amazon EC2, ArcGIS.com,
ArcLogistics and Business Analyst Online (BAO). ArcGIS Server on Amazon EC2 and ArcGIS.com are
based on one similar architecture (showed in Figure 3), which is running ArcGIS Server on the
virtual machines on Amazon EC2 cloud. The ArcGIS Server software installed on Amazon EC2 is
the same as the single microcomputer based ArcGIS Server software. The scalabilities are
achieved by creating more virtual machines with ArcGIS Server pre-installed on Amazon EC2. This
architecture certainly provides good scalability in terms of how many individual servers can be
deployed at the same time; however, these servers cannot provide as good scalability in terms of
performance. All the ArcGIS servers on Amazon EC2 are running individually, which means it lacks
a central management system that parallelizes tasks automatically or shifts working load
dynamically. Therefore, this architecture may provide excellent accessibility, but cannot provide
as excellent performance.
ArcLogistics and Business Analyst Online (BAO) are in the category of software as a service (SaaS) provided by ESRI. ArcLogistics can provide users with optimized routes and schedules based on multiple factors. BAO is a web-based service that provides reports and maps based on locations and other forms of data from various sources such as demographic, consumer spending, and business data. The cloud-based ArcLogistics runs on Amazon EC2, while BAO runs in the ESRI data center [45]. Both ArcLogistics and BAO have on-premises versions, which can be installed in users' own data center. ArcLogistics and BAO may surely provide certain powerful GIS query and analysis functionalities, but they cannot be categorized as comprehensive cloud-based GIS systems according to the definition proposed in Section 3.1.2 since they only provide quite limited types of GIS analysis functionalities.

giscloud.com also claims to be a cloud based GIS system. Users can upload, edit, convert, create and visualize GIS data through their internet browsers by using the services provided by
giscloud.com. It also provides certain spatial analysis abilities such as hotspot and buffer analysis. However, due to the facts that it does not explicitly explain how it works as a cloud-based system, there is no evidence to support that it has the ability to take advantage of the advanced features of cloud computing such as distributed data storage and parallel computing.

3.2. Why Cloud-based GIS

3.2.1 Advantages of cloud-based GIS
Cloud based GIS may obtain many benefits from what a Cloud Computing infrastructure offers in general. These benefits may include better scalability, reliability, availability, parallelization, economy of scale, more efficient update models, ease to share/distribute, large data volume friendly, etc.

Specific to GIS, adopting a Cloud Computing infrastructure for GIS may bring a specific advantage: lower barriers to entry and more service oriented, particularly for occasional users. Extreme upfront costs on hardware and software are usually associated with GIS systems. Cloud based GIS systems give easy access to users. Unlike most Desktop or C/S GIS (Client/Server GIS) systems, a cloud-based GIS system has no requirements for upfront investment in hardware or software. A user could just upload his or her data into the cloud-based GIS, run applications and pay-per-use. All he or she needs is a credit card to pay for that service.

Another specific advantage of cloud-based GIS system is its ability to provide data services without giving away the actual data. For example, there are certain high resolution household-level census data that cannot be distributed to researchers due to privacy concerns. When researchers request such data, certain extrapolations have to be conducted to provide them with
coarser resolution block-level census data. This practice has resulting in inconveniences to researchers especially when they need perform analyses based on household-level data. By taking the advantages of cloud-based GIS, researchers can make requests of conducting analyses of household-level data on the servers of Census Bureau, and then Census Bureau can deliver the results of these analyses to researchers without giving away the actual data to them.

### 2.2 Disadvantages of Cloud-based GIS

A Cloud Computing based GIS also share some general disadvantages from cloud computing. Special attentions are needed for the following issues: privacy and data securities, data lock-in and performance unpredictability. However, due to the large amounts of data that most GIS analyses involve, one particular disadvantage for Cloud-based GIS is the network speed bottleneck of Cloud Computing. In spite of the high speed networks that connect computing nodes inside Cloud Computing infrastructure, the speed of inbound and outbound networks that connect between users’ client computers and Cloud Computing’s infrastructure are largely limited by public Internet Service Providers (ISP). Occasionally, users may need to upload or download their own data to Cloud-base GIS systems. If in case, these data are large, it is going to take a long time to finish transferring these data.

### 4. Architecture Design of Cloud-based GIS

Depending on who provides services and where the data are, a cloud-based GIS system could take one of three basic forms: public cloud-based GIS, private cloud-based GIS and hybrid cloud-based GIS. Table 1 summarizes the pros and cons of these three forms which are discussed in detail below.
4.1. **Public Cloud-Based GIS**

In a public cloud-based GIS system, services are created, maintained and delivered as software or platform (SaaS or PaaS) by a Cloud GIS Vendor. Users execute all applications and store all data in a cloud-based GIS provider’s infrastructure. The cloud infrastructure is not inside users’ buildings and thus is not owned or managed by users.

Some common data such as public satellite imagery, public road and boundary maps or census data might be provided by the cloud vendor in cloud servers and would be shared by all authorized users. Cloud GIS vendors would be responsible for maintaining these data to keep them up-to-date and accurate.

A cloud GIS user may still have private data stored on the cloud server which will only be accessible to certain people defined by the user. If any cloud user wants to process or share any data, he/she will have to upload the data to the cloud server.

All applications in a public cloud-based GIS can only be executed on the cloud server side. End users may use or create applications compatible with the cloud server.

The advantages of adopting this structure are that: 1. Users may choose some applications already developed by the cloud GIS vendors so that they do not have to be experts in the GIS system. 2. Users pay less for some shared public data hosted by cloud vendors due to economy of scale. 3. Cloud GIS vendors will take care of all hardware, software and data related issues such as bug-fixing and upgrading. Hence it is maintenance-free to users.
The possible disadvantages of this type of cloud GIS are: 1. Users have less control over privacy and security related issues. 2. The data format used by the public cloud-based GIS might be exclusive to one GIS system only which makes all data locked in with this cloud-based GIS vendor, which is usual for most desktop GIS software.

![Public Cloud-based GIS system architecture](image)

**Figure 4: Public Cloud-based GIS system architecture**

### 4.2. Private Cloud-Based GIS
A private cloud-based GIS system is built, operated and maintained by a user. The hardware of a private cloud-based GIS can be an in-house-cloud or a virtual-cluster in a third party Cloud Computing providers' infrastructure. In either case, the user needs to create the cloud GIS servers (physical or virtual); deploy and distribute self-developed or third party-developed GIS applications; acquire and update GIS data and maintain hardware, software and data.
The major advantage and probably the most important reason why certain users may prefer this type of cloud GIS is because of concerns over security and privacy of data. For certain industries, the security and privacy of data are critical to users such as military organizations and medical institutions. In a public cloud GIS system, users share physical hardware with other users. It is difficult for a user to fully monitor the condition of its data storage. Furthermore, the locations of physical data stores are usually unknown to users. In contrast, if a user chooses an in-house-cloud GIS, then the user knows where the data are physically located and has full control over the hardware/network which makes it easier to monitor and defend the system which will help guard privacy and security of confidential data better.

The disadvantages of using this type of cloud-based GIS are obvious too: 1. Services are expensive to maintain. 2. Some data acquisition could be expensive compared to public cloud-based GIS due to a lack of economy of scale. 3. It has limited scalability compared to a public cloud based GIS system since users are less flexible to scale up or down.

Figure 5: Private Cloud-based GIS system architecture
4.3. Hybrid Cloud-Based GIS

A hybrid cloud-based GIS system refers to a GIS system that uses part of a public cloud-based GIS system and part of a private cloud-based GIS system. A reasonable design might be to host private or highly-confidential data in a private cloud-based GIS system and to host other data and applications in a public cloud-based GIS system. Or a user may want to create a small size private cloud-based GIS system to provide daily services and meanwhile use public cloud-based GIS to provide extra services once demands from users are beyond the private cloud-based GIS system’s capacity.

The advantages of this structure for users are: 1. Better control over privacy and security. 2. Benefit from public cloud-based GIS system’s scale of economy. 3. Better scalability compared to private cloud-based GIS. 4. Better reliability and availability -- depending on a GIS system’s set up, the public cloud part and the private cloud part can back each other up.

The biggest disadvantage of this hybrid cloud-based GIS system is probably its complexity. In order to make both public and private parts work, the user has to configure hardware/software, create services and manage data on both the public and private cloud infrastructures, and connect these two parts seamlessly.
Table 1 summarizes the pros and cons of the above three types of Cloud-based GIS based on the discussion above and reasonable expectations of the technologies involved.

<table>
<thead>
<tr>
<th>Cloud Type</th>
<th>Data Stored in</th>
<th>Reliability / Availability</th>
<th>Privacy</th>
<th>Performance</th>
<th>Cost</th>
<th>Sharing ability</th>
<th>Complexity / Ease of administration</th>
</tr>
</thead>
<tbody>
<tr>
<td>Public</td>
<td>Third-party Server</td>
<td>Good</td>
<td>potentially weak</td>
<td>Good</td>
<td>Low</td>
<td>Easy</td>
<td>Good</td>
</tr>
<tr>
<td>Private</td>
<td>In-house Server</td>
<td>Poor</td>
<td>Strong</td>
<td>Fair</td>
<td>High</td>
<td>Fair</td>
<td>Fair</td>
</tr>
<tr>
<td>Hybrid</td>
<td>Third-party Server and In-house server</td>
<td>Very Good</td>
<td>Strong</td>
<td>Depends on setup</td>
<td>Depends on setup</td>
<td>Easy</td>
<td>Poor</td>
</tr>
</tbody>
</table>

Table 1: Cloud-based GIS system architecture designs

5. Challenges
As mentioned above, cloud infrastructures are comprised of massive commodity computers. The way computations are executed and data are stored in a cloud infrastructure are drastically different from the way it is handled on desktop computers [6] [11]. Since in most cloud infrastructures data are stored across many commodity computers [6] and computation may be executed on multiple commodity computers in parallel, different strategies are required to handle
data storage, data flow and computing related issues in a cloud-based GIS. How to guarantee the performance, privacy and security of data are also problems that need to be solved. Some of the challenges such as performance guarantees and privacy are broadly shared by general Cloud Computing while the others are more specific to cloud-based GIS systems.

**Data storage:** To fully capitalize on the power of a cloud infrastructure, some new GIS data storage schemes specialized for cloud architecture are needed. For example, any large raster images could be partitioned into many smaller tiles and be stored in different places across the cloud infrastructure; if many users need to retrieve different parts of this raster image, the traffic load can be dispersed among the cloud infrastructure instead of overloading one computer. In this example, there is a trade-off in deciding how many pieces to partition the raster image into. Thus, it is up to the cloud-based GIS vendors to make an optimized decision. How to efficiently store, query and retrieve GIS data in a cloud-based GIS could be challenging.

**Data transfer speed:** Data transfer mentioned here includes both outside network data transfer and inside network data transfer. Outside network data transfer refers to data flow between the user’s computer and the cloud-based GIS system’s infrastructure such as data uploading and downloading. Inside network data transfer refers to data flow inside the cloud infrastructure. How to accelerate outside network data flow to offer a more interactive user experience is a challenge. The ideal result is that a user cannot tell if the GIS operation happens in his/her local machine or in a remote cloud infrastructure. Compared to outside network data transfer, inside network data transfer could be tougher to solve. In the current cloud infrastructure set up, inside network data transfer may become a potential bottleneck that limits the whole cloud performance [5]. Without
fast inside network data transfer, a user may need to wait hours to get some spatial analysis results.

**Parallel algorithms:** One distinct feature of Cloud Computing is its superior ability to conduct parallel computing. This feature would offer cloud-based GIS a huge performance improvement over traditional desktop GIS. In current GIS software, though, there is not much focus on design and implementation of parallelization of GIS related algorithms at the cluster level. In some instances, one quick fix to this type of problem could be splitting the data into subsets to make them compatible with the distributed nature of Cloud Computing infrastructure. However, truly spatial operations such as distance or cost distance require more than just splitting the data. To fully benefit from the cloud infrastructure’s parallel computing ability, some new GIS parallelized algorithms need to be developed and implemented. This is another challenge that needs to be addressed.

**Performance guarantee:** As mentioned before, performance in a cloud infrastructure is not very stable. For cloud-based GIS to work practically in the industry’s tough production environment, we need to guarantee the performance of cloud-based GIS system based on Quality of Service (QoS) agreements between cloud-based GIS vendors and end users. To develop such a QoS agreement could be complicated; to make the service performance comply with this QoS agreement would be even more difficult.

**Privacy and data security:** It is possible that some private or confidential data is in a GIS database. How to guarantee the safety and integrity of these data is another important question any cloud-based GIS system has to answer.
6. Conclusion
Based on the considerations above, the advantages of Cloud Computing can be summarized as:

low hardware and maintenance costs, low entry barriers for public users, friendly to large
volumes of data, high reliability, high availability, strong scalability, superior economy of scale,
and promising potential for high performance computing; however, Cloud Computing also has
some unresolved issues, which include privacy and data security, performances stabilities,
inflexibility to switch, network speed and data lock-in. Despite these issues, Cloud Computing still
offers a great opportunity for the future development of GIS systems. For instance, Cloud
Computing based GIS systems can provide users with huge amounts of pre-collected data and
high performance parallelized data analyses, while it only requires minimum investments from
the users to start (low barrier to entry). The three Cloud-based GIS system architectures proposed
in this article may serve as guidelines for colleagues who are interested in designing Cloud
Computing based GIS systems.
Chapter 2: A cloud computing compatible algorithm for the calculation of feature distance for raster GIS

Abstract
Calculating the shortest Euclidian distance with raster data is an essential capability for spatial analysis. Although several popular distance algorithms have served this purpose well for years, the increasing map layer size and demand for faster processing speed have led to new challenges for these traditional distance algorithms. This article proposes a novel parallel Euclidian distance algorithm that can operate on top of modern distributed systems such as cloud computing. Exploiting the advantages of distributed computation and storage, this algorithm can handle very large data sets and run dramatically faster than traditional Euclidian distance algorithms.

Key words: cloud computing, parallel computing, distributed computing, distance algorithm, distributed algorithm
1. Introduction
Calculating the shortest Euclidian distance from feature pixels to all non-feature pixels within a raster image is an essential capability for spatial analysis. Although several popular distance algorithms have served this purpose well for years, the increasing map layer size and demand for faster processing speed have presented new challenges for these traditional distance algorithms. Fortunately, the advancement of information technology has provided more powerful tools than ever before. Distributed computing systems such as cloud computing have successfully stored and served huge amounts of data to millions of people efficiently in recent years [6]. As building a Geographic Information System (GIS) on top of a distributed computing infrastructure is becoming increasingly promising, this article proposes a parallel minimum Euclidian distance algorithm that could fit well in a distributed computing infrastructure. Successful implementation of this algorithm on top of a modern distributed computing infrastructure such as cloud computing would greatly improve processing speed and eliminate the size limitation of some traditional distance algorithms.

2. Review of Distance Algorithms
There are three widely known algorithms for calculating the shortest Euclidian distance between non-feature pixels and feature pixels over raster-based data: exhaustive search, growth rings, and pushbroom. Each of these algorithms has its advantages and disadvantages.
2.1. Exhaustive Search
The exhaustive search method is self-evident but is not used in any published system because of its limitations. To calculate the nearest distance from any pixel to feature pixels, this approach uses row and column subscripts to calculate the Euclidian distance from the pixel to each feature pixel using the Pythagorean Theorem (Figure 7) [28]. This approach is easy to understand and implement. But the problem is that as the raster data dimension increases, the computational complexity increases exponentially. If we take a raster image with $n$ pixels as an example, in order to calculate the nearest features for these data, the distance between $n$ pixels and $n$ potential feature pixels have to be calculated, which yields a computational complexity of $O(n^2)$. This makes this approach an unrealistic solution for large raster data sets.

![Figure 7. Using the Pythagorean Theorem to calculate the nearest distance from a pixel (the circle in Figure 7) to feature pixels (the stars in Figure 7)](image)

2.2. Growth Rings
The second approach implements the growth rings technique [27]. The first step in the growth rings algorithm marks all feature pixels as 0 in distance. Next it calculates the distance for all pixels neighboring each feature pixel by propagation using Pythagorean geometry. Then the algorithm recursively repeats the process. Thus, through the whole raster image, distance is derived from the feature pixels in a series of concentric rings (Figure 8) until all pixels in the raster image are covered. The advantage of this algorithm is that it makes the computational complexity decrease...
to $O(n)$. For example, for most pixels in a raster image with $n$ pixels, the algorithm will search their 3 neighboring pixels for nearest distance to known pixels. Thus, approximately $3 \times n$ calculations are required to finish calculation, which yields a computational complexity of $O(n)$. The other distinctive advantage of this algorithm is its ability to calculate distance for a part of the raster image. This advantage is particularly useful for buffer analyses. However, there are two major problems with this algorithm. The first is that it suffers from image size limitations since it has to store the edges of growth rings either in memory or on a hard drive. If the edges of growth rings are to be stored in memory, in the worst case scenario, the whole raster image needs to be stored in memory, which is not feasible for real applications. If the edges of growth rings are to be stored on the hard drive, multiple random accesses to the stored file are required during calculation, which will greatly decrease the performance of this algorithm. The second problem is due to the fact that during the ring growth process, knowledge is only 1 layer of pixels deep, which means a pixel computes its distance to feature pixels based on only its immediate neighbor pixels. For example, in Figure 8, the shortest distance for the pixel on the first row, second column should be square root of 5 (which is 2.24). But according to the growth rings algorithm, its shortest distance is 2.41, which is calculated by adding 1.41 to the pixel on the second row, third column. This behavior in growth rings results in error propagation along the diagonal directions such that growth rings from a single point would appear as octagons rather than circles.
2.3. Pushbroom
The third algorithm is called the pushbroom procedure. Two similar yet different pushbroom algorithms were proposed by Eastman [27] and Tomlin [29] [32] respectively.

In Eastman’s original algorithm [27] there are four steps or passes. It starts from the top-left pixel of the raster image and goes along each row and then sequentially down the image from one row to the next. As the algorithm reads each pixel from the input raster image, a temporary file is generated. It records the horizontal distance (delta X) and vertical distance (delta Y) to the nearest feature pixel which is on top or on the left side, or on the upper-left diagonal of that pixel. A special flag value L (where L represents a very large value) is given to each pixel if no feature pixel has yet been detected. Once a feature pixel has been detected, delta X and delta Y to the feature pixel are carried along by incrementing the delta distance for each successive column and row. As long as a pixel is not a feature pixel, its delta X and delta Y are calculated from its nearest neighbor by comparing distance from delta X and delta Y values from the pixel to the left, the pixel diagonally above, and the pixel directly above. “In essence, the effect is one of determining
the lower-right quadrant of all growth rings in a single pass – a kind of ripple effect in which knowledge of feature positions is carried along in the pass.” [27]

![Figure 9. The result (square of distance) after the first step of the pushbroom algorithm. L represents the special flag value for a very large number. Only the lower-right quadrant of growth rings has been calculated. Three additional passes for the other quadrants are required to achieve the final result.](image)

After the first pass through the data (please see Figure 9), three similar passes are undertaken with different orientations. The second pass follows the same logic as the first but starting from bottom-right to top-left to decide the upper-left quadrant of the growth rings; the third pass repeats the routine but starts from top-right to bottom-left to decide the lower-left quadrant of the growth rings; the fourth pass then starts from bottom-left to top-right to decide the upper-right quadrant. After these four passes, distances will have been completely calculated.

Tomlin’s algorithm shares some similarities with Eastman’s algorithm. Four sweeps are also needed to calculate the distance for a pixel. However, the sweeps proceed from left to right, right to left, top to bottom and bottom to top. In Eastman’s algorithm, the calculation of the distance
of a pixel depends on its eight neighboring pixels. In Tomlin’s algorithm, the calculation of the distance of a pixel depends on its four neighbor pixels in four directions [30].

There are two distinctions between Eastman’s [27] and Tomlin’s [32] [33] algorithms. 1. Instead of calculating the square of delta X and delta Y between the feature pixels and the normal pixels, Tomlin’s algorithm uses a technique that decomposes a squared number into a series of sum of odd numbers. Figure 10 shows this process. This feature may greatly increase the calculation speed due to the fact that no squaring operations are needed during the process. 2. Although some minor revisions could be taken for both Eastman’s and Tomlin’s algorithms to reduce data traverses from four passes to two passes, Tomlin’s algorithm has to traverse the data in both horizontal (from column to column) and vertical (from row to row) directions while Eastman’s algorithm only needs to traverse the data in the horizontal direction [46]. The slightly revised version of Eastman’s algorithm scans the two rows we mentioned above from left to right and from right to left while these two rows are loaded in memory. This left to right and right to left process only happens when the algorithm traverses the data from top to bottom and from bottom to top. Thus, only two data traverses are needed for Eastman’s algorithm since it works based on 2 by 2 quadrants. However, for Tomlin's algorithm, it works based on four neighbor pixels from left, right, up and down. As a result, at least two hard drive traverses of the raster data are needed, one in the horizontal direction and another in the vertical direction. Especially, for the horizontal direction traverse, which consists of left to right and right to left traverses, a series of hard drive seek operations have to be carried out to fetch a column of pixels, or the entire image has to be read into memory for fast pixel retrievals. By effectively reducing the data
traverse times and hence hard drive traversal, Eastman's algorithm can significantly increase its performance.

![Incremental calculation of squared distance by sweeping](image)

Figure 10 "Incremental calculation of squared distance by sweeping. Each of a set of point locations is labeled according to the square of its distance to the nearest shaded sample point location (if any) due left. Note how the left-to-right sequences of squared distance values can be generated by incrementation." From C. D. Tomlin, 1987 [32]

There are several advantages of the pushbroom algorithms. First, they decrease the computational complexity to $O(n)$, because a raster image with $n$ pixels only needs $4 \times n$ calculations at most, it also decreases the need for large amounts of memory, since only two rows (for Eastman's algorithm) or two columns (only for Tomlin's algorithm) of data are stored in the memory at any time (the one being calculated and the previous one for propagation of distances). The disadvantage of this algorithm is that it must operate over the entire image. That is to say, the algorithm cannot produce distance results for only a part of the raster image. It has to generate either the result for the entire data set or none. This might be a disadvantage for certain Euclidian distance based spatial analyses that only need distance calculation for a small part of the entire image such as with buffer analyses.
3. The Need for Parallelizing the Euclidian Distance Algorithm
There are several substantial advantages we can obtain from parallelizing Euclidian distance calculations. A major limit for calculating Euclidian distance with raster data is the image size. Both the exhaustive search and growth rings algorithms suffer from it. The pushbroom algorithms are the most amenable to large volumes of data. But, when it comes to processing speed over huge raster images (hundreds of thousands of rows or columns), this traditional algorithm may be unacceptably slow.

On the other hand, with the increasing popularity of distributed computing systems, new opportunities have emerged to greatly improve the speed and reliability of GIS related operations. However, fully exploiting the advanced features of a distributed computing infrastructure is no easy task. Traditional algorithms that are migrated to distributed computing systems without redesign cannot take advantage of it. For example, the traditional algorithms can be run on a distributed computing system using only one computing node, but it would operate just like a desktop computer environment with all its inherent problems. Therefore, we need to find an innovative way to parallelize a Euclidian distance algorithm such that it can take advantage of the superior file access speed and higher reliability of distributed systems while improving processing speed and being amenable to the processing of huge data volumes.

4. Parallelized Euclidian Distance Algorithm (PEDA) - Description
The basic logic of the proposed distance algorithm is to cut the original raster image into smaller sub-images, then send these smaller sub-images to multiple computing nodes. Then each computing node individually calculates the Euclidian distance between non-feature pixels and
feature pixels for the sub-image assigned to it. When all computing nodes perform this procedure at the same time, the Euclidian distance will have been calculated in a parallel manner. Like the algorithms reviewed above, this distance algorithm produces results in projected map units.

In order to make each sub-image self-contained, which means it does not need any information from the source data (the full raster image) or its neighboring sub-images while calculating distance for its pixels, this type of path dependency needs to be cut. This algorithm calculates the distance for each pixel on the splitting edges that split the image into sub-images, even before the sub-image is generated. By doing this, sub-images no longer need information from their neighbors or the full source data when they calculate the pixels’ distance within. That is to say, in a distributed computing system, the computing nodes which host those sub-images need to access neither the original full source data nor the sub-images on other computing nodes. This can effectively reduce network traffic and the computational delay caused by transporting data through the networks.

To be more specific, this algorithm takes the following three steps: 1. Split the source raster data into sub-images; 2. Calculate the Euclidian feature distance for each sub-image; 3. Merge the results from step 2. Details of these procedures are described in the following sections.

4.1. Step 1: Splitting the source raster data and generating the sub-images
Based on user input or the source data volume, the preferred number of vertical and horizontal splitting edges can be decided (further discussions about the preferred number will be proposed in Section 6). Figure 11, shows an example of splitting an 11 x 11 raster image into four sub-
images. Each sub-image is a 6 by 6 raster data. Although we use square sub-images as an example here, the sub-image shape can be rectangular as well.

Figure 11. Input raster data is an 11 x 11 raster image. Each cell represents a pixel. The pixels in the red areas are splitting edges. The stars represent feature pixels. In this example, the 11 x 11 source data has been divided into 4 pieces of 6 x 6 blocks.

4.1.1. Calculate Euclidian distance for pixels on the edges:
Once the number of splitting edges has been decided (the optimized number of splitting edges will be discussed in Section 6), their locations can be allocated accordingly. With all pixels on splitting edges being allocated, the algorithm will calculate the shortest distance between splitting edge pixels to feature pixels. To make sure this algorithm can handle huge raster data with thousands of rows and columns, the following procedures are taken: first, read one line of source data at a time; second, search all pixels in that line for feature pixels; third, if any pixel on that line is a feature pixel, calculate the squared distance of this feature pixel to all known splitting-edge pixels by using the Pythagorean geometry equation between two points $\text{square}_\text{dist}= (X_1 - X_2)^2 + (Y_1 - Y_2)^2$; fourth, if the calculated distance is smaller than the previously calculated squared distance, the splitting-edge pixel will store the current distance along with the delta X and delta Y, which represents the column and row distances from this edge pixel to the current feature pixel.
respectively; read every line in the source data and perform the above steps (see Figure 12). At the end of the above procedure, each splitting-edge pixel will store its shortest distance to any known features as well as the delta X and delta Y between them.

Figure 12. Every pixel on splitting edges has found its shortest Euclidian distance to its nearest feature pixel. Delta X and delta Y to the nearest feature pixel are stored. For example, the value (2, 3) in the top left pixel represents the delta X and delta Y from this pixel to the nearest feature pixel.

4.1.2. Generate sub-images
After every splitting-edge pixel has found its shortest distance to feature pixels, the algorithm will generate the sub-images. Each pixel that is encompassed by these splitting edges will be read from the source image and sent to the new sub-images. As seen in Figure 12, four sub-images are generated; two six-pixel splitting edges and all those pixels that are encompassed by them (5 x 5 in Figure 12) comprise each sub-image; all feature pixels will also be included in sub-images.
Procedures in step 1 prepare sub-images for the pushbroom distance calculated in step 2. At the end of step 1, the algorithm will generate sub-images which cover the whole source data. These sub-images contain pixels read from source data and edge pixels which have stored calculated distance as well as delta X and delta Y to the nearest feature.

4.2. Step 2: Calculate the value of all pixels inside the sub-image

After step 1, sub-images will be assigned to different computing nodes. Once a computing node has a sub-image assigned to it (or sub-images, depending on settings), this algorithm will utilize a slightly modified version of the original pushbroom algorithm to calculate the shortest distance between each non-feature pixel and all feature pixels inside the sub-images.

The original pushbroom algorithm created four extra “virtual edges” that wrap the raster image and assigned these virtual edges with very large distance values to start the iterative calculation. The modified version of the original pushbroom algorithm uses the “actually calculated edges” for splitting edges derived from step 1. When the recursive pushbroom starts, the algorithm reads the first line inside the sub-image which is the neighbor to the top edge. The algorithm will calculate the shortest distance for each pixel (except for feature pixels) on this line based on that pixel’s three neighboring pixels, using delta X and delta Y calculated for the edge pixels from step 1 above. After it finishes, the algorithm reads the next line and uses pixels on the previous line to calculate the distance until a feature pixel is met during the calculation. When a feature pixel is met, similar to the original pushbroom algorithm, all of the pixel’s distances around this feature pixel will be updated. In Figure 13, the four steps show how the modified version of the pushbroom algorithm works for the top left sub-image of Figure 12. After four push procedures,
the shortest distance to the nearest feature pixel will have been determined for every pixel in the sub-image.

Step a: From top to bottom, left to right. The square of the current nearest distance is recorded.

Step b: From bottom to top, right to left. The square of the current nearest distance is recorded.

Step c: From top to bottom, right to left. The square of the current nearest distance is recorded.

Step d: From bottom to top, left to right. The nearest distance is recorded.

Figure 13. This diagram shows how a modified pushbroom algorithm works for a sub-image. Those two numbers separated by a comma on the edge pixels represents the delta X and delta Y to the nearest feature pixel from it. The step d shows the final result of this calculation. The red squares represent for splitting edges.
4.3. Step 3: Merge all sub-images
The last step of this algorithm is to merge all calculated sub-images from step 2 and generate the final shortest Euclidian distance result. Figure 14 shows the merged result of the 4 sub-images split from the 11 x 11 raster data we used as an example above.

Figure 14. Merged result of 4 sub-images calculation.

4.4. Why it works
In the original pushbroom algorithm, each pixel derives its distance to feature pixels strictly from its neighboring pixels. When we try to parallelize the Euclidian distance algorithm, we realize that as long as we can guarantee that the pixels get correct information from its neighboring pixels, it does not matter if the original input raster data is split into pieces or not. Thus we split the original raster data and wrap each split sub-image with a layer of pixels which already know their shortest distance to the nearest feature pixels. If there is no feature pixel within this group of wrapped pixels, each pixel can derive its distance from the wrapping layer (the splitting edges); if there are feature pixels within this group of wrapped pixels, each pixel can derive its distance by
using the wrapping layer and the feature pixels. In either situation, we can be certain that each pixel that has been wrapped inside the sub-images will have enough information to derive its correct nearest distance to features. By doing so, we have successfully transformed needs for global information to needs for local information.

5. Algorithm Result
From a comparative test, the result of this new algorithm is identical to the original pushbroom algorithm. As illustrated in Figure 15, we took a 500 x 500 raster image (Figure 15a) and split it into 100 51 x 51 sub-images. The split result is shown in Figure 15b. Figure 15c is the final result after merging all sub-images. Through an overlay analysis (not shown), it was verified that the results are identical.

Figure 15a. Feature raster data.

Figure 15b. Result from PEDA after calculating of the edges of the 100 sub-images.
6. Discussion and Optimization

6.1. **Number of Splitting Edges**

The number of splitting edges needs to be chosen well. If an algorithm user chooses to split the source data into many sub-images, it certainly would make each sub-image smaller and offers the possibility that more computing nodes could be used for calculation at the same time. However, splitting the source image into sub-images comes with a cost. In step 1 of the algorithm, it needs to calculate the shortest distance to the nearest feature pixels for each pixel on all splitting edges. In this procedure, the distance calculation for edge pixels is much more expensive than the pushbroom distance calculation, because the algorithm has to calculate the distance between each splitting-edge pixel and each feature pixel which would yield an $O(n^2)$ computational complexity. The total calculation time for the entire PEDA calculation can be described by the following equations:

$$T_{\text{Block}} = \frac{K_1 XY}{CR}$$
\[ T_{\text{Edge}} = K_2(X(R - 1) + Y(C - 1)) \]
\[ T_{\text{TotalTime}} = T_{\text{Block}} + T_{\text{Edge}} \]

where \( T_{\text{Block}} \) and \( T_{\text{Edge}} \) represent the time needed for calculating all blocks and all splitting edges respectively. \( T_{\text{TotalTime}} \) represents the total calculation time. \( X \) and \( Y \) represent the length and width of the rectangular source image respectively (\( X = 2000, Y = 500 \), if the image is 2000 x 500). \( C \) and \( R \) represent the number of blocks along the long edge and the short edge of the rectangular source image (\( C = 6, R = 4 \), if there are 6 x 4 blocks in total). And the \( K_1 \) and \( K_2 \) are constants that represent the calculation time needed for calculating a single block pixel and a single splitting-edge pixel respectively.

In order to conduct standardized performance evaluations for optimized block numbers, both raster images and blocks are assumed to be square in the following discussion. The equations above thus will have simplified forms as the following:

\[ T_{s\text{Block}} = K_1 X^2 / C^2 \]
\[ T_{s\text{Edge}} = 2K_2 X(C - 1) \]
\[ T_{s\text{TotalTime}} = T_{s\text{Block}} + T_{s\text{Edge}} \]

where \( X \) represents the image size (=1000, if the image is 1000 x 1000) and \( C \) represents the number of blocks along an edge (=2, if there are 2 x 2 blocks in total). And the \( K_1 \) and \( K_2 \) are constants that represent the calculation time needed for calculating a single block pixel and calculating a single splitting-edge pixel respectively.

As illustrated in Figure 16, with increasing block numbers, it takes longer to calculate the splitting edges but less time to calculate the blocks. The optimized number of blocks for the least total
calculation time would be the point where the sum (indicated as the green curve in Figure 16) of the block curve and splitting edge curve is minimal. Please note that although the $T_{\text{edge}}$ is a linear function, the X axis scaling in Figure 16 is not linear, which makes the corresponding Splitting edge calculation time curve appear non-linear.

![Figure 16. Block calculation time vs. edge calculation time](image)

Although computing environments may vary, it only makes senses to split large-size image data with a limited numbers of splitting edges to achieve optimal performance. An evaluation chart is presented in Figure 17: the X axis represents the numbers of blocks that an image is divided into; the Y axis represents the total time that is needed for distance calculation. Three different sized images (2500 by 2500, 5000 by 5000 and 7500 by 7500) are evaluated. With increasing image size, the trade-off lines are becoming more concave, which indicates that more splitting edges (more blocks) can improve the calculation speed before it slows down the process.
Figure 17. Number of computing nodes (X axis) vs. total calculation time (Y axis). This evaluation is based on the performance data gathered in the following environment: Intel i7 920 CPU, 6G 1333MHZ Memory, WD 7200RPM Harddrive, Windows 7 64bit, Sun JVM 1.6.0_16-b01.

6.2. How to Split the Raster Data
Although we split the original data into more than one sub-image per row in the example above (Figure 15b), in reality, a 1 column rectangle sub-image (Figure 18a), whose sub-image width is equal to the original raster data width, may yield better computational performance. For example, in Figure 18a and Figure 18b, we cut a 500 x 500 raster image into 4 pieces in two different ways, 1 column split (Figure 18a) and 2 column split(Figure 18b). The actual performance of these two splitting ways may vary depending on the distributed computing system's setup. The advantage of having a 1 column split resides in file access speed. Unlike reading multi-column splitting sub-images, accessing 1 column splitting sub-images does not require lots of random access in the
original raster data file. It is a line by line sequential operation. When the file handle has finished reading one line, it starts to read the next line at its current position in the file. On the contrary, multi-column splitting sub-images requires the file handle to go to certain locations (random access) when sub-images are constructed because one line of the original raster data file would have been split into many parts. However, a multi-column split has its unique advantages – it requires fewer splitting edges for any specified number of sub-images. For example, in Figure 18a, a 1 column split needs 3 splitting edges to generate 4 sub-images, whereas a multi-column split only needs 2 splitting edges to generate 4 sub-images as shown in Figure 18b. The number of splitting edges increases exponentially with the number of sub-image numbers in a 1 column split. In a 100 sub-image case, a 1 column split would require 99 splitting edges while a multi-column split might only require 18 splitting edges (See Figure 15b). As we have discussed above, calculating nearest distance for the edge pixels is expensive. So, a multi-column split may yield much better performance if there are a lot of sub-images going to be created. Essentially, it is the tradeoff between hard drive access speed versus CPU computational speed. However, due to the fact that reading multi-column images can be optimized and the computational complexity grows exponentially for a 1 column split, in reality, the multi-column split will show much better performance with large images.
Figure 18a. 1-column splitting.
Generating 4 parcels needs 3 splitting edges.

Figure 18b. 2-column splitting.
Generating 4 parcels needs 2 splitting edges.

6.3. Feasibility of extension to Cost Distance

Although this algorithm works well for Euclidian distance calculation, the same logic might also be applied to cost distance calculation [27] with one major problem, which makes it futile to apply it.

To understand why the PEDA can only minimally assist cost distance calculation, we need to understand how cost distance is calculated [31]. Both the growth rings and pushbroom algorithms can be used to calculate the cost distance. However, only the growth rings algorithm is guaranteed to produce an exact solution [29] [30].

"a matrix is first constructed containing the designated feature cells marked with a distance of 0, and with all other cells being tagged as unknown. In addition, a second matrix is formed in which the frictional effect of each cell is stored. All frictions are indicated with a value relative to 1. Thus, for example, a friction of 2 would indicate that it costs twice as much as usual to pass through that cell. The procedure then involves a series of passes through the matrix in which unknown cells which are adjacent to a cell of known distance are given a distance equal to the known cell plus one times the frictional effect in the cardinal directions and a distance equal to the known cell plus 1.41 (square root of 2) times the frictional effect in the diagonal direction." [27]
Calculating the cost distance for the edge pixels would require pixel-to-pixel propagation. In order to calculate the cost distance for edge pixels by propagation, it has to essentially calculate all pixels' cost distance along the cost distance path, which makes splitting the original raster image impossible. Therefore, the PEDA will not work for cost distance calculation.

7. Conclusion
The PEDA algorithm is one which subdivides a raster image into sub-images and wraps each sub-image in a one pixel deep layer of the distance information needed so that separate processing nodes can compute the remaining distances within each sub-image. As has been demonstrated, the PEDA algorithm can successfully parallelize the raster Euclidian distance problem. Implementing this algorithm on top of a distributed system can substantially improve processing speed and is amenable to the handling of massive amounts of data.
Chapter 3: A Distributed Storage Schema for Cloud Computing based Raster GIS Systems

Abstract
Cloud computing is a novel computing paradigm that offers powerful data and computing services on various scales. GIS systems that are built on top of cloud computing infrastructures are capable of offering superior services compared to traditional GIS systems. However, without a cloud computing compatible data storage system, such GIS systems cannot practically exploit advantages from the underlying cloud computing infrastructure. Although a relational database-based GIS data storage system could provide acceptable performance when the hosted data volumes are fixed at small to medium levels, it lacks a practical solution to scale with data volume. This makes it less desirable for cloud computing based GIS systems.

Given this limitation of relational database management systems, the NoSQL distributed database management system (NDDBMS) is being developed to provide highly scalable data services. It is capable of hosting petabytes of data across thousands of commodity computers. NDDBMS can work seamlessly with many types of cloud computing infrastructure. By adopting NDDBMS, this paper proposes a NDDBMS based distributed data storage schema that is specialized in hosting raster GIS data. Systems adopting this schema can provide highly scalable services for huge volumes of raster GIS data over commodity computers, which makes them great components for cloud computing based raster GIS systems.

Keywords:
cloud computing, distributed database, spatial database, GIS, cyberinfrastructure
1. Introduction
With the rapid development of cloud computing, NoSQL distributed database management systems (NDDBMS) are becoming increasingly popular due to their advantages over relational database management systems (RDBMS) in the cloud era. One of the most significant advantages is the inexpensive scalability of NoSQL distributed databases, which is key to handling huge volumes of data in today’s cloud computing world.

Bigtable [20], HBase, Dynamo [37], Cassandra [36], and Hypertable are all examples of this type of NoSQL distributed database management system. Most of these systems are under rapid and intensive development. However, other than Google’s Bigtable publication [20] which briefly mentions that Bigtable hosts image data for Google Earth and Google Map, there is no literature or open projects about hosting spatial data sets in NoSQL distributed database management systems.

This paper proposes a storage schema for hosting raster spatial data in NoSQL distributed database management systems. This schema can be used as a blueprint to build the storage system for a cloud based raster GIS system. This paper mainly focuses on the storage schema for hosting raster spatial data sets. Thus only the necessary background on NDDBMS that relates to this topic is covered. Among all the available NDDBMS, HBase will be the focus as the underlying database model. The rationale behind this choice will be evident in the next section. Despite some differences among currently available NDDBMS, the storage schema proposed in this article can be transplanted to other NDDMBS since they share some common features such as scalability and being NoSQL based.
2. A Brief Introduction to HBase

Among all NDDBMS mentioned above, HBase is one of the most popular scalable NDDBMS that fits a cloud computing infrastructure seamlessly. In this section, HBase is briefly introduced along with the compelling reasons for choosing it to be the underlying database model for a Cloud-based GIS storage schema.

2.1. History

In the Symposium on Operating Systems Design and Implementation (OSDI) 2006, Google engineers proposed a database architecture that is quite different from traditional RDBMS such as Oracle’s RDBMS and Microsoft SQL Server. It is named Bigtable, which is defined as: “a distributed storage system for managing structured data that is designed to scale to a very large size: petabytes of data across thousands of commodity servers.” [20] Ever since its creation, Google has successfully used it for many purposes [20]. However, Bigtable is copyrighted by Google and thus is not publicly accessible. Fortunately, the open source Apache Software Foundation community has developed a similar data storage system named HBase, which imitates Bigtable in many ways [34] [11]. Although HBase was first developed as an open source project by the company Powerset, which was later acquired by Microsoft, it has been a top-level project as part of Apache’s Hadoop project [11].

2.2. Why HBase?

HBase offers many distinctive features that make it a good underlying database model for the proposed storage schema. Among the many features of HBase, the most relevant ones to this storage schema are as follows:
1. HBase offers good scalability. HBase is designed to scale out, which means an HBase storage system can expand its capacities by adding more computing nodes. This is how it triumphs over most RDBMS, which can only scale up by adding more resources into one giant server.

2. HBase is built to use commodity hardware. Unlike RDBMS, HBase storage systems use commodity hardware instead of expensive server hardware to achieve their scalabilities. HBase is also built to be tolerant of hardware failure.

3. HBase can host huge volumes of data. Based on the above two features, HBase can host huge amounts of data with relatively low cost. When an HBase storage system runs out of resources, the HBase administrator can just add more commodity computing nodes to expand its capacity without evident performance decrease. Furthermore, the hardware cost to the hosted data size is linear for HBase.

4. HBase offers high availabilities. HBase can provide highly available services due to the fact that it replicates and distributes data over the computing nodes. Even after a computing node has failed, HBase can still provide uninterrupted data services for data that were originally hosted by that failed computing node.

5. HBase offers consistency through its data operations. This is a key feature that is essential for GIS systems. Unlike Cassandra’s eventual consistency data model (eventual consistency means that there are times that different copies of the same record may have different values in the database system, but eventually the values of these copies of the same record will be the same after certain synchronization processes), HBase locks the data it is updating to
ensure that data operations are atomic. No GIS user would like to see two identical queries
sent at the same time over the same target area produce two different results.

To summarize the above features, HBase provides a highly available, highly scalable, consistent,
and low cost solution to the hosting of large volumes of data. These features can greatly enhance
the data operation capabilities of cloud based raster GIS systems.

2.3. HBase Architecture
To better understand this proposed storage schema for cloud based raster GIS systems, it will be
helpful to briefly review how HBase works and what its limits are. The discussion below is based
on the documents provided in the HBase project web site, which is hosted by Apache foundation
[34].

Similar to the data model of Bigtable, HBase is a distributed, persistent, and sparse
multidimensional map, which is sorted lexicographically [34]. Table 2 is the conceptual view of an
example table in HBase. In HBase, applications store data rows in tables that are distributed over
all the distributed computing nodes. Each row has a row key that is sorted lexicographically and
serves as the index. A row may have an arbitrary number of column families. For example, in
Table 1, “water height inside dam”, “submerged village area” and “hurricane” are column families.
Each column family may have an arbitrary number of qualifiers and corresponding values. For
example, in Table 1, “village a” and “village b” are the qualifiers; “20000 m\(^2\)” and “5000 m\(^2\)” are
their values respectively.
<table>
<thead>
<tr>
<th>Row Key</th>
<th>Time Stamp</th>
<th>Column &quot;water height inside dam&quot;</th>
<th>Column &quot;submerged village area&quot;</th>
<th>Column &quot;hurricane&quot;</th>
</tr>
</thead>
<tbody>
<tr>
<td>&quot;Key state park&quot;</td>
<td>t9</td>
<td>&quot;50 m&quot;</td>
<td>&quot;village a&quot; &quot;20000 m²&quot;</td>
<td>&quot;Ike&quot;</td>
</tr>
<tr>
<td></td>
<td>t8</td>
<td>&quot;village b&quot; &quot;5000 m²&quot;</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>t6</td>
<td>&quot;49 m&quot;</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>t5</td>
<td>&quot;42 m&quot;</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>t3</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Table 2: HBase conceptual table model

As indicated above, there are some null-value database cells in the Table 2. However, physically, HBase stores this example table as indicated in Table 3. Each column family is stored in a separate table. Each table will be stored in the file system as an individual file. If there is a query for “water height inside dam” on time t9, the database will not return any value. This is why HBase is called a sparse storage system since it does not store any null-value, which RDBMS does. Because the table is stored sparsely, the rows in the same table may have dramatically different numbers of columns. And there is a field called "Time Stamp" in the table that represents the time when a database cell in the table was created.
Table 3. The physical storage of column families in HBase

In HBase, tables are split into pieces based on row ranges, which are then stored distributively over the storage system. These pieces of tables are called regions. Each region contains a certain number of rows. Regions are maintained by distributed computing nodes called region servers. Once a region grows larger than a pre-configured size, it will be split by HBase. HBase will then create two new regions. Each of them will take half of the data from the original region.

Figure 19. Three-tier HBase architecture.
The addresses of data regions (regions that store the actual user data) are maintained by special regions called META regions. The addresses of META regions are hosted by a special region called the ROOT region. All regions (including the ROOT region and META regions) are assigned to different computing nodes (region servers) by a master computing node called HBaseMaster. In general, its minimum setup will be a three-tier distributed cluster (see Figure 19).

A typical row query will work this way: first, the client application communicates with HBaseMaster to find the location of the ROOT region. This is the only communication between the client and the master; second, the client scans the ROOT region server to find the META region server that contains the location of the data region server that contains the desired row; third, it then contacts the META region server and scans the META regions to find the location of the data region containing the desired row; fourth, the client contacts the data region server that serves the data region containing the desired row to locate the desired row; finally, after locating the desired row, the client issues the read request. All information from the above steps is cached in the client so future requests need not go through this process again unless the regions are being re-located [34].

In theory, a three-tier HBase can host up to $2^{34}$ regions with each region hosting 256 (configurable) megabytes of data which equals up to 4096 petabytes of data. The following explains how the capacity is calculated: suppose that the storage system uses 2028 bytes to hold the address of each region, and then the first tier-root region can host $256 \times 1024 \times 1024 / 2048 = 131072$ META regions. Each META region can host up to 131072 data regions. In a 3-tier HBase storage system, it can host $131072 \times 131072 \times 256M = 4096$ petabytes of data.
2.4. Limits
There are certain soft limitations that are set for HBase to achieve better performance. For the particular interests of raster GIS applications, the following limitations are highlighted when designing the storage schema: 1. the database cell size should be kept small, which in general should not be larger than 20 megabytes. HBase reads database cells into memory when it accesses them. Thus, the database cell size is preferably not too large, especially when considering that many database cells could be accessed at the same time on a region server [40]; 2. the number of column families cannot be infinite. Because each column family will consume a part of the memory, the total number for column families in a table cannot be infinite. The preferred database cell size and column family numbers vary based on the hardware of the region server.

3. Data storage schema for cloud based raster GIS system
After understanding the underlying database storage system (HBase), a data storage schema for cloud based raster GIS system is proposed in the following sections. This storage schema will work seamlessly with a cloud computing infrastructure.

This storage schema is designed to provide the following features: 1. it is able to practically host huge volumes of data sets; 2. it is capable of hosting both 2-dimensional data sets (e.g. satellite images) and 3-dimensional data sets (e.g. image time series data sets); 3. it offers data operations over both 2-dimensional geographic space and the 3\textsuperscript{rd} dimension, quickly and efficiently even for very large data sets (large in the sense of 2-dimensional area size or in the sense of the third dimension length); 4. for the majority of data intensive operations, the loads over the distributed
storage system are distributed nearly evenly across all region servers; 5. the cost of building the storage system will be near-linear to its storage capacities; 6. expanding the storage systems will not interrupt present services; 7. it provides near-stable performance no matter the volume of hosted data.

The basic idea of this storage schema is to split the raster data sets into multiple blocks over their 2-dimensional geographic spaces and store these blocks into HBase. In this article, a block is defined as an area over a 2-dimensional geographic space that contains a group of pixels, whose geographic locations are within the block area. All pixels in blocks will be stored either in S-mode (band interleaved by pixel) or T-mode (band sequential) [41], as will be explained in greater detail below. Each block will be stored in an HBase row. The block is the basic storage unit of this storage schema. The pixel and block storage models, along with the other details of this storage schema, will be discussed in greater detail in the following sub-sections. First, two types of commonly used GIS data sets will be discriminated. Corresponding storage schemas are going to be proposed for them respectively. Then two block storage data models inside HBase will be introduced. In the final part of this schema description, the strategies for how to split the data sets and generate blocks will be discussed.

3.1. Data Types and Pixel Storage Model
A raster GIS system may need to host various raster data sets for serving different applications. It is of interest for this storage schema to divide these different raster data sets into two categories: high third dimensional data sets (H3D) and low third dimensional data sets (L3D). A H3D data set would have many layers over the same geographic regions and these layers share the same geospatial attributes such as spatial resolutions, coordinate systems, projection systems, etc. A
A typical example data set in this category is an image time series of climate data. Conversely, a L3D data set has only a few or only one layer over certain geographic regions. A set of three bands of Landsat satellite imagery belongs to this category.

The most important reason to discriminate between these two types of raster data sets is that most applications have different access patterns and different needs for them. For high third dimensional data sets, users tend to pay more attentions to the data continuities over the third dimension than they do to the low third dimensional data sets; on the other hand, for low third dimensional data sets, users tend to pay more attention to the data continuities over the 2-dimensional geographic spaces, rather than the data continuities over the third dimension. For example, it is not rare to see applications that analyze time series data query on only a few pixels, but over the entire time span of the data sets; it is also usual for applications to do buffer analysis over large geographic areas on only one layer of raster data sets.

Understanding the differences between these two types of data sets is essential to design an efficient data storage schema for raster GIS. To offer better performance, this storage schema provides two different pixel storage data models to store block pixels based on their data set types, either in T-mode (band sequential) or in S-mode (band interleaved by pixel). The terms T-mode and S-mode have been adopted here based on Cattell’s (1966) modes of Factor Analysis [41].
**T-mode**: Pixels are stored based on their locations in the original 2-dimensional geographic spaces. They are stored in sequences that are ordered from left to right, top to bottom, layer by layer. For pixels at the same geographic location but from different layers, they will not be stored together. Instead, pixels that are neighbors in the geographic spaces are stored together in sequences (See T-mode in Figure 20). In remote sensing, this form of storage is known as band sequential.

T-mode storage will mainly be used for operations that focus on 2-dimensional geographic areas of one data layer or few data layers of the data sets (e.g. the green band in a multi-band TM images). When a user performs such operations, all pixels that fall in the requested areas can be accessed speedily since they are stored together in sequences. Therefore, this storage schema stores low third dimensional data sets in T-mode.

**S-mode**: Pixels at the same geographic location from all data layers through the third dimension are stored together in sequences from the first layer to the last layer, and then for each layer from left to right and top to bottom. For example, the pixel at the first row first column position in the first layer will be stored, and then the pixel at the first row first column position in the second layer will be stored and so on; after all the pixels from different layers at the first row first column
position have been stored, the pixel at the first row second column position in the first layer is

goint to be stored and this process continues until all data are stored (see S-mode in Figure 20). In
remote sensing, this form of storage is known as band-interleaved by pixel.

S-mode storage will be primarily used for operations that focus on arbitrary spans over the third
dimension of the data sets (e.g. layers within certain time periods of a time series data set) for a
certain geographic area. When a user sends such requests, the storage system will first locate the
pixels inside the geographic areas of interest. Once a pixel is located, pulling out its values over
the third dimension would be fast since all its values are stored together in sequence. This is the
reason why this storage schema uses S-mode to store high third dimensional data sets.

3.2. Block Storage Model in HBase
For both high third dimensional data and low third dimensional data, this storage schema stores
all pixels from a block into a single row in HBase to achieve the best performance. This storage
schema assumes that most GIS applications will more often need to make operation requests
over 2-dimensional spaces than over one pixel or one line of pixels. This is one of the reasons why
the basic storage unit in this storage schema is a 2-dimensional block instead of an individual pixel
or a line of pixels. Also, to store pixels together in sequence would greatly improve the search and
access speed in HBase compared to storing the individual pixels as single units. However, for H3D
and L3D data sets, this storage schema adopts different block data storage models.

For high third dimensional data set: all pixels inside a block will be stored in S-mode in a single
row in HBase. All pixels from different layers over the third dimension that share the same
geographic position will have their values stored together in sequence in an individual database cell in S-mode. All such database cells belong to a single column family (please see Figure 21).

The reason for storing all database cells into a single column family is due to performance considerations. To speed up data access to pixels in a data block, storing pixels together in sequence would greatly decrease the search time and hard drive access time. By storing the high third dimensional data this way, an application can easily identify a block of pixels by using the row key, and then fetch the values for certain pixels inside that block quickly by reading the database cells in the same row from the same column family file. Otherwise, as mentioned in the previous section, each column family is physically stored in an individual file in HBase and consumes some memory. If this storage schema stores each pixel into an individual column family, there will be two potential problems: 1. accessing geographically neighbored pixels will not be as fast as storing them into a single family since each column family is physically stored individually; 2. more memory would have to be consumed.

The performance of this approach can be less than ideal if an application just needs to access all pixels in a large geographic area in one data layer (e.g. all pixels in one layer of a time series data set). For all pixels inside that block, each database cell storing the S-mode pixel values has to be
traversed to get only one value in the S-mode stored sequences. In this case, using T-mode would be strongly suggested.

**For low third dimensional data:** all pixels inside a block will be stored in T-mode in a single row in HBase. Each column family stores a layer of pixels that are within the geographic extents of the block. For example, to store all pixels in a certain area over a 4-band TM image, each column family will store one block of each band of that image. There is only one database cell in each column family (see Figure 22).

![Figure 22. Data storage model for low third dimensional data in T-mode](image)

The logic behind this design is to provide good locality to data operations over 2-dimensional geographic spaces. All pixels in the same block are stored together in T-mode. When an application sends operation requests that are related to a block, this storage schema ensures that the storage system can easily locate the entire block by searching the row key. Once the row is located, pulling out all pixels in one data layer of that block will be fast since they are all stored together in the same column family. The reason to put multiple layers in different column families in the same row is because this can decrease the total number of row keys and facilitate quick layer to layer operations such as overlay. However, this storage model might cause performance issues if an application just needs operations on a few pixels but from a lot of layers. Many
column families (and database cells inside them) will have to be traversed to locate values from layers for that few pixels. In this case, using S-mode would be strongly suggested.

3.3. Block Generation
To store raster GIS data sets into the storage system, the first step is to split the raster data sets into smaller blocks, and then store these blocks into HBase. For both high third dimensional data sets and low third dimensional data sets, the split strategies are similar. The splits are generated based on a modified quadtree (Q-tree) [35] [39] structure over the 2-dimensional geographic spaces of the raster data sets. All generated blocks are indexed based on the modified quadtree structure and a space-filling method that is analogous to but different from Morton curve [39] [44]. All parts of the raster data sets are recursively subdivided into four quadrants in the 2-dimensional geographic spaces, and all layers along the third dimension share the same splits. This is different from the traditional splitting logic of quadtrees, and is designed to ensure that spatially proximate blocks are stored in near-by rows of the database. Please see Figure 23: the demonstrated splits show how to use squares to divide the top left quadrant of a 2-dimensional space. In practice, the splits can generate blocks in squares or rectangles. This storage schema uses either squares or rectangles, depending on the 2-dimensional geographic size of the raster data sets. The rationality for using squares or rectangles is based on the assumption that most applications would have equal chances to have pixel operations along both the x and y directions in a 2-dimensional geographic space.
Figure 23. The procedure of splitting a raster data set into blocks and indexing them based on a quadtree structure. The numbers on the bottom-right corner of each block is the index for that block. Only the leaf level blocks (in this example, 0120, 0121, 0123, 0122) will be stored.

The recursive divisions will stop once the quadrants' sizes (number of pixels that a quadrant contains) reach the configured number. The smallest quadrants (leaf level quadrants) are the blocks that are going to be stored in the storage system.

After the blocks are generated, a unique key will be assigned to each of these blocks. These keys are produced by an indexing procedure that indexes all quadrants, which include both leaf level and non-leaf level quadrants. The indexing procedure starts from the top level of the quadtree: first, the top left quadrant is indexed with number 0; then, the top right quadrant is indexed with number 1; the bottom right quadrant is indexed with number 2; finally, the bottom left quadrant is indexed with number 3. The sub-level quadrants will follow the same indexing rule, except that their indices will keep their upper level quadrants' indices as the first part of their own. For
example, in Figure 23, the smallest quadrants (leaf level of the quadtree) have indices such as 0120, 0121, 0122, and 0123. The indices for the leaf level quadrants (blocks) will be their row keys in HBase.

By assigning block keys this way, blocks that are neighbors in 2-dimensional geographic spaces are stored closely to each other in HBase. Because each block is stored in a single row and HBase sorts all rows in lexicographical order, blocks starting with the same key part will be stored in adjacent rows in HBase. For example, in Figure 24, keys for blocks (Figure 24a) in the top right quadrant all start from the same key part “001”. In HBase they will be sorted lexicographically and stored adjacently as shown in the Figure 24b. The major advantage of storing nearby blocks in HBase closely is that it decreases hard drive random accesses since all the sequential rows in HBase are physically stored sequentially on the hard drive as well.

By assigning block keys this way, blocks that are neighbors in 2-dimensional geographic spaces are stored closely to each other in HBase. Because each block is stored in a single row and HBase sorts all rows in lexicographical order, blocks starting with the same key part will be stored in adjacent rows in HBase. For example, in Figure 24, keys for blocks (Figure 24a) in the top right quadrant all start from the same key part “001”. In HBase they will be sorted lexicographically and stored adjacently as shown in the Figure 24b. The major advantage of storing nearby blocks in HBase closely is that it decreases hard drive random accesses since all the sequential rows in HBase are physically stored sequentially on the hard drive as well.
3.4. Data Operations

Based on the character of this storage schema design and related HBase features, the implications of this storage schema for database operations can be discussed.

**Create:** When raster data sets are initially created (or imported from other sources and then created) in HBase, these data sets will be split into blocks based on a quadtree structure and then corresponding indices will be applied to each split block. Then each block will be stored into HBase either in T-mode or in S-mode, based on the nature of the data sets. The indices assigned to blocks will be used as the row keys in HBase.

**Query:** When an application needs to make a query over a certain area of a raster data set, the bounding box coordinates of this area will be interpreted into the block index (or block indices, if more than one block falls into the query area) by the client application. Then the block index (or
indices) will be sent to the storage system and be used to locate the rows where the blocks are stored.

**Update:** One important HBase feature is its consistency, which is implemented by locking the updating row. When an application needs to update some data over a block, the updated information will be written into the corresponding row while the row is locked. This guarantees that no other applications will get partly-updated rows, and thus all data in the storage system are consistent.

**Delete:** A row (which equals a block in the raster data sets) in the storage system can be deleted. However, this operation is generally not suggested unless the row was created by error. Storage space is inexpensive in today’s computing world. Delete operations can often result in permanent data loss. Updating data with timestamps would be a better choice in most cases.

4. Discussion
In this section, some issues that are not covered in the above sections will be discussed. These issues are most related to the performance optimizations of the storage system.

4.1. Load Balancing
In a cloud computing infrastructure, there are two ways to balance loads: vertical load balancing and horizontal load balancing. Vertical load balancing means that the loads are distributed and thus balanced by replicating the data that are under heavy load. For example, if certain rows of a storage system are being queried by millions of users, identical copies of those rows can be created and hosted in other computing nodes to provide service to users. Cassandra uses a vertical load balancing strategy [36]. On the contrary, horizontal load balancing does not create
vertical load balancing strategy [36]. On the contrary, horizontal load balancing does not create replications of heavy load data. Instead, it moves data among computing nodes to make sure the loads are evenly distributed among the distributed computing infrastructure. For example, if there are certain rows in the storage system that are being queried by millions of users, each part of each row will be moved to separate idle computing nodes to provide the best possible performance. By doing this, the load can be distributed. Currently, HBase (0.89) only supports horizontal load balancing by using region counts. All regions are assigned to region servers by the region master in a semi-random manner to make sure all region servers have almost the same numbers of regions [42].

For a distributed computing based raster GIS system, a different strategy can be developed to provide extra help for horizontal load balancing. In this storage schema, the quadtree structure is adopted to produce four quadrants (blocks) at all leaf level branches of the quadtree. If these four quadrants (blocks) are assigned to four different region servers, the load for the region servers will be reduced to up to a quarter. In Figure 25, for all possible 2 x 2 operation windows (marked in red circles), the strategy suggested by this paper (Figure 25b) provides an evenly distributed load among its four region servers, which in most situations produces better load balancing compared to the normal sequential storage strategy (Figure 25).
Figure 25. Load balancing diagram. Red circles represent possible access patterns for a 2 x 2 operation window. The four different colored squares represent blocks assigned to four different region servers.

This strategy can be applied in larger scales. Instead of assigning the leaf quadrants to different region servers, quadrants in higher hierarchies can be assigned to different region servers or region server groups.

4.2. Null-value handling
Since HBase is a column based database, as the example showed in Table 2, there is no need to store null-values in HBase. Instead, no database cell will be created for that null-value. Combining this feature of HBase with the quadtree structure, a lot of resources can be saved, as long as all
data layers in the data sets have the same spatial appearance. For example: in Figure 26, a raster image is first split into a 4-level quadtree; for quadrant '01', there are no valued pixels inside. Its existence then can thus be neglected, and nothing needs to be created for it. However, for quadrant (block) '0230', there is only 1 pixel inside it, and all null-values within this quadrant will have to be stored. Without using a quadtree and HBase, all null-values of this raster data will have to be stored.

Figure 26. Null-value handling in HBase with quadtree based split
5. Summary and Conclusions
The storage schema proposed here for cloud-based raster GIS is built upon the fundamental logic of the HBase NoSQL database structure. By adopting NDDBMS as the underlying database storage model rather than RDBMS, this storage schema is able to provide highly scalable and available data services to host extremely large volumes of GIS data on inexpensive commodity computers. Furthermore, the unique design of this storage schema guarantees speedy and efficient data services. The storage schema first divides raster data sets into multiple blocks based on a quadtree structure. Indices are then generated and assigned to these blocks to ensure rapid block search. Each block is stored as a single row in HBase. According to the nature of a data set, the pixels inside a row (block) can be stored in either in S-mode (band interleaved by pixel) or T-mode (band sequential) to offer the best performance. Corresponding data storage models are provided to accommodate these two different pixel storage modes. As a result, this storage schema is able to effectively host any raster data sets no matter of their sizes, and provide speedy and efficient data operations. GIS data sets stored under this storage schema will also work seamlessly with parallel algorithms such as the parallelized distance calculation algorithm [43] that sub-divides data into multiple sub-images.

Although it would be helpful to benchmark the performance [38] and effectiveness of a data storage system that adopted this storage schema, building up such a storage system and benchmarking it is beyond the scope of this paper. However, the success of similar systems such as Bigtable, which have been serving millions of requests daily for Google Maps, implies that the use of a similar structure for GIS would experience similar success for cloud-base GIS systems.
Conclusion

About 50 years ago, the first GIS system, the Canada Geographic Information System (CGIS), was developed as a mainframe based system. All resources, which included hardware, software, and data, were centered in one large expensive mainframe computer. In order to execute spatial analyses on a mainframe based GIS system, users had to get in line and submit a punch card command sequence to the mainframe administrator and then wait hours to perhaps days to get the analysis results back. Despite the comparatively high computational power of mainframe computers at that time, how to efficiently share those limited resources among many users became a problem. That was the era of centralized computing based GIS systems.

With the development of minicomputers in 1970’s, minicomputer based GIS systems such as Arc/INFO and ERDAS were developed, which have significantly cut the cost of GIS systems. However, these systems were still so expensive that only wealthy organizations could afford them. Along with the prevalence of microcomputers and personal computers (PCs), started in middle 1980’s, many microcomputer (PC) based GIS systems emerged. IDRISI and PC based Arc/INFO are both examples of PC based GIS systems. Only until then, individuals could finally afford GIS systems. In PC based GIS systems, resources are decentralized. Each user owns a set of hardware, software, and data. Once the GIS system was set up and data were ready, a user could run as many spatial analyses as he or she wants. In the recent 30 years or so, the resource sharing problem from the mainframe based GIS systems seemed having been eliminated by the development of inexpensive PC based GIS systems.
Thanks to the advancement of science and technology, large amounts of data that were not available before are being produced today, and data update frequencies are getting significantly higher. For instance, today's navigation maps offer much more detail than they were 10 years ago and they now get updated on a daily basis. Since the resources are decentralized in PC based GIS systems, most GIS data have to be either delivered on storage medium (such as CD or DVD) or delivered via FTP servers from data vendors. As data size is increasing at an exponential rate and data are getting more frequent updates, how to acquire the most up-to-date data rapidly is becoming increasingly challenging. This has been the era of decentralized computing based GIS systems.

The recent emergence of cloud computing, offers an excellent solution to the problems in both centralized computing based GIS systems and decentralized computing based GIS systems. Cloud computing is a hybrid computing paradigm, which combines both centralized and decentralized computing. Most data and computations are hosted in the massive cloud computing infrastructures. Users keep limited cached data and simple computations on the client computers. High-speed networks are used to connect the cloud computing infrastructures and the users’ client computers. In cloud computing, resources can be shared easily and data can be updated frequently.

The advances of cloud computing technology offer many potential opportunities to enhance the functionalities and performances offered by traditional GIS systems. Fully exploiting the advanced features of cloud computing for GIS systems, however, is quite challenging. Understanding cloud computing would be a fundament requirement to develop a capable cloud computing based GIS
system. However, as a newly emerged computing paradigm, details of cloud computing are still unknown to many GIS community members. Additionally, many truly spatial GIS algorithms are not compatible with the distribution nature of cloud computing. Re-designing these spatial GIS algorithms to furnish them with the benefits provided by cloud computing infrastructures would be essential to the cloud computing based GIS systems. Storing and serving large amounts of GIS data in cloud computing infrastructure is another challenge that needs to be tackled.

To fully understand the implications of cloud computing, discover the potential challenges and possible solutions in developing cloud-based GIS systems, three stand-alone articles have been developed. Each of these three articles answers a key part of the general question: how to develop a cloud computing based GIS system? Detailed conclusions from each article are listed in the following section.

First Article: Cloud Computing and Its Applications in GIS

An extensive review of cloud computing was first conducted in this article. Differences such as administrative domains were examined between cloud computing and similar distributed computing technologies, e.g. GRID computing and HPC. Advantages of cloud computing were discovered such as low hardware and maintenance costs, low entry barriers for public users, friendly to large volumes of data, high reliability, high availability, strong scalability, superior economy of scale, and promising potentials for high performance computing. Based on these merits of cloud computing, the concept of cloud computing based GIS systems was raised. Its definition was presented as following:
A GIS system that builds on top of a Cloud Computing infrastructure, using the cloud infrastructure to dynamically scale its computing and/or storage capabilities, providing parallelized services that are able to serve various user bases including authorized users and/or massive numbers of public users; these services should include basic GIS related functionalities such as reclass, overlay and so forth, as well as truly spatial GIS analysis functionalities such as cost distance, watershed runoff, etc.

Three architecture designs of cloud computing based GIS systems were then proposed, which are public cloud-based GIS systems, private cloud-based GIS systems and hybrid cloud-based GIS systems. Public cloud-based GIS systems provide the lowest entry barriers to users with maximum economy of scale, but their advantages are offset by data security and privacy related issues. Private cloud-based GIS systems provide the best security and privacy protections for GIS data, however, they have the highest entry barriers among these three types of cloud-based GIS systems. Hybrid cloud-GIS systems offer a compromise between public cloud-based GIS systems and private cloud-based GIS systems. A hybrid cloud-based GIS system is comprised of a private cloud-based GIS system and a public cloud-based GIS system. The private cloud provides daily services, while the public cloud provides extra back up services once the private cloud is unable to satisfy users’ demands. Although hybrid cloud-based GIS systems possess obvious advantages from both public cloud-based GIS systems and private cloud-based GIS systems, they also suffer from disadvantages of these two types of cloud-based GIS systems.

Second Article: A cloud computing compatible algorithm for the calculation of feature distance for raster GIS

This article proposed a distributed minimum Euclidean distance calculation algorithm to calculate the shortest distance between image pixels and feature pixels. This algorithm can be implemented on a cloud computing infrastructure, and then be executed distributely to provide high performance distance calculations for large raster data. The fundamental logic of this
algorithm is to subdivide a raster image into sub-images (blocks) and wraps each block in a one
pixel deep layer of the distance information needed so that separate computing nodes can
compute the remaining distance within each block.

This article started with a review of several popular raster based minimum Euclidean distance
calculation algorithms such as pushbroom and growth rings. The proposed distributed minimum
Euclidean distance algorithm was then presented in detail. Among many parameters, the number
of generated blocks is the most important factor to the algorithm performance. To better
evaluate the optimal block number, a series of equations were provided in this article to generate
the optimal block numbers for any given input raster images. Also, two different block generation
strategies, which are the 1-column split and multiple-column split, were also discussed in detail.

The logic of this proposed algorithm can also be applied to parallelize many stand-alone spatial
GIS algorithms and image processing algorithms for running on cloud computing infrastructures..
For example, a DEM interpolation algorithm can adopt this logic by creating sub-images that are
wrapped with a layer of pixels that has the width of the moving sampling window. Although there
are a few algorithms that cannot be parallelized by adopting this method, e.g. cost distance
algorithm, many spatial GIS algorithms can be successfully parallelized by implementing this
“subdivision” logic.

Third Article: A Distributed Storage Schema for Cloud Computing Based Raster GIS Systems

This article proposed a distributed storage schema for cloud computing based raster GIS systems.
The proposed storage schema adopted HBase as its underlying database storage system, which is
a NoSQL Distributed Database Management System (NDDBMS). As an innovative database
management system, NDDBMS offers many advantages such as strong scalabilities and high availabilities over Relational Database Management System (RDBMS).

Based on the advantages offered by HBase, this article presented a storage schema that is able to effectively host any number of raster data sets no matter of their sizes, and to provide speedy and efficient data services. This storage schema first divides raster data sets into multiple blocks based on a quadtree structure. Indices are then generated and assigned to these blocks by using a modified quadtree structure and a space filling method that is analogous to but different from Morton curve to ensure rapid block search. Each block is stored as a single row in HBase. According to the nature of a data set, the pixels inside a row (block) can be stored in either in S-mode (band interleaved by pixel) or T-mode (band sequential) to offer the best performance. Corresponding data storage models are provided to accommodate these two different pixel storage modes.

The data storage schema proposed in this article is a comprehensive storage schema that dedicates in distributed spatial database. Successful implementations of this storage schema will potentially change the way most raster GIS data are stored, from RDBMS or file systems to NDDBMS due to the benefits of adopting it: 1. it will considerably decrease the cost of raster GIS data storage systems; 2. it will significantly increase the capacities of the raster GIS data storage systems; 3. it will increase the speed, availability, and reliability of raster GIS data services. Although this storage schema has focused on raster GIS systems, some logics and methods (such as load balancing) proposed in this storage schema can be applied to vector GIS systems too.
For any scholar who is interested in cloud computing based GIS systems, it will be of interest to become acquainted with this novel database management system and the corresponding data storage schema.

**Dissertation Conclusions**

Through the in-depth research conducted in this three-article dissertation, the general question of how to build a cloud-based GIS system was answered with promising results. First, the advantages of building a cloud-based GIS system were discovered. Second, migrations of truly spatial GIS algorithms to a cloud computing infrastructure are proved to be feasible, with certain challenges. Third, a cloud computing infrastructure compatible GIS data storage schema was proposed, which is able to store extremely large volumes of GIS data on inexpensive hardware and provide highly scalable, available, efficient and speedy data services for cloud computing based raster GIS systems. This dissertation supports the feasibility of building a cloud-based GIS system. However, there are still some challenges such as some truly spatial algorithm migrations that need to be addressed before a full-scale functional cloud-based GIS system can be successfully implemented.
Reference


