Western Michigan University

ScholarWorks at WMU

Master's Theses

Graduate College

6-2013

Data Storage Alternatives for a Gridded Crop Disease Risk Forecasting System

Paul J. Roehsner

Follow this and additional works at: https://scholarworks.wmich.edu/masters_theses

Part of the Agriculture Commons, and the Geographic Information Sciences Commons

Recommended Citation

Roehsner, Paul J., "Data Storage Alternatives for a Gridded Crop Disease Risk Forecasting System" (2013). *Master's Theses*. 157.
https://scholarworks.wmich.edu/masters_theses/157

This Masters Thesis-Open Access is brought to you for free and open access by the Graduate College at ScholarWorks at WMU. It has been accepted for inclusion in Master's Theses by an authorized administrator of ScholarWorks at WMU. For more information, please contact wmu-scholarworks@wmich.edu.

المنارة للاستشارات

www.manaraa.com

DATA STORAGE ALTERNATIVES FOR A GRIDDED CROP DISEASE RISK FORECASTING SYSTEM

by

Paul Roehsner

A thesis submitted to the Graduate College
in partial fulfillment of the requirements
for the degree of Master of Arts
Geography
Western Michigan University
May 2013

Thesis Committee:

Kathleen Baker, Ph. D., Chair
Chansheng He, Ph. D.
Lei Meng, Ph. D.

المنارة للاستشارات

www.manaraa.com

DATA STORAGE ALTERNATIVES FOR A GRIDDED CROP DISEASE RISK FORECASTING SYSTEM

Paul Roehsner, M.A.

Western Michigan University, 2013

Three separate storage technologies able to serve gridded data were selected for comparison of performance in terms of providing speed and expandability to a crop disease forecasting system. The three storage technologies chosen were PostgreSQL (a relational database management system), MongoDB (NoSQL system), and netCDF files. Speed tests were performed for each by running two different crop disease risk forecasting models requiring data of different spatiotemporal resolutions. Multiple trials were done using different storage hardware. Systems were then qualitatively compared for expandability by noting the process involved in adding successive crop disease forecasting models.

It was found that due to different respective limiting properties of each implementation of all three storage technologies the speed differences using traditional storage hardware were few. Given this, it would be possible to further fine-tune a system using netCDF files for speed gains. Qualitative notions of expandability featured by the different storage technologies then become a significant factor when making a choice between the three to use for a crop disease forecasting system. Both PostgreSQL and MongoDB storage technologies offered better expandability in terms of difficulty of adding additional models compared to the system using netCDF files.

المنارة للاستشارات

www.manaraa.com

Copyright by
Paul Roehsner
2013

المنارة للاستشارات

www.manaraa.com

ACKNOWLEDGMENTS

I would like to thank my wife and family. They have been patient and supportive throughout. This has allowed me to achieve my educational goals.

Next I would like to thank the Geography Department at Western Michigan University. I believe that knowledge, skills and fortitude are essential things for most successes. The Geography Department at Western provides a challenging and rewarding atmosphere for students where knowledge, skills and fortitude can be cultivated.

My thesis chair Dr. Kathleen Baker has provided much assistance with this thesis as well as offered an enriching research environment to work in. I am grateful for the opportunity offered and the overall experience. Quite a few students have greatly benefitted by working as research assistants in the crop disease forecasting research lab.

I am also thankful for Dr. Chansheng He and Dr. Lei Meng. They have been gracious in focusing my research perspective and have improved the presentation of ideas in this thesis.

Paul Roehsner

المنارة للاستشارات
www.manaraa.com

TABLE OF CONTENTS

ACKNOWLEDGMENTS iii

LIST OF TABLES viii

CHAPTER

I. INTRODUCTION 1

Problem Statement 2

Purpose of Research 5

Possible Effective Storage Solutions 6

Objectives 8

Objectives Relating to Methods 9

Objectives Relating to Results 9

Expected Outcomes 9

II. LITERATURE REVIEW 11

Gridded Weather Data Applicable to Crop Disease Forecasting 11

Storage Technologies in Systems Able to Serve Geographic Data 12

Relational Database Management Systems for GIS 13

RDBMS Brief Background 13

Data Models for RDBMS and GIS 13

NoSQL Databases for GIS 17

What are NoSQL Databases? 17

NoSQL Database: Key-Value Pair 19

NoSQL Database: Column-Family Data Store 20

NoSQL Database: Document Store 22

NoSQL Database: Graph Database 23

المنارة للاستشارات

www.manaraa.com

Table of Contents – Continued

CHAPTER

NoSQL Database Discussion 24

Scientific Data File Formats for GIS 25

Chosen Technologies 27

PostgreSQL: A Relational-Database Management System 27

MongoDB: A NoSQL Database 28

NetCDF: A Scientific File Format 30

Why Rasters or Shapefiles Were Not Used as Test Storage Platforms 31

III. METHODOLOGY 35

Methods for the PostgreSQL System 35

PostgreSQL: Logical Data Structure 35

PostgreSQL: Optimizations 37

PostgreSQL: Data Retrieval During Model Runs.................... 38

Methods for the MongoDB System 39

MongoDB: Logical Data Structure 39

MongoDB: Optimizations 42

MongoDB: Data Retrieval During Model Runs 42

Methods for the NetCDF System 44

NetCDF: Logical Data Structure 44

المنارة للاستشارات

www.manaraa.com

Table of Contents – Continued

CHAPTER

NetCDF: Optimizations 45

Chunking 45

Data Compression 46

NetCDF: Data Retrieval During Model Runs 46

Specifications for Designed System Comparison 50

Hardware 50

Software 51

Quantitative Assessment: Speed Tests 52

Test Models Used 52

Initial Case Example: A Wallin Potato Model Forecasting Potato Late Blight 52

Second Case Example: Model Deriving Daily Inputs and Daily Target Output For Training of a Neural Network to Predict Risk of Barley Head Blight 52

Speed Test Description 54

Quantitative Assessment: Expandability Test 55

IV. RESULTS AND DISCUSSION 56

Quantitative Results 56

Quantitative Results Discussion 59

MongoDB Version One 59

المنارة للاستشارات
www.manaraa.com

MongoDB Version Two 60

Table of Contents – Continued

CHAPTER

NetCDF System 61

PostgreSQL System 63

Qualitative Commentary on Expandability 63

Conclusion 65

REFERENCES 67

APPENDICES 71

A. Tables in a PostgreSQL Database for Gridded Crop Disease Forecasting. 71

B. Document Hierarchy in a MongoDB Datastore for Gridded Crop Disease Forecasting 73

C. One Growing Season's NetCDF File 75

المنارة للاستشارات
www.manaraa.com

LIST OF TABLES

1. Time Required to Run a Model Forecasting Daily Risk of Potato Late Blight over Four Years for a Five Month Season for an Area Covering Michigan 58

2. Time Required to Derive Daily Inputs and Daily Target Output to be Used to Train a Model Forecasting Risk of Barley Head Blight for a Five Month Season over Four Years for a Tri-State Area 58

www.manaraa.com

CHAPTER I

INTRODUCTION

Computationally intensive research that is able to address real world problems is enabled by relevant quality data, retrieval of that data, processing of that data, and storage of derived outputs. Informational needs have to be met before many of today's global, interdisciplinary problems can be effectively solved. Methodologies of storage and use of data impact the ways in which research is carried out and can also impact how much benefit is derived from the results. Applied data management in a research project changes the efficiency of analysis and supports effective problem solving which in turn affects decision support that is provided to end users.

This proposed research recognizes a methodology gap hindering the ability to use particular data in a cost effective way for research aimed at reducing impacts of crop diseases. Data grids containing weather variables used to forecast crop disease risk are available from the National Weather Service. These grids may improve forecasts providing agronomic and environmental benefits. For example, farmers could be better aided in the decision making process, and could apply pesticides at times and rates that are most beneficial to the farmer and consumer. Crop disease risk forecasts attempt to limit the environmental effects of pesticides while sustaining the economic benefit of a successful harvest.

Data grids from the National Weather Service containing forecasted weather variables are applicable to more than one crop disease risk forecasting model. Taking

المنارة للاستشارات
www.manaraa.com

this into account, along with the large number of models available for different crops and crop diseases, it would make sense to store and process these grids in such a way that grids could be reused with lowered research cost outlay with each additional crop disease risk model.

One key choice in system design is the storage technology chosen for the system. The storage technology as a subsystem should support the goals of the overall system. Therefore, is it important for storage systems to be assessed in terms of the likelihood of each to lend features leading to lowered research costs and fast development of quality models.

Sections of this document detail the problems this research addresses, the purpose of research, objectives, literature review, methods, and results. A produced thesis and presented results would partially fulfill requirements leading to a Master of Arts Degree in Geography: Geographic Techniques. This research involves a comparison of storage technologies used to store gridded data from the National Weather Service in addition to gridded model outputs used in a Geographic Information System forecasting crop disease risk.

Problem Statement

A layered problem can persist because of related data, technology, or methodology gaps. New methodologies of processing and storing the gridded forecast data available from the National Weather Service (NWS) are necessary to enable application of those data grids to crop disease risk forecasting systems. The layered

المنارة للاستشارات
www.manaraa.com

problem is as follows: 1) crop diseases significantly impact agro-economics, 2) forecasting crop disease risk at point locations has inherent weaknesses, 3) gridded forecasts are possible solutions, 4) there is difficultly in applying large volumes of temporal gridded data within current GIS systems, and 5) quick unplanned solutions to the above may fail to realize potential benefits of effectively applying gridded data. An overarching goal of my research is to develop quality processing and data storage methodologies for gridded weather forecasting data used within a GIS.

Crop diseases impact agronomics and the environment. For example, Potato late blight is estimated to cost growers $287.8 million annually with $210.7 million due to impacts on yield and $77.1 million in fungicide costs (Guenthner et al. 2001). Losses of the barley crop due to Fusarium head blight from 1998-2000 is estimated to be $136.4 million, or 25.7% of the total value of barley production (Nganje et al. 2002). In Georgia peanut leaf spot in 2009 caused $6.0 millions in crop damage and $26.8 million in chemical control costs (Williams-Woodward 2010). Losses due to crop diseases can be reduced by fungicides but fungicides adversely impact humans and the natural environment. Crop disease risk forecasting systems advise when and how much fungicide to apply, retaining value of crops while limiting yield production costs related to fungicide. A lowered amount of fungicide applied also reduces adverse impacts on the environment.

While traditionally risk estimates have been provided at point locations, crop disease risk forecasts provided at point locations are not always applicable to a given farm location. If a farm is located far from city centers or airports where detailed

المنارة للاستشارات
www.manaraa.com

forecasts are commonly provided, accuracy of those forecasts applied to the farm will be reduced. City centers commonly experience urban heat island effects and rural areas tend to be cooler and have higher relative humidities. Topographies involving factors like distance to water bodies and elevation changes will impact local applicability of weather forecasts.

A solution to issues inherent with point-based forecasts is providing gridded crop disease risk forecasts. These require either interpolation of point based inputs or outputs, or ideally, gridded forecast inputs. Interpolation of point-based inputs may be skewed by local effects similarly to point based forecasts. Points in urban areas may reduce accuracy in rural areas. Underrepresentation of data in certain areas changes overall accuracy and distorts the prediction field. The best possible solution may be to find applicable gridded climate variable forecasts with suitable resolution.

Gridded forecasts are available from the NWS in 5km spatial resolution and are distributed in Gridded Binary (GRIB) format. The GRIB format is not directly useable in the most common commercial GIS, ESRI's ArcGIS. Crop prediction models use this data in dimensionally large x, y, and time axes. For example, a model predicting risk of Fusarium head blight of barley using raster data with eight variables over five days with hourly data would require the processing and storage of 960 rasters per forecast day. Lining up the geometry of 960 rasters for multiple calculations may take longer than necessary especially if the geometry of each raster does not change in a given grid specification. A better way to store and process this multi-dimensional data is possible.

المنارة للاستشارات
www.manaraa.com

Newly introduced gridded data from the NWS are likely to continue to increase in resolution over time and include new variables calculated in different ways. Some examples of grids and spatial resolutions the NWS has made available include National Digital Forecast Database (NDFD) grids at 5km and Gridded Model Output Statistics (GMOS) at 5km and 2.5km resolutions. GMOS grids are available to download as they are issued but are not archived for later access by the public. As new formats and higher resolutions are introduced, new grids should be able to be used and tested in terms of output accuracy. If planned carefully, data storage and processing methodologies should allow for the addition of new grids without much increased research overhead. Available grids should then be applicable to various crop disease research models. Possible new data storage and processing methodologies of gridded NWS data should realize gridded crop disease forecasting with higher accuracy for varied crops with reduced research overhead for each added crop model. A quick solution may not take into account the benefits of a good solution. For this reason to effectively apply weather grids to crop disease forecasting solutions should be designed in ways striving to approach currently unidentified best practices.

Purpose of Research

The subsystems of a GIS are functional pieces of a GIS. Variability of these subsystems tends to be wide and impacts overall GIS functions. Focusing on storage technologies as one key subsystem, the purpose of this research is to identify a storage technology lending itself to a system enabling best practice methodologies to

المنارة للاستشارات
www.manaraa.com

process crop disease risk forecasting models using gridded data as inputs as well as storing derived outputs. Grids should be stored and processed in methodologies conducive to reducing research overhead when applying the same stored data to multiple models. Model forecasts should be produced in a timely manner to hasten the model prototyping stage. This research will identify effective storage technologies lending these features to a gridded crop disease forecasting system.

Possible Effective Storage Solutions

During the summer of 2011 I participated in a netCDF workshop led by Unidata, the group who created netCDF. The workshop contained an overview of the netCDF technology. Using the netCDF storage format, a single file, representing a time span of a growing season or month, can contain all data and outputs of different crop models and store all variables with different dimensional attributes. NetCDF, as an example of a scientific data file format seems to have potential within a system likely representing a solution using best practices for storage and processing gridded data.

Relational database management systems (RDBMS) have been commonly used to store geographic information in the past. Spatial and scientific data are usually retrofitted for table schemas originally designed to hold financial, commercial and organizational types of non-spatial data. Even though a retrofit is involved in holding spatial data, during the prior decades technologies within relational database management systems have emerged to increase the speed of storing and querying this

المنارة للاستشارات
www.manaraa.com

spatial data. For these reasons a relational database may also have potential within a system likely representing a solution using best practices for storage and processing of gridded data.

Increasing volumes of different types of data needing to be stored and served out to the Internet have made apparent certain weaknesses in RDBMS. One weakness is that RDBMSs tend not to offer "horizontal" expansion. In a horizontal expansion additional low-cost computers as servers can be cheaply added to an existing cluster of low-cost computers to be used as a single server with that cluster utilized to serve a single large dataset. The feature of horizontal expansion allows NoSQL databases to better scale when dealing with expanding data volumes found in Internet applications and potentially scientific applications. RDBMS tend to offer vertical expansion. An example of vertical expansion would be the act of adding additional resources like memory to a single machine. Vertical expansion is limited as the single machine may be upgraded to a maximum before another single machine with higher maximums is purchased to replace the former machine. The sheer number of data transactions involved in blogs, social networking, and online shopping carts has also made another weakness apparent. There is a distinct per transaction slowdown involved with the significant overhead of transaction quality assurance involved in RDBMS needed in financial systems. As a reaction and a solution to store the big data involved in today's Internet applications, developers have created and turned to NoSQL solutions. Most NoSQL solutions have reduced transaction overhead and are able to horizontally expand. NoSQL solutions also forego a strict

المنارة للاستشارات
www.manaraa.com

table schema used to store data. Since the table schema is abandoned and spatial-temporal data needs a retrofit to fit within the table schema, the alternative schemas introduced in NoSQL solutions should be considered. This is in addition to reduced transaction overhead; both of these features may be leveraged in a grid storage system.

In order to compare and identify best practices three types of systems running the same crop forecasting model will be created. Systems using a RDBMS, a NoSQL system, and a scientific file format data storage solutions will be created. Time trials running the same model and season combination will be used to test the three data storage solutions for speed. Comparative expandability of systems using the different storage methodologies will be then qualitatively assessed by reusing the same national data stored in those formats and applying that data to a second system running a different crop disease model.

Objectives

This comparison of storage technologies for gridded crop disease forecast requires meeting the following objectives. The first pair of objectives is methods related and involves design and implementation of system test cases to be compared. The second pair of objectives involves trials and assessment of those implemented systems and is related to research results.

www.manaraa.com

Objectives Relating to Methods

1. Assess alternative solutions within Relational Database Management System, NoSQL, and Scientific File Format Types. Choose three to compare, one of each type.

2. Create three different data storage and processing systems able to apply the same stored gridded forecast and validation data to different crop disease forecasting models.

Objectives Relating to Results

1. Quantitatively test the processing speed of a crop disease forecasting model for each of the three systems.

2. Qualitatively assess aspects of expandability among the three systems by applying the same gridded data to a second crop disease forecasting model.

Expected Outcomes

Because of inherent capabilities of scientific data files, including internal descriptive attributes tagging of data and speed of retrieval of subsets of data, use of a scientific data file format will likely best meet speed requirements of desired data storage and processing methodologies. The speed of a system using a scientific data file format will likely be superior to a system using a NoSQL data store or a RDBMS as storage mediums. In terms of expandability, the NoSQL data store should offer flexibility in the data schema used leading to the desired expandability. Use of either a NoSQL datastore or a scientific data file format should streamline testing of prototype systems where different datasets are applied to different risk models.

المنارة للاستشارات
www.manaraa.com

This research should lead toward systems that greatly streamline both the addition of crop forecasting models as applied to available NWS gridded datasets as well as the addition of possible future gridded datasets. The perspective of different storage systems being able to lend strengths of the format to processing and file format storage methodologies, enabling expandability and speed of processing of those methodologies, guide this research. This perspective will be tested and results will be reported with expandability used as a qualitative metric and speed as a quantitative metric.

المنارة للاستشارات
www.manaraa.com

CHAPTER II

LITERATURE REVIEW

This literature review is divided into four sections. The first section considers the available gridded data from the National Weather Service (NWS) to be used in a crop disease forecasting system. Potential applicability of that gridded forecasting data to crop disease forecasting systems is established. In the second section general storage system types usable for GIS are next considered to provide base knowledge of those types and inform the choice of storage technology to implement among examples of each type. The second section also informs the design of data schemas I implement when the system is created. The third section covers the choices among types and why those choices were made. The fourth section considers standalone raster files and why they were not chosen as a solution to implement and compare in trials. The final section takes the ideas presented in this chapter's previous sections and details which technologies where chosen given that knowledge. Afterward, the following chapter covering methodology will detail how those chosen technologies will be used and assessed.

Gridded Weather Data Applicable to Crop Disease Forecasting

Gridded datasets offered by the NWS have increased in accuracy over time (Myrick and Horel 2006; Ruth et al. 2009). In many cases they are better than previous forecast Model Output Statistics (MOS) available for the same point

للاستشارات المنارة
www.manaraa.com

locations (Ruth et al. 2009). A system able to incorporate these and future NWS grids has the potential to increase accuracy in final risk forecasts for a variety of applications.

Compared to gridded data, due to past availability of point forecasts and observations, crop risk forecasting has tended to be made at point-based locations. Within the last ten years the NWS has made higher resolution gridded forecasting data available. A few recent studies have detailed usage of this NWS data to predict crop diseases: spread of soybean rust in Minnesota (Tao et. al 2009) and strawberry fruit rot in Florida (Pavan et. al 2011).

Storage Technologies in Systems Able to Serve Geographic Data

The following will examine three storage system technologies for storage and retrieval of of the gridded dataset described above. Relational Database Management Systems (RDBMS) are first covered, then NoSQL databases, and finally scientific file formats. Since methodologies for RDBMS use in a GIS is established I cover some methodologies to inform system design of a crop disease forecasting system using a RDBMS. Extra detail is provided for types of NoSQL databases because the category is a catch-all including different subtypes. A choice for data storage technology can be made among the NoSQL subtypes and this coverage of NoSQL subtypes informs that choice.

المنارة للاستشارات
www.manaraa.com

Relational Database Management Systems for GIS

RDBMS Brief Background

The relational model was proposed by E.F. Codd (Codd 1970). In the relational model tables contain records of uniform fields. Records usually include some fields used to uniquely identify those records and allow for referencing specific records across tables. All of the early basics underpinning relational database technology and implementation appeared in published form in the 1970s. In the 1970s the practice of indexing individual records using stored values in chosen fields within individual records to speed up access to data began. Indexes made use of relational databases feasible. Without indexes records would have to be searched in order to find a target record. The B-tree index (Bayer and McCreight 1972) is the pre-cursor to the B+-tree, the most commonly used index today. The SQL language was also proposed (Chamberlin and Boyce 1974). The ER-diagram, a data modeling tool connecting the entities' different relationships in a database was also created (Chen 1976). These basics still underpin relational database technology and implementation today.

Data Models for RDBMS and GIS

GIS software packages became widely established in the late 1980s and 1990s. Underlying data models were an important consideration in the early period because data models involve some levels of geographic abstraction for both raster and

المنارة للاستشارات
www.manaraa.com

vector data (Goodchild 1992). Beyond the question of raster vs. vector is the choice of unstructured and unmanaged stand alone data files vs. a relational database for data storage and retrieval of either raster or vector data. If the relational database is chosen, the data model, in this case the sets of tables and fields used to store different data enabling the abstraction of a particular real world system must also be chosen. For example, how are tables in the relational database model going to be used to store the variables of a location needed to solve a problem? If the variables change quickly over time the temporal element must be considered because the amount of data related to a single location quickly increases and cannot all be reasonably stored in a single record using instances of the traditional float datatype. These samplings, or in a system using forecasted data, projections, have a fixed locational dimension and a differing temporal dimension.

A look at the Arc Hydro data model in terms of how spatial and temporal data are stored in tables partially informs the structure of a data model for a crop disease forecasting system also using tables. The Arc Hydro data model is a prescribed set collection of tables and fields used to abstractly represent and enable modeling of hydrological systems. The Arc Hydro extension used by the common GIS software ArcGIS analyses and produces hydrological system data in the Arc Hydro data model. I used the groundwater version of the Arc Hydro data model as an informing source (Strassberg, et al. 2011) while designing the tables of my early versions of a crop disease forecasting system using a RDBMS.

المنارة للاستشارات
www.manaraa.com

The Arc Hydro model stores temporal data including flow measurements at the same location repeated over time as well as changes in channel banks stored over time (Maidment 2004). Date/time fields in multiple borehole records list when the borehole was taken, the depth of the borehole, and a field tied to an unchanging record of the borehole's geographical location. Multiple tables containing data and metadata about the observation are kept to hold information related to the borehole made at that time. Here a date/time field exists for each observation record along with locational attributes (Chesnaux, et. al. 2011). This is a snapshot time approach as a vector at a single x-y location, but includes depth data.

Snapshot coverages in the instance of remote sensing data are typically made in raster format (Goodchild 2005), but attributes with vector representations can be stored in tables with vector geometry linked to that data. Patterned after the Arc Hydro data model's handling of spatiotemporal data my early version of a crop disease forecasting system held the unchanging geometry of grid cells separate from the weather data associated with each individual grid cell. The early version used a Microsoft SQL Server as the background RDBMS along with ArcSDE to provide access to data from within ArcGIS. Running models was done by data matched by cell id and a server-side ArcSDE table view joined all pieces of the data to associated cell geometry appearing as a table of slices of overlapping grid cells to the ArcGIS desktop client. A definition query on the client would further subset the data by single time slices. Redraw and table access was quite fast when using ArcSDE despite the volume of data.

المنارة للاستشارات
www.manaraa.com

Techniques for representing general data in non-redundant and compact ways which limit work needed to update records when attributes within a relational database change when changing of the values of that data is pretty well established as database normalization has been around since the 1970s (Codd 1970 & 1974). Establishing data models for scientific data with a both static spatial dimension and expansive temporal dimensions in a relational database management system is a little more difficult. Recently there has been a push in the big data computing environment against standard relational tables applied to all data storage problems in an anti-one size fits all movement (Stonebraker and Cetintemel 2005). A database created with the main intention of storing multi-dimensional arrays was created and tested against a traditional relational database management system. Comparing against a traditional relational database management system the multidimensional-array database was found to be 102.5 to 119.1 seconds faster in a single dot product calculation for dimensionally tagged data with speed differences depending on the number of dimensions involved (Stonebraker, et al. 2007). Storing data in array formats has been found to be a good solution in certain applications, especially for storing dimensionally rich scientific data. Calculations are fast and redundant dimensional metadata is reduced.

Slices of the weather data in grids could be converted to and used as raster in a GIS and raster data in a GIS could be seen as XY arrays with spatial tags. In addition to normalizing tables and adding tables and fields which enable the storage of time dimensional data, storing variables from the same dataset in arrays should speed up

المنارة للاستشارات
www.manaraa.com

processing of crop disease forecasting model data, and given the volume of data, doing so should be required in a system providing gridded crop disease risk forecasts with a relational database management system used for storage. Extending the data model of my early version of a crop disease forecasting system using a relational database, arrays should be incorporated in subsequent versions. To be discussed later, this will limit relational database software choices to Oracle and PostgreSQL, two popular packages supporting array data types. RDBMS that support arrays are few because an un-extended Structured Query Language (SQL) that is used with most RDBMS was not meant to handle arrays. The array concept is against the original RDBMS data schema with field intended to contain granular data rather than groups of data.

NoSQL Databases for GIS

What are NoSQL Databases?

The 'No' in NoSQL databases is commonly said to be short for 'Not only'. Usually though, the 'No' can be taken literally. In most cases a NoSQL database does have a query and operational language to do searches, inserts, updates, and deletes on data, but that language does not follow SQL syntax. Besides the lack of standard SQL, a good indicator of a database being a NoSQL database is the database does not rely on a logical data storage schema consisting of tables of fields uniform across records. The SQL language was made to interact with databases holding tables of

المنارة للاستشارات
www.manaraa.com

uniform fields and the NoSQL databases's lack of traditional tables makes a modified or new query language necessary.

Data schemas used to replace uniform tables vary by each NoSQL database but are categorized into four main types. The four main types of NoSQL databases are: key-value pair, column family, document store, and graph database (Stainer 2010). The popularity of NoSQL databases are fairly recent, beginning around 2009, and serve as a general part of the solution to the 'big data' problem. Functionality focused on providing ways to store, index and query spatial data has slowly been introduced to the feature set of a few of the NoSQL databases.

When a geographer thinks of the word "database" tables of records having uniform fields are what typically come to mind. In this case the conception of database really equates to a relational database rather than a general type of database. The possible utility and use of non-relational databases may be slow to catch on because the relational database has been around for decades and it is the technology, in addition to separate data files, usually used to store large amounts of digital geographic data in a structured schema.

Because of the timeliness of the technology there was, as of December 2012, very little academic literature on NoSQL databases used to store spatial data. A Google Scholar search of the terms 'Nosql' and 'GIS' only returns 161 results and most are not directly applicable to GIS applications. NoSQL itself has initially been a corporate phenomena with corporations and employees publishing in the form of journal articles and white papers about engineered or applied NoSQL solutions to

المنارة للاستشارات
www.manaraa.com

their cloud computing and big data problems. Examples include articles documenting Google's BigTable in 2007(Chang F., et. al 2008) and Facebook's Cassandra in 2010 (Lakshman and Malik, 2010). A limited number of academic papers containing examples of NoSQL applied to GIS followed these. Fortunately for the overview, each of three useful results describing NoSQL/GIS systems covers a different main type of NoSQL databases more applicable to GIS. The following paragraphs will briefly explain each data schema of the four general types of NoSQL databases, list NoSQL databases of each general type and the offered support of storing spatial data, and in the case of the three more applicable data schemas one example each of the NoSQL database type applied to a GIS.

NoSQL Database: Key-Value Pair

The data schema for key-value pair is simple; it is a list of lookup keys with each key pointing to what is usually a single value. Instagram, a photo sharing website, uses a key-value pair database to store a list of key-value pairs of unique photoIDs as keys, with unique userIDs as the value. In this way a quick lookup is done for each displayed photo/user pair. Key-value databases feature a fast simple schema, but have limited applicability to store a GIS's entire dataset and lacks good way a way to spatially index locations in relation to each other (Jonas, et al. 2010). Some examples of key-value pair databases include Amazon Dynamo, Voldemort, KyotoCabinet, and Redis. Amazon developed Dynamo to serve data in different

المنارة للاستشارات
www.manaraa.com

systems comprising its online retail site including the shopping cart system. Instagram uses Redis for photo-to-user lookup.

NoSQL Database: Column-Family Data Store

The column-family data store is a NoSQL database type. The data structure consists of rows of data describing individual entities, but those rows in a column-family can contain cells holding columns of more data related to the entity with each column containing potentially more columns at lower tier depth. At any tier a collection of columns can be denoted as a group forming a column family. On a hard drive each group is stored physically in sequence. Speed of aggregations of an individual column family is increased because of their physical location on the hard drive.

To provide an example that illustrates a column-family data store, a column family data store used to store customer information could denote a customer's birthdate and gender as two key-value pairs making up a single column family. Values would be stored on the hard disk in the form [1995-04-23, 'M'], [1994-12-7, 'F'], [1989-7-9,'F'], etc. If a query is done which asks the average age of all women, the query system can go down the two variable column instead of having to move the drive head across entire rows, requiring time to skip over the beginning and ending data for individual records to retrieve only the birthdate and sex of an individual.

Column families at the highest tier within the same parent field share the same number of indexed rows with each index belonging to a single entity. An entity's

www.manaraa.com

uniform index groups the data of that entity across column families. If the system needed to search for all people born in 1994 and return a list of the names of those people, if the *n*th row contained a birthdate in 1994, the *n*th row would be marked as a positive match and the *n*th row of the column family containing the name would then be pulled out.

Xiao and Liu (2011) provide an example of a column-family store applied to a GIS. For GIS data the team had tiles of remote sensing images with each tile from the same tile set having the same resolution with the set, completing a global coverage. If the data were accessed in a viewer program, depending on the zoom level, a different set would be drawn. This is similar to the access pattern and applied column family database technology used by Google Earth.

The system by Xiao and Liu (2011) utilizes the general data schema of a column-family type NoSQL database and works the properties of remote sensing imagery into to that schema. By doing so, access speed gains can be realized by arrangement of pieces of the data within the database. As their system was a prototype the group did not test against a relational database so comparative metrics are not available.

Other examples of column-family stores include Google's BigTable, HBase, and the Facebook-developed Cassandra. Google's BigTable is used for both Google Earth and Google Maps data, both GIS applications. HBase is used in Xiao and Liu's remote sensing image storage system above.

المنارة للاستشارات
www.manaraa.com

NoSQL Database: Document Store

The third type of NoSQL database is the document store. The document within the document store consists of multiple key-value pairs with each value of any data type. Each document describes a single entity. A typical document store implementation example is an Internet message board. Each board post would be an entity with author, contents and comments as different key-values within the document. The value for the comments key comprises a list of comments. Each individual comment can be "record-like" including content, a posting time, and author key-values pairs. Everything related to a post is stored together.

"Two-Tier Architecture for Web Mapping with NoSQL Database CouchDB" presented at a GIS conference in 2011 describes a document store used to store both raster and vector data (Miller, et al. 2011). A document store offers significant flexibility in the document paradigm as the entity can have any set of keys pointing to values of different data types including lists, records, audio-visual media, or geometries describing the entity.

The document store, unlike the column-family store stores all related information about an entity sequentially on the disk drive. Because of this, a document store is best used when all data about an entity is required to perform a task.

CouchDB and MongoDB are two popular document-store databases. Native support for geographic datatypes and location indexing has been limited and differs by each NoSQL database. Currently the GeoCouch add-on CouchDB has good support for vector datatypes and includes support for single and multiple polygons,

المنارة للاستشارات
www.manaraa.com

points, and lines. MongoDB directly supports only points. Both CouchDB and MongoDB support only limited location indexing by r-trees. R-trees speed up queries finding the closest points to a particular point.

Some things which differentiate CouchDB and MongoDB include CouchDB's multi-version concurrency control which stores information updates separately before they are committed to the main database. This allows multi-user access on snapshots of data non-changing from the start of each user's access transaction. MongoDB updates are performed in place without versioning. CouchDB itself can be used as a webserver while MongoDB accessing is done through programs running requests on web-servers or workstations.

NoSQL Database: Graph Database

A graph database is the forth type of NoSQL database. Graph databases conceptualize and connect data records in a way that creates what in computer science is called a graph network. A graph network consists of nodes (points) and edges (lines connecting point nodes). In a graph database the nodes are entities and the edges describe the relationships between any two entities. Properties are stored for both nodes and edges. Graph databases were commonly developed and used for Internet social applications with nodes containing information about users and the edges containing information about the relationships between users. A graph database comprises an implementation of a network in addition to the algorithms created to traverse the network thereby answering distance queries. This makes a graph database

المنارة للاستشارات
www.manaraa.com

apt to store a vector-based geographical transportation network (Baas 2012).

In order to compare performance between a relational database and a graph database Bart Baas, a Master's student in a GIS program chose PostgreSQL(a relational database) and Neo4j(a graph database), and compared each testing the speed in loading and querying a road and feature point network. Stemming from the quality of the location indexing implemented in PostgreSQL performing Euclidian point-to-point queries was faster with PostgreSQL than performing the same queries with Neo4j. However, Neo4j queries were faster when querying point-to-point queries along the transportation network. This is because the network is already connected within the database schema and does not have to be first built through additional queries as it does in the relational database (Baas 2012).

NoSQL Database Discussion

A running theme in the NoSQL movement is that the nature of the data to be stored and the pattern of data access should be taken into account when choosing the technology to store and serve that data. If this process is well thought out the strengths of the database technology facilitate faster data access of that particular type of data. This is in contrast to the relational database concept where a single solution, data tables in forms that reduce total disk space and redundancy is applied to various data storage problems. Diminished focus on the data table also leads to designed data schemas of systems to taking different shapes. The different data schemas are starting points for possible designs that speed up access based on the pattern of access of

المنارة للاستشارات
www.manaraa.com

pieces of data. The benefits are apparent in the Baas thesis where a database taking the form of a graph network quickly answers a point-to-point distance query traversing a transportation network.

Scientific Data File Formats for GIS

NetCDF files, as an example scientific data file format, are unlike relational and NoSQL databases in that the files do not have a database management system attached to them. Data within netCDF files consist of sets of indexed multidimensional arrays. While preparing a netCDF file, before a variable can be stored, each dimension, such as the y index in a grid or a valid time, are first created within the netCDF file. These created dimensions can be set to either be limited, or occurring within a range, or unlimited, a dimension increasing in length, as with progression in time. Each possible value a dimension could take is attributed an index, from zero to the number of unique values the dimension can take. NetCDF4 is a subset of HDF5 technology with netCDF4 conceptually a combination of netCDF3 indexing and HDF5 file hierarchy.

There are a number of articles from the past decade covering projects which take gridded data in an HDF4 or HDF5 scientific data file format and convert them into a raster format desktop GIS programs can readily use to do analysis and modeling (Zhao, et al. 2011; Bachoo, et al. 2008; Abdella and Alfredsen 2010, etc.). These are attempts to bridge scientific or meteorological data between the atmospheric and other science fields for use in the GIS field. The end product to the

www.manaraa.com

conversion process usually consists of standalone spatially referenced rasters or rasters in an ESRI raster catalog. There are weaknesses in this approach if this converted data is to be used in further modeling though. These weaknesses include a lack in ease of use in management and retrieval of numerous inputs if modeling is to be done as well as speed issues in the process required in performing those calculations in a typical GIS.

A collection of a large number of rasters is in general unwieldy, hence the larger atmospheric and science community's usage of scientific data formats. Retrieving needed temporal data slices through indexing by time is also difficult, because unless raster bands are used, each raster is limited to a single slice. Raster catalogs in ArcGIS partially solve this problem with time-based lookups. Still numerous records each containing a single hour's worth of data must be looked up for each variable and for each model day.

As mentioned earlier in this document, spatial data in a GIS must be checked to see if each raster's geometry directly overlays to perform clean raster algebra. Since the grid system is pre-defined checking spatial registration of layers should be unnecessary. An alternative approach available to ArcGIS is converting raster data to arrays in Python scripts before calculations using the provided ESRI Python function to do so. At some point in the conversion process of this data from the original GRIB file format to raster the state of the data was closer to already being arrays in memory. It does not make sense to complete a conversion process only to reverse part of that process in order to perform repetitive calculations for modeling. Part of the process

المنارة للاستشارات
www.manaraa.com

would be performed again to store model output in converting output arrays to raster and again setting the spatial registration. Storing and processing the data in arrays avoids the spatial registration of all of the input data in the conversion process and the spatial registration of all output data in addition to the conversion of raster data values to array data values.

Chosen Technologies

Because of the diversity of options when selecting from storage technologies, and the time required to implement each technology, three were chosen to implement and compare. The superset of technologies considered included Relational Database Management Systems, NoSQL databases, and scientific file formats. One technology per technology superset was chosen. Rationale behind choices supported by knowledge of the features of the technology is given below. The following Methodology chapter will then detail how these technologies are implemented.

PostgreSQL: A Relational-Database Management System

PostgreSQL was chosen to represent Relational Database Management Systems(RDBMS). In choosing a RDBMS solution for a GIS, consideration was given to the fact that Western Michigan University's Geography Department has access to ESRI's ArcSDE technology. ArcSDE serves as a bridge between ArcGIS and the chosen relational database. Both attribute and spatial data are stored in the backend database and ArcSDE manages the spatial properties of that data. Since

المنارة للاستشارات
www.manaraa.com

ESRI products dominate the GIS software market in the US, it makes sense to limit the relational database choices for this analysis to those which ArcSDE can connect to. Doing so enables visualization of the data through ArcGIS without transforming the data. This removes MySQL from one of the choices, as ArcSDE cannot connect to a MySQL database. Remaining choices ArcSDE is able to use as a backend are IMB DB2, IMB Informix, MS SQL Server, Oracle, and PostgreSQL.

PostgreSQL is able to store gridded data in a simple multidimensional array datatype. Comparing database packages, at the time of this document, Microsoft SQL Server does not yet offer any comparable array datatype. Oracle contains three different data-types able to store arrays, but the version suited for larger amounts of data consists of nested tables. Use of Oracle for array data would involve creating a system more complex than needed for the task at hand.

Oracle, IBM DB2, and IBM Informix require license fees. PostgreSQL is free and open source. For the ability to store arrays and being able to connect to an ESRI ArcGIS ArcSDE server instance PostgreSQL was chosen as the representative RDBMS to be compared against the representative NoSQL database and scientific data file format.

MongoDB: A NoSQL Database

A document store type of NoSQL database has features able to serve the data involved in a crop disease forecasting system. Other possible NoSQL alternatives include the column-family store and graph database. The column-family store seems

المنارة للاستشارات
www.manaraa.com

overly complex for use in this system because each piece of weather data in the system has only one set of dimensional attributes. Column family stores excel at applications where pieces of data have multiple repeating attributes. Lacking multiple instances of metadata, a table in a column-store for a crop disease forecasting system would look like one in a relational database. Graph databases are suited for data with networked schemas like a social or transportation network. Unlike the column-family store and graph database the document store offers a flexible document paradigm where time attributes and related metadata can be stored in fields within a document and the actual data can be stored alongside it. Since this is a typical problem for forecasting, we select document store type as our example.

Although there are quite a few alternatives the two most popular types of document store NOSQL type seems to be MongoDB and CouchDB. Both of these technologies were created first and foremost to serve data over the internet. The developers of CouchDB had a focus of CouchDB as not only a repository of data but also a web server to serve that data. For CouchDB, like most document stores, access is mostly done through HTTP language. Internet documents containing HTTP language are used as the Application Programming Interface (API). MongoDB as an alternative is meant to be separate from the web server. APIs available to developers include most of the programming and scripting languages in use right now. Since code is used to generate substantial model output in a crop disease forecasting system MongoDB is more suitable than CouchDB as a document store applied to a crop disease forecasting system.

المنارة للاستشارات
www.manaraa.com

NetCDF: a Scientific File Format

NetCDF4 was chosen from among alternative scientific file formats such as GRIB and HDF5. Forecasting data is issued from the National Weather Service in the GRIB format. The GRIB format has a weakness of storing data only in two-dimensional slices. Ideally data would be retrieved in chunks rather than individual time-slices. NetCDF4 contains a subset of HDF5 and actually writes HDF5 format files. Both can access multidimensional chunks of data. However HDF5 lacks shared dimensions in its standard data model. NetCDF4 enforces dimension declaration. Variable data must take the shape of the declared dimension. For example, if x, y, and time dimensions were declared, the cube shape of all data variables must be the same. This insures consistency when slicing data at the same dimension for different variables. For this reason the netcdf4 subset of HDF5 is chosen as the scientific data file format to compare.

To meet requirements of a crop disease forecasting system able to add future NWS grids and apply those grids to various crop disease forecasting models the processing and storage methodologies of that system must feature expandability. An expandability requirement in terms of storage can be best met by utilizing netCDF files for storing gridded data. NetCDF is faster than comparable Relational Database Management Systems for access of geospatial data (Cohen et al. 2006). NetCDF can store spatiotemporal data for cell regions with indexes (Goodall and Maidment 2009), an aspect that can be used to speed up processing. Spatiotemporal data is best retrieved given both time and space subsets to save time, netCDF is able to do this

المنارة للاستشارات
www.manaraa.com

(Weigel et al. 2010; Zender 2008). The hierarchical storage structure categorizing and separating types of data offered by recent versions of netCDF should best serve expandability requirements of the a proposed crop disease risk forecasting system.

NetCDF was developed to store large datasets of multidimensional data including spatiotemporal data. The format is commonly used by governmental agencies (NOAA, NASA, U.S. Navy, etc.) for storing datasets used in climatology, meteorology and oceanography. NetCDF will likely be able to facilitate goals of a crop disease forecasting system requiring expandability in terms of adding gridded datasets and applying gridded datasets to various models.

Why Rasters or Shapefiles Were Not Used As Test Storage Platforms

Within the GIS modeling community standalone shapefiles, rasters and tables stored in disk directories are commonly used in modeling because these standalone files require low initial time resources to incorporate data into a system running a model. By using standalone files the modeler can devote more focus on the model itself and creation of a model processing system instead of having to be concerned with how data is stored. This significant benefit to using standalone files can be outweighed by their weaknesses during a broader operational system life cycle. A system's life cycle could include initial planning, development, execution and the refinement of the system running the model. Standalone files can be easily moved from directories and be renamed, but raster files used within a model may be numerous and it may be hard to differentiate names. Commonly if the project is large

المنارة للاستشارات
www.manaraa.com

and prolonged, analysts working on a project change during a system life cycle. Initial effort is required for each new analyst to understand where each file is or what a data each file contains. Other weaknesses of standalone files include possible data redundancy in tables, where for example, a weather station's latitude and longitude is stored numerous times, once for each observation. Lacking a decentralized repository, data in standalone files may reduce functionally in the application of the same gridded data to more than one model as directory paths and filenames are not static.

It would be possible to create a system using shapefiles or rasters within a file hierarchy but such a system would be cumbersome to manage, hard to implement for expandability, and likely take longer to process model data. Ideally national data would not be regionally subsetted beforehand. Subsetting the data at the time of running the model in the case of shapefiles, for each time slice, involves two time-consuming processing operations. First, the data would have to be extracted for only the region concerned. This would probably work fastest if done by selecting correct values of y, x indexes of each defined cell instead of performing spatial comparisons of location where in the case of NDFD data at 5km resolution greater than 750,000 cells would be compared against a regional boundary shapefie for each hourly slice to see whether each cell is in a region or not. Still this data would not be rectified with other slices. Second, the data does need to be spatially referenced with other slices. Traditionally for shapefiles this would be done with an attribute or spatial join. An attribute join would likely be faster. Faster still would be to fill up a multidimensional array by slice during the first operation scanning for correct y, x indexes. Once the

المنارة للاستشارات
www.manaraa.com

data is in a multidimensional array, model computations should be processed more quickly than by cell location. So the end goal should be to get the data into a multidimensional array. Systems using a relational database, NoSQL database, or scientific files as compared to shapefiles should have a less processing intensive path to bring data into multidimensional arrays.

Python libraries able to work with raster data usually have functions able to convert a raster into a single slice two-dimensional array. The slices can then be stacked and sliced regionally.   So working with raster data as compared to shapefiles in this case may be a more viable option. On the other hand, managing a directory and file hierarchy of all of a season's model data along with output is cumbersome in the conversion of grids. Either incorporating the metadata into file names or within the raster's own schema is both cumbersome as well as increases redundancy. The set bins used for categorized data of PostgreSQL, NoSQL, and NetCDF files removes most of the redundancy of metadata, as for example a temperature grid inside a bin containing temperature grids by the bin's definition would have Fahrenheit units.

Another option is using an ArcGIS raster catalog. Querying the needed dimensions along with a sort would yield an ordered selection of the catalog. An iterator would then in turn take each record and retrieve the raster's logical location in storage then load that raster, convert it into a two-dimensional array, and stack subsequent rasters. A downside to using raster catalogs is individual areas or pixel stacks cannot be directly queried without first querying by table attributes the entire raster. Raster algebra could be used instead of algebra done on mutli-dimensional

المنارة للاستشارات
www.manaraa.com

arrays, but then the rasters have to be lined up first and certain operations cannot be easily done with raster algebra, where the sequence of values at certain locations must be taken into account.

Generally the focus should be using the gridded data in a way that utilizes the property of particular NOAA grid specifications in that after the specification is made, it does not change. Spatially referencing a grid in raster algebra or doing a spatial join with vector data should not be needed because it is known that the data overlays. Spatially referencing each shape is an expensive operation given the number of times it would be done for all of the time-slices. For this reason common GIS storage and processing method in commercially available packages is not suitable for this volume of data processing to be done in a timely manner.

المنارة للاستشارات
www.manaraa.com

CHAPTER III

METHODOLOGY

This section will explain the methods used to develop, test and compare the three chosen alternative storage technologies. The logical data structure, optimizations, and a general access pattern for each of the technologies will be described. Comparisons are made among the technologies of training two crop disease forecasting models that vary by spatiotemporal scales. The impact of the size of data chunks retrieved during access and processing of queries is examined with respect to indexing data into subsets by region and time.

Methods for the PostgreSQL System

PostgreSQL: Logical Data Structure

The data structure in PostgreSQL (Appendix A) includes tables holding forecasting, validation, and derived gridded data along with various metadata. Forecasting and validation data are stored in two-dimensional arrays in the size of the national grid within records of appropriate tables. As some forecasting data can take the form of categorical descriptions of forecasted weather types, such as lightning storms occurring with rain, a field to hold two dimensional text arrays is also declared in the tables. For any record that contains numerical data within an array the text array would be empty. Empty arrays in PostgreSQL only use the space for a data

المنارة للاستشارات
www.manaraa.com

pointer so there is not a significant storage cost incurred to be able to handle text arrays in addition to numerical arrays.

The Validation table contains associated metadata including the valid time for each two dimensional slice. Both the reference time (when the forecast was made) and the valid time (when the forecast is applicable) are stored for forecasting data. A dataset field acts as a foreign key to the Dataset table. The Dataset table stores metadata about a given dataset. Any metadata about a dataset can be stored in a text string containing a JSON record. JSON (Javascript Object Notation) consists of an extendable record. This field could store additive metadata about a dataset including the issuing agency and any other notes. The var field is a foreign key to the Variables table. The Variables table can contain metadata about the variable and link the variable to the originating dataset. Potential metadata related to a variable could include the timestep information about the variable. If the variable contains timestep notes whether the accumulations occur before or after a valid time could be indicated. This metadata can be assessed by those managing or running models and should reduce confusion related to what the variables are conceptually.

The tables used to run models include: Derived_Params, Model_Params, and Model. The Derived_Params table can hold derived in-between and model output data contained in arrays within records. These arrays are associated with region, crop, and model. This schema allows the derived grids to be associated with both a region and a crop. In some model applications a mask can then exclude areas not growing the crop of interest. Descriptive metadata about a single parameter from a single

المنارة للاستشارات
www.manaraa.com

model can be stored in the Model_Params table and can be used similarly as the above mentioned Dataset table and Variables tables storing respective metadata. The Model table contains metadata about a given model.

Cells of National Grid and Cells of Regions Grid tables are designed in a way that ESRI ArcSDE can manage the spatial properties of individual cells making up a grid. Table views made on the PostgreSQL server dynamically link forecasting, validation and derived data from arrays to these records. Records containing geometry can then be displayed in ArcGIS for Desktop without designing programming tools to facilitate this. For this reason only two dimensional array data slices are used for PostgreSQL-based system because designing table views to fill in three-dimensional data to individual cells would be more difficult and complex and possibly reduce model performance.

PostgreSQL: Optimizations

PostgreSQL optimizations include record indexing for each table by the fields underlined in Appendix A. This allows faster lookup when querying by those indexed records. B+-tree indexes, which speed up range queries, with querying for forecasting times falling within a range as an example, are used. Queries for records containing two dimensional data slices of a given variable within a given time frame are fast using B+-tree indexes.

المنارة للاستشارات
www.manaraa.com

Determining settings for optimized compression is not necessary. The PostgreSQL server manages compression of all data stored. The system designer has no control over what data is compressed or the levels of compression.

PostgreSQL: Data Retrieval During Model Runs

Executing each model relies on two researcher-written python code files. The first file contains specialized code for an individual model. The second file, a library of functions, can be used for various models and provides generalized functionality for accessing grids within a PostgreSQL database. A separate open-source library pyscopg is used to access and perform queries on the data on the PostgreSQL server. The generalized steps for retrieval of a sub region from available national gridded data are listed below:

1. A database connection object is made to reference the database. This is done in the individual model's code.

2. The target region boundaries are queried from the database.

3. The model's local start time and end time is determined and then converted to UTC.

4. Per each model input variable, a query is issued for the appropriate time range.

5. The two-dimensional arrays from returned records are spatially trimmed by regional boundaries retrieved in step two and are inserted into a three-dimensional array at the temporally correct positions.

6. Linear interpolation for missing hours is done if necessary.

المنارة للاستشارات
www.manaraa.com

Running models using PostgreSQL will differ from the MongoDB and netCDF systems in that only single slices of national data would be loaded into memory at a given time. This means that two-dimensional national coverages are brought into memory and then trimmed spatially down to the region before more individual slices are loaded. A system using this method requires less computer memory to run compared to systems loading a three dimensional cube national array before spatially trimming the array.

Methods for the MongoDB System

MongoDB: Logical Data Structure

MongoDB offers flexibility in representing data via the document store NoSQL paradigm. All documents related to a single season are stored within a single collection and that collection's sub-collections, stemming in a hierarchy from the root collection store national forecasting and validation grid or regional derived model output grids (Appendix B).

Metadata and documents containing spatial information about national standard grids are stored in the root collection. To increase the optimization of lookup indexes and to reflect epistemological differences, documents representing national forecasts and validation were put into two different sub-collections. Documents storing validation data need only an indicator of the time for which the data is applicable. In addition to the time applicable, forecast data also needs a reference time for when the forecast was made. Storing validation and forecast data in different

المنارة للاستشارات
www.manaraa.com

collections allows an index to be created for validation data on the combined {Datasetname, Variable, VLDDate} fields by avoiding containing unrelated forecast records not having a the VLDDate field. If validation and forecast grids were in the same collection, results from a search using the Datasetname, Variable, and VLDDate index created with default options would contain results with null values for the VLDDate field for all forecast documents, which would not be useful.

Designing a storage sub-system to serve data using a document store allows for flexibility in design. Two versions of systems using MongoDB were created. In both versions a validation document contains a multidimensional array of values for one variable twenty-four hours deep with a UTC date stamp in the document's ValidDate field. The versions differ by the way forecasting data is grouped together. In version one all forecasting data for the same reference time for a variable is stacked and stored together. In version one a forecasting document for most variables contains around 157 hours of forecast grids made at the same reference time. This includes hours of missing data within the stack as forecasts are provided for every third hour for most of the variables. The second version of a system using MongoDB does not store forecasting data in three-dimensional stacks with the same reference time. Only single two-dimensional grid slices containing valid data is stored.

The rationale behind stacking forecast data in version one follows. Typically crop disease risk forecasting models use weather forecasting variables with identical reference times. To store the range together in one chunk means that all the forecasting data relating to a single variable can be retrieved in a single data lookup.

المنارة للاستشارات
www.manaraa.com

If the data were stored in hourly slices, the system would be required to do hourly lookups and retrievals. To have the validation data in twenty-four hour UTC chunks, however, should mean a maximum (n + 1) lookup and retrievals per variable for per day, with n being the number of days of data needed. A reduction in lookups and chunk accesses should greatly reduce retrieval times when models run.

Ideally each grid would be stored in the same document as that grid's metadata. Unfortunately MongoDB imposes a 16 megabyte (mb) limit for each item within a document. Some national grids after compression are larger than 16mb so they cannot be stored in a single document. To deal with items larger than 16mb developers of MongoDB incorporate a pseudo-file based system of storage. The pseudo-file system, GridFS, divides an object, in this case the multidimensional array into chunks 16mb or less. Information about the array is stored in a '.files' sub-collection and the data itself is stored in a '.chunks' sub-collection with lookup ids bridging the two. This schema adds at least two lookups for national variables but large data reads remain at one or two logical data chunks. Regional reads and writes will tend to be smaller than 16 megabytes and will remain at single chunk reads.

To enable a multi-model and multi-region crop disease forecasting system documents containing derived data have a field indicating which model the output variable relates to. By including a region field the same model can also be applied to the same crop type in different grid-defined regions of a country, possibly storing regional masks for non-important areas within a region. The model field can then be used to group variations or versions of models, i.e. different parameters for

المنارة للاستشارات
www.manaraa.com

forecasting and validation or sets of possible neural network inputs. The parent_dataset field can be also be used to vary the source input while trying the different above variations in the model.

MongoDB: Optimizations

Optimizations for MongoDB include lookup indexing and compression. Indexing is done on particular fields marked by underlined keys within the schema figure found in the appendix (Appendix B). The MongoDB server itself does not offer compression so compression is achieved with the Python lz4 library. A previous system I created with MongoDB as the storage technology used the zlib library for compression. Use of lz4 in the versions created for this thesis proved to perform compression and decompression much faster than the zlib library.

MongoDB: Data Retrieval During Model Runs

Executing each model relies on two researcher-written python code files. The first file contains specialized code for an individual model. The second file, a library of functions, can be used for various models and provides generalized functionality for accessing grids within a MongoDB system. Region definitions are stored in documents contained in model collections using four-corner bounding indexes.

The generalized steps for retrieval of a sub region from available national gridded data are listed below:

المنارة للاستشارات
www.manaraa.com

1. A database connection object is made, referencing the collection containing the season's gridded data. This is done in the individual model's code.

2. References to locations of sub-collections within MongoDB containing forecasting, validation, and model output are sent to the function to run the model.
3. A model's start time and end time is determined and then converted to UTC.

4. The function can either 1) piece together a filename or filenames to do a direct lookup in the files sub-collection or 2) do a lookup by [Dataset, Var_Parameter, REFTime] to get the MDA_Id representing a record in the files sub-collection.

5. Using either of the two lookup methods the multi-dimensional array chunks are first retrived from MongoDB and then decompressed and read into a numpy Python multi-dimensional array data-type.

6. Within Python, retrieved chunks are pieced together, the national data is trimmed to regional, temporal linear interpolation is done if requested, and time overhangs for hours which are not needed are then trimmed.

The lookup/retrieval of large initial national multi-hour chunks of data in a MongoDB system is less complex than either the system based on PostgreSQL or netCDF (the netCDF description follows). A trade-off is, in especially the first version using MongoDB, that larger chunks are loaded into memory and then have to be trimmed. Trimming the data in Python involves additional memory and processing overhead. As designed, the Python process running the PostgreSQL system only has to subset regional axes of single slice. After the data is first retrieved, the Python process using the netCDF system subsets neither region nor temporal axes. Loading larger chunks in the MongoDB system creates a greater initial RAM requirement for the modeling process running in Python. However, even with the larger memory requirement and the extra subsetting needed, the additional time required to do so will

المنارة للاستشارات
www.manaraa.com

likely be small because of the speed of CPU paired with RAM processing and access operations.

Methods for the NetCDF System

NetCDF: Logical Data Structure

Tree diagrams (Appendix C) show the content hierarchy of one season of data within a netCDF file. A number of leaf nodes are not filled and act as placeholders for data possibly used in future modeling. Additional branches within a hierarchy could be appended based on future needs.

Forecasting and verification data are logically separated even though the same grid specification is shared between the two. In terms of attribute dimensions, the only difference between forecasting and verification data is that verification data does not have a reference date; both forecast data and verification data have a date for which they are valid. Regional model results for different crop disease forecasting variables are also saved on different branches with their own subset of x/y grid indexes. This introduces very small amounts of redundancy in stored index values in exchange for being able to functionally keep validation and forecasting data on separate branches of the hierarchy.

Declared variables in the diagrams are marked by the numbered index of the dimensions attributed to them. A branch containing gridded data also contains gridlat and gridlon variables, each with x/y grid dimension indexes. These can be used to

المنارة للاستشارات
www.manaraa.com

create shapefile and raster versions of the data as well as feed spatially auto-correlated regression tools or kernel-based transformation functions.

NetCDF: Optimizations

Chunking and data compression are two optimizations the system uses, both set at time of declaration of variables. The chunking parameter sets minimum multidimensional blocks of values physically stored sequentially together within a storage medium. Data compression reduces data read time and storage footprint while increasing required processing resources. The chunking and data compression options used will be described in turn in the next two subsections.

Chunking

While running regional models from national data, regional subsets are sliced from the superset by y, x, and time dimensions. Without chunking, data is physically stored in index order sequentially. When a subset of the multidimensional variable is requested within the physical storage system, if it is a traditional spindle-based hard drive, the read head of the drive has to make large non-sequential access jumps from where data related to the sliced maximum indices jumps to the data related to the next minimum indices to where needed data is physically located. Dimensional chunking allows data to be stored in chunked order rather than only index order.

To speed up access times, ideally chunks reflect the typical dimensional sizes of the slices requested at read time. Since need sections of grids will be different

المنارة للاستشارات
www.manaraa.com

depending on the region, not partitioned into equal rectangles, moderately small subsets of chunked areas will be used for this study. This reduces the chance of having to read a large overhang of grid cells occurring with a large remainder of edge chunks. For forecasted grids the reference time dimension, when the forecast was made, and the valid time dimension will each be single-index chunked. Data will not be chunked by time.

Data Compression

The system will use zlib compression on all variable data handled by the netCDF library. Compression of the data will be lossless without early truncation of floating point values. This reduces access time but may slightly increase processing time. Storage access tends to bottleneck systems, so this exchange likely improves overall performance.

NetCDF: Data Retrieval During Model Runs

A forecasted temperature for a given cell would have dimensional attributes including forecast reference date, forecast valid date, center point latitude, center point longitude, and cell id. Upon data retrieval, subsets of area by longitude and latitude as well as other dimensions are stored in x and y array slices. Considering this aspect of netCDF, gridded data can then be processed via indexes in multidimensional arrays within the Python programming language. Arrays vertically register by x and y dimensions, similar to how data lists for a point location create a data vector.

المنارة للاستشارات
www.manaraa.com

Processing this way would be similar in ways to how climate modeling is done for cell centerpoints in climate and atmospheric sciences on grids with a standard geometry.

Use of a GIS software package, ESRI's ArcGIS, would take place in this kind of processing methodology at three times within the development process and forecasting cycles. First, occurring once at the introduction of a new grid standard, the national NWS grid would be indexed by cell and then referenced by region. Later data retrieval of regional subsets needed for separate areas related to particular crop diseases is possible because of this step. Secondly, non-forecasting data, such as generalized climate and recent actual observations at points that are a part of a crop disease forecast still need to be interpolated once for new climatology data and cyclically for observational data. This is done by ArcGIS with output geometry of grids ideally coinciding with the regional subset format of the standardized NWS grid for a given crop disease risk forecast. Thirdly, after processing of inputs by an artificial neural network or separate statistical methods, visualization and distribution of crop risk forecasts are done using a GIS. Utilizing the standardized grid to pre-establish data vectors also cuts out an additional middle process of a GIS having to format inputs into vectors usable as inputs for the artificial neural networks or other statistical methods. Outputs themselves will be stored within the same netCDF file as forecast inputs. Those outputs can then be sub-setted by dimension into multidimensional arrays usable as input in artificial neural networks or other statistical method processing.

المنارة للاستشارات
www.manaraa.com

As an example, given a single growing season of data, a time dimension would be defined, such as the number of hours since an arbitrary standard reference time. The first index value of the first time dimension would always be 0, stored at that index location would be the difference in the number of hours since the standard reference of the first hour of the season. In this way, a one dimensional array, representing a single dimension, in this case time, can have values stored at indexes with those values compared to a target time range when a variable slice of a multi dimensional array is later requested when forecasting.

During a request, if the time values at the index location meets the criteria of the desired times at the times' associated indexes, values for that requested multidimensional variable will be returned. In this way instead of a key field within a record of a table to be queried, (as in a relational database), data already indexed for each dimension is stored. Range (greater than/equal to and less than/equal to) queries of desired dimension values as well as equality queries for single slices can be answered quickly without maintaining separate indexes as is standard in relational databases.

After dimensions are declared and values are attributed to dimension indexes, variables to store data are then declared. On declaration a variable is given a name, a datatype, and an ordered set of dimensions associated with it. Order of dimensions once declared cannot change. Metadata for the new variable can be set through variable key/value pairs. Any key name not already used by the variable can be chosen to store metadata. For example, a key 'units' is usually set with the value

المنارة للاستشارات
www.manaraa.com

containing a text string containing a description of the variable's units. Later, a program running a crop disease forecasting model can query both the order indexed dimensions occur in addition to any metadata set at declaration.

Executing each model relies on two researcher-written python code files. The first file contains specialized code for an individual model. The second file, a library of functions, can be used for various models and provides generalized functionality for accessing grids within a netCDF file. Aside from providing access, the generalized library also contains time conversion, variable unit conversions, Grib file to netCDF file conversion, regional slicing tools, region definitions, gridded statistics, and gridded model output formatting and conversion. Region definitions are done by max and min latitudes and longitudes.

The generalized steps for retrieval of a sub region from available national gridded data are listed below:

1. A data access object is made referencing the tree node within the netCDF file containing the gridded data. This is done in the individual model's code.

2. A RegionSlice object is made using both a keyword of the desired region and the above access object. The RegionSlice object contains metadata about the desired region later used to slice the national dataset.

3. A model's start time and end time are determined and then converted to UTC.

المنارة للاستشارات
www.manaraa.com

4. The VariableSlicing function is called and returns necessary data based on passed slicing function parameters. Parameters to the VariableSlicing function include the Access object made in step 1, the name of the variable to be returned, and a boolean indicating whether temporal linear interpolation is necessary, the RegionSlice object made in step 2, and the start and end times determined in step 3.

The VariableSlicing function was made in a way to be able to access variables having different dimensional attributes. As long as references to all required dimensions are passed to the function, the function accessing the dimension order stored in the netCDF file is able to arrange the requested dimensions properly. A main focus in creating the function was to access both forecasting data (with a reference time for when the forecast was made) and validation data (lacks a reference time) with the same function.

One possible weakness is that the VariableSlicing function only handles lateral slices greater than a single width in only the y, x, and valid time dimensions. Unless a comparison is needed between forecasts of different reference times this should not be a problem.If a comparison between reference times is necessary, the Variable Slicing function could be called twice, using two different reference times.

Specifications for Designed System Comparison

Hardware

The computer running the test models is an up to date higher end personal computer as of 2013. It has an Intel i7 processor model 3770-k and 32GB of ram.

المنارة للاستشارات
www.manaraa.com

Drives to retrieve and store model data include a 3TB Seagate Constellation CS drive and two 256GB Samsung SSD 840 Pro drives. The drives are connected via a LSI 9271-i8 RAID card. The hardware cost of the system would be under $1900.

Three tests per test model will be done to see the impact of type of storage hardware used has on the speed of running models based on the storage method. Raw speed of the Seagate drive should be around seven times slower than the Samsung SSD. Certain storage types may derive different benefits from use of an SSD drive because of different patterns of access. The RAID -0 pair of SSDs should roughly halve raw access times of the single SSD. RAID-0 is a technology that groups two or more disks of the same type together and utilizes each drive's read/write controller to concurrently write successive units of data.

Software

The operating system installed is Linux Mint 14.1 (MATE version). Python 2.7.3 will be used to run models. Python itself is a run time interpreted language where some speed gains are lost by not utilizing optimizations possible with languages that are pre-complied. Yet Python offers the numpy package. The numpy package is able to process multidimensional array data quickly. Standard Python programs are limited to running on a single processing core. This is a significant weakness of the Python language in today's multiple core computer hardware. Python derives other benefits from code readability and ease in model prototyping.

المنارة للاستشارات
www.manaraa.com

Quantitative Assessment: Speed Tests

Three sets of speed tests varied by disk technology will be performed for two different test models running four years worth of data deriving daily outputs for five month seasons over four years. The test models used are described below.

Test Models Used

Initial Case Example: A Wallin Potato Model Forecasting Potato Late Blight

This first model requires twenty-four hour slices of relative humidity and temperature variables. A binning of hourly relative humidity is partitioned by three hourly temperature ranges with hourly accumulation in that bin if the relative humidity at a location is over eighty percent. Based on the amount of accumulated hours in a bin a total risk is assigned per location (Baker and Kirk 2007; Baker, Lake, Roehsner, et al. 2012).

Second Case Example: Model Deriving Daily Inputs and Daily Target Output For Training of a Neural Network to Predict Risk of Barley Head Blight

A model predicting head blight of barley was chosen as a second test model to implement. The estimation model takes wide 240 hour slices of gridded temperature and dew point from the Great Plains area to calculate a single day's estimation of risk. The following operations are then carried out for the estimation model used as the daily target output in training a neural network to forecast head blight of barley (Bondalapati, et al. 2012):

المنارة للاستشارات
www.manaraa.com

1. Relative humidity is found for all dew point values in the multidimensional array given each temperature value.

2. The average temperature by location is calculated.

3. A wetness metric is calculated by first taking the relative humidity multidimensional array and assigning True or False depending on whether the relative humidity was greater than 90 percent. Then the sequence of relative humidity values over the 240 hours are processed by location and is assigned a Wetness metric based on the pattern of hours with relative humidity over 90 percent. The wetness term is given bonuses whenever there are consecutive hours of relative humidity over 90 percent. An intermediate, the binary sum of relative humidity values over 90 percent, is also saved.

4. The average temperature and wetness metrics are combined to determine a Weibull value for the pair based on a Weibull distribution.

5. A cutoff is applied to the Weibull value to determine estimated risk.

6. The average temperature, relative humidity binary sum, wetness term, and Weibull value for the day are then saved in the format of the respective test storage system.

The head blight of barley forecasting model utilizes both forecasting grids and validation grids. 120 hours of validation data reflecting what occurred the first five

المنارة للاستشارات
www.manaraa.com

days of a ten day window and 120 hours of forecasted data reflecting what is predicted to occur in the second five days of a ten day window is combined to forecast risk for the tenth day. The steps are somewhat similar to operations done for the above validation data except five days of forecasting temperature and relative humidity data is first accessed and then appended to the first five day slices of the above temperature and relative humidity verification data. The steps carried out from the 3rd step onward are the same.

This test model was chosen to test the storage methodologies' abilities to moderately scale for larger model timespans along with a larger spatial area, lending the trials to retrieval and processing of more data. The test model also contains a Wetness calculation that would be difficult to do using a multi-dimensional array operation. While performing the wetness calculation the array must be sliced spanning the time axis by each location. Many long thin slices must be made.

Speed Test Description

For each of the two models, for each of the three chosen data storage methodologies, for each of the three storage hardware categories the models will be run over four growing seasons, 2009-2012, each season consisting of five months. Initial states of the hardware will be the same with the computer booted and in the case of PostgreSQL and MongoDB servers will be started an hour prior to the trial allowing for any indexes to be loaded.

المنارة للاستشارات
www.manaraa.com

Qualitative Assessment: Expandability Test

The amount of time resources needed and general ease of adding the model forecasting for head blight of barley after the potato late blight model is complete will be considered. Adding the second model will involve adding a region to the system and finishing the model by coding the model aided by researcher created libraries used to access stored data used with the first model.

المنارة للاستشارات
www.manaraa.com

CHAPTER IV

RESULTS AND DICUSSION

This chapter will first describe benchmark quantitative results differing by crop disease forecasting model, storage technology used and storage hardware. Then those quantitative results will be discussed in turn, grouped by storage technology. Thirdly qualitative results will be covered. Lastly a concluding discussion will be given. Even though each storage technology works quite differently, performances when using a traditional platters-based hard drive were somewhat similar but with a few interesting differences. Therefore, quantitative factors become important when choosing a storage technology for a crop disease risk forecasting system.

Quantitative Results

This section first displays quantitative benchmarks of running two different models varied by different storage methodologies as well as storage hardware. Two models were initially chosen to test performance of the different storage methodologies chosen. Requiring different sizes of spatiotemporal dimensions by model indicate how well storage methodologies scale as larger dimensions increase the amount of data needed to be retrieved.

A model forecasting potato late blight risk for a Michigan region is the first test model used. A rectangular area consisting of 18,354 cells and 36 hours worth of forecasting and observed gridded data is retrived each day. Five month growing

للاستشارات المنارة
www.manaraa.com

seasons are assumed with derivation of daily risk forecasts and validation estimates provided for four growing season. The two needed variables are temperature and relative humidity and both sets of validation and forecasting data is retrieved.

A second test model involves deriving daily inputs and deriving a daily target output of head blight of barley risk to be used to train a neural network for a tri-state barley region including North Dakota, South Dakota, and Minnesota for five growing seasons consisting of four months each. A rectangular area consisting of 39,446 cells and 240 hours worth of observed gridded data is used. A rectangular area consisting of 39,446 cells and 120 hours worth of forecasted gridded data is used. Like the model forecasting potato late blight risk the two needed variables are temperature and relative humidity. The spatiotemporal resolution of needed data is much larger for a model forecasting head blight of barley. To reduce repetitious loading of observed data, as a subsequent day uses 226 hours of data needed for the previous day all validation data needed for a season is loaded at the beginning of processing of that season.

The tables below first show the total time required to run four seasons of a potato late blight model and then show the total time required to run four seasons of a head blight of barley model. Times are in decimal minutes. In previous sections of this thesis I described, introduced and addressed storage technologies in the order of likely decreasing familiarity and increasing learning curve a geographer would have with each of the formats. In the results section alphabetical order is used.

المنارة للاستشارات
www.manaraa.com

Table 1. Time Required to Run a Model Forecasting Daily Risk of Potato Late Blight over Four Years for a Five Month Season for an Area Covering Michigan
Times are in decimal minutes.

Storage Type	Spindle Drive	Single SSD	Two SSD
MongoDB(v1)	82.14	75.64	74.67
MongoDB(v2)	42.08	42.32	40.09
NetCDF	43.06	31.78	30.38
PostgreSQL	43.68	41.64	41.38

Table 2. Time Required to Derive Daily Inputs and Daily Target Output to be Used to Train a Model Forecasting Risk of Barley Head Blight for a Five Month Season over Four Years for a Tri-State Area
Times are in decimal minutes.

Storage Type	Spindle Drive	Single SSD	Two SSD
MongoDB(v1)	165.75	160.70	159.88
MongoDB(v2)	169.40	174.40	170.46
NetCDF	193.97	154.43	153.87
PostgreSQL	156.98	155.67	153.85

المنارة للاستشارات
www.manaraa.com

Quantitative Results Discussion

MongoDB Version One

The worst performing storage solution in processing the potato late blight model is version one of a system using MongoDB. During systems design I believed that due to fast access of data through the MongoDB server and the fast decompression library used could overcome the loading of more data than was needed per day. Loading the 157 hours stacks of forecasting data while only a 28 hour subset is required to interpolate missing hours within the 24 hour span of data used for the forecasted day did not prove to result in a fast system.

Low performance may also be due to the pymongo driver's requirement to turn MongoDB documents into Python dictionaries internally within the driver when the data is accessed. This is a slow process and is done for large amounts of data. In addition to this, once the data is retrieved and decompressed it takes the form of a text string which is then converted into an array. This also takes time. Both of these steps are not necessary for the systems which use netCDF or PostgreSQL.

Where the system using MongoDB version one suffered from slowdown in the case of the potato late-blight model having to load excess data, in the case of the barley head blight model, because five days of forecasting data is needed, just about all needed data is loaded when a temporal stack of forecasting data is accessed. Despite being the slowest for the potato head blight model this makes version one of the system using MongoDB second fastest when using a traditional spindle drive in

المنارة للاستشارات
www.manaraa.com

running the head blight of barley model.

MongoDB Version Two

For running the potato late blight model my version two of a system using MongoDB is fastest when using a traditional spindle hard drive. Slices of forecasted data were stored as hourly slices instead of as stacked arrays. Nine hours are each loaded to be able to interpolate the missing hours needed within a twenty-four hour period. This avoids loading more data than needed.

The time recorded for version two of a system using MongoDB with a single SSD drive is a little unusual. This benchmark should be a faster than using a traditional spindle drive. It is possible that the pymongo driver connection is overwhelmed with the amount of data throughput. While running the scripts using a spindle drive with version one of the MongoDB system I noticed that after multiple successive accesses through a single python GridFS object slowdown would occur after roughly the fortieth day of a growing season. At this point the buffers attributed to the GridFS object were not clearing before successive accesses. To fix this issue, before running a season, a pool of GridFS objects were created. Successive access would use a random GridFS object from this pool. This gave time for other GridFS objects to clear their buffers. To possibly remedy slowdown when using a single SSD drive a pool of pymongo connection objects may have to be used. Currently the multiple GridFS objects work through a single pymongo connection object and this may introduce slowdown.

المنارة للاستشارات
www.manaraa.com

When running the head blight of barley model the system using MongoDB version two suffered from slowdown because of the large number of single hours of forecasting data that was loaded. For each hour, the process of bringing compressed text strings into python arrays is carried out. This is slower than MongoDB version one because even though version one's data amount of a single set of fives days of forecasting data with a single reference time is larger, per variable, only one conversion to python array of forecasting data is done.

NetCDF System

NetCDF is the second best performing storage solution when using a traditional spindle drive when running the potato late blight model and the best of all cases running the model when using multiple SSD drives. Unlike the MongoDB system the netCDF system does not require a process of converting text to in-memory python arrays.

There is also management overhead for both the MongoDB server and the PostgreSQL server. Data management overhead is more significant in the PostgreSQL server. A lack of database management equates to speed gains for the netCDF system. Unfortunately, the lack of a database management system means that in a system processing multiple models simultaneously, if the netCDF file that contains output data is being written to, other models cannot be writing to that file simultaneously. A solution to this is containing model output for different models in different netCDF files. This may introduce unwanted file-system complexity, as sets

www.manaraa.com

of outputs would multiply as different models are added to a crop disease forecasting system.

Use of SSD drives significantly sped up access in a system using netCDF. National data is stored on disc in these files in a grouped row-column order with pieces of a single temporal coverage consisting of tiles. Using a traditional spindle drive involves drive read head jumps across spatially unneeded data when a regional subset is accessed from national data. Although the time of a single jump is very small multiple jumps add up. The use of SSD eliminates mechanical read heads and non-sequential data is accessed as fast as sequential data. Gains are reduced when a second SSD drive is added as the traditional read head factor had been eliminated on use of the single SSD drive.

A negative of the netCDF based system is data for missing hours in forecasting data is loaded as part of the three dimensional cube subset for the model day and region. This may make the netCDF based system slower but does not impact the system to the point of it being significantly slower in the case of when the potato model is run.

In running the head blight of barley model netCDF again suffers from drive head seek time eliminated with use of an SSD drive. For this model a system using netCDF and a spindle drive is slowest. When a SSD drive is used it becomes second fastest by only one-hundredths of a decimal minute. Again netCDF includes the loading of missing forecasting values in the three dimensional y, x, valid time cube.

المنارة للاستشارات
www.manaraa.com

PostgreSQL System

The PostgreSQL system is comparable to the other two systems. The three perform well when running the model forecasting late blight of potato. PostgreSQL suffers from data management system overhead but does not require the conversion into Python objects the MongoDB system does. This is the oldest most mature technology of the three storage alternatives and the python libraries created to access server data are efficient and backed by fast C libraries with python code calling those C libraries.

PostgreSQL does not significantly benefit from use of SSD drives. This may be due to database management system overhead. There may also be a buffer speed issue as in the MongoDB system.

PostgreSQL is the fastest using a spindle drive for the head blight of barley model and becomes the fastest of all cases despite database management overhead. This is likely because of the storage of single slices prevents the need of retrieving hours with missing data. Missing hour data is loaded in the case of the netCDF system and the MongoDB version one system, but not the MongoDB version two system.

Qualitative Commentary on Expandability

NetCDF was found to be the most cumbersome to use as data variables have to be defined in code or with a command line utility before a model could be added to

المنارة للاستشارات
www.manaraa.com

a system processing multiple models. Despite being the fastest in most cases, this should taken into account when considering netCDF as a potential storage solution.

The MongoDB and PostgreSQL systems all incorporated single repositories for each forecasting, validation, and derived data. For this reason no additional code was needed to store derived regional data. Only region definitions need to be added and the models coded before models can be run.

The MongoDB systems offers additional flexibility the PostgreSQL system does not. A single MongoDB document can potentially store references to multiple derived outputs of a model. This includes derived output sets used as inputs for a neural network model. If a neural network model requires more than a few inputs, storing these inputs as separate references within the same document is quite useful. This was also the case of a model incorporating linear regression of an input variable. The crop disease forecasting system developed using MongoDB stored the by grid cell values of a regression in the same document. This included the R-value, slope, y-intercept and p-value.

The version two system using MongoDB differs from the version one system in that forecasting data is stored in single slices. This may aid in reduced time to implementation and lowered storage space requirements if implemented systems only require storage of a smaller subset of data. For this reason apart from the speed gains of the crop disease forecasting system using MongoDB version two is preferable to version one in terms of expandability.

المنارة للاستشارات
www.manaraa.com

Conclusion

It was expected the inherent capabilities of scientific data files, including internal descriptive attributes and tagging of data, and speed of retrieval of subsets of data, the use of a scientific data file format would likely best meet speed requirements of desired data storage and processing methodologies. As discussed in the rest of this chapter what was expected was not strictly the case. When using default chunksizes, netCDF consistently outperformed the other technologies only when using an SSD drive. It was comparable to the PostgreSQL system when using the large spatiotemporal data chunks involved in the head blight of barley forecasting model. It was thought that in terms of expandability, the NoSQL data store should offer flexibility in the data schema to be used leading to desired expandability. This was true in the test cases for systems using MongoDB as well as PostgreSQL. Although different chunksizes for netCDF files were not tested, with chunksizes potentially impacting speed based on different models, the speed gains seen with netCDF systems while using a traditional hard drive did not offset the expandability reducing requirement of having to declare newly derived variables when using netCDF.

MongoDB version two, netCDF, and PostgreSQL were similar in speed in most cases. Each storage technology had strengths and weaknesses impacting performance. The time needed for array conversions in the case of the MongoDB systems offset the beneficial property of NoSQL database's less database management overhead. Non-optimized chunk sizes impacted the performance of netCDF files when using a mechanical hard drive in the case of forecasting risk due

المنارة للاستشارات
www.manaraa.com

to head blight of barley. PostgreSQL was generally fast overall but still reaches a limit due to database management overhead. Because of strengths and weaknesses of the technologies having performance impacts making system running time differences resulting from storage technology roughly small, if these findings were used as decision support for a choice among tested storage technologies the qualitative factors should significantly impact the decision.

If overall system speed were paramount I would choose netCDF as the preferred storage solution and attempt to further increase that speed by choosing good sizes for on disk dimensional data chunks to further improve access times. If research team personnel resources were small I would choose PostgreSQL as the preferred storage solution as the code needed to use the PostgreSQL library is simple and does not require additional coded tools to view inputs and outputs in ArcGIS Desktop if an ArcSDE license is available. I would use the MongoDB version two if new creative model development were paramount as the MongoDB document paradigm allows for flexibility in stored data contents. During model development data intermediates can be stored together in a related document as those intermediates are created.

المنارة للاستشارات
www.manaraa.com

References

Abdella, Y. and Alfredsen, K. 2010. A GIS toolset for automated processing and analysis of radar precipitation data. Computers and Geosciences. 36(4): 422-429.

Baas, B. 2012. "NoSQL spatial -- Neo4j versus PostGIS". Master's Thesis Utrecht U, Netherlands, 2012. Universiteitsbibliotheek. Web. January 2013. http://igitur-archive.library.uu.nl/student-theses/2012-0822-200532/UUindex.html.

Bachoo, A, Van Den Bergh, F and Gazendam, AD. 2008. Efficient temporal access of satellite image data. PositionIT (November/December 2008), pp 34-39

Baker, K.M. and W.W. Kirk, 2007. Comparative analysis of models integrating synoptic forecast data into potato late blight risk estimate systems. Computers and Electronics in Agriculture 57, 23-32.

Baker, K.M., Lake, T., Roehsner, P. and K. Schrantz. 2012. Forecasting Disease with 10-Year Optimized Models: Moving Toward New Digital Datasets. The First International Conference on Agro-geoinformatics 2-4 Aug, Shanghai, China.

Bayer, R. and McCreight, E. 1972. Organization and Maintenance of Large Ordered Indexes. Acta Informatica. 1(3): 173-189.

Bondalapati, K.D., Stein, J.M., Osborne, L.E., Neate, S.M., Hollingsworth, C.R., 2012. Development of weather-based predictive models for Fusarium head blight and deoxynivalenol accumulation for spring malting barley. Plant Dis. 96, 673-680.

Brown, P. G., et. al. (The SciDB Development Team). 2010. Overview of SciDB. In Proceedings of the ACM SIGMOD International Conference on Management of data. 2010. 963-968.

Chamberlin, D. D. and Boyce, R. F. 1974. SEQUEL: A Structured English Query Language. Proceedings of the 1974 ACM SIGFIDET Workshop on Data description, Access and Control, May 1974.

Chang, F., Dean, J., Ghemawat, S., Hsieh, W.C., Wallach, D.A., Burrows, M., Chandra, T., Fikes, A., and Gruber, R.E. (2008). Bigtable: A Distributed Storage System for Structured Data. *ACM Transactions on Computer Systems* 26, 2, Article 4 (June 2008), 26 pages. DOI=10.1145/1365815.1365816 http://doi.acm.org/10.1145/1365815.1365816

المنارة للاستشارات
www.manaraa.com

Chen, P. 1976. The Entity-Relationship Model - Toward a Unified View of Data. ACM Transactions on Database Systems. 1(1): 9-36.

Chesnaux, R., Lambert, M., Walter, J., Fillastre, U., Hay, M., Rouleau, A., Daigneault, R., Moisan, A., Germaneau, D. 2011. Building a geodatabase for mapping hydrogeological features and 3D modeling of groundwater systems: Application to the Saguenay-Lac-St.-Jea region, Canada. Computers & Geosciences. 37(11): 1870-1882.

Codd. E.F. 1970. A Relational Model of Data for Large Shared Data Banks. Communications of the ACM. 13(6): 377-387.

Codd, E.F. 1974. Recent Investigations in Relational Database Systems. In IFIP Conference. Stockholm, Sweeden. 1017-1021.

Cohen, S., Hurley, P., Schulz, K.W., Barth, W.L., and Benton, B. 2006. Scientific Formats for Object-relational Database Systems: A Study of Suitability and Performance. SIGMOD Record 35(2): 10-15.

Goodall, J.L. and Maidment, D.R. 2009. A Spatiotemporal Data Model for River Basin-scale Hydrologic Systems. International Journal of Geographical Informational Science 23(2): 233-247.

Goodchild, M.F. 1992. Geographic information science, International Journal Geographical Information Systems, 6(1), 31–45.

Goodchild, M.F. 2005. GIS and Modeling Overview. GIS, Spatial Analysis, and Modeling. Maguire, D.J., Batty, M., Goodchild, M.F., Eds. ESRI Press. Redlands, CA. 2005

Guenthner, J.F., Michael, K.C., and Nolte, P. 2001. The Economic Impact of Potato Late Blight on US Growers. Potato Research 44(2): 121-125.

Jonas, bFlood, JoshFinnie [Online forum usernames], and others. 2010. Is there any way i can use a key-value store for geospatial data? [Online forum comment]. Retrieved from http://gis.stackexchange.com/questions/59/is-there-any-way-i-can-use-a-key-value-store-for-geospatial-data/553

Lakshman, A. and Malik, P. (2010). Cassandra: a decentralized structured storage system. *SIGOPS Operating Systems. Review.* 44, 2 (April 2010), 35-40. http://doi.acm.org/10.1145/1773912.1773922

Maidment, D.R. 2004. Arc Hydro Data Model: A Hydrologic Data Integration Tool. Southwest Hydrology. 3(3): 18-19.

المنارة للاستشارات
www.manaraa.com

Miler, M., Medak, D., and Odobašić, D. (2011) Architecture for Web Mapping with NoSQL Database CouchDB. *Geospatial Crossroads @ GI_Forum '11: Proceedings of the Geoinformatics Forum Salzburg,* Car, A., Griesebner, G. and Strobl, J. (eds) Geospatial Crossroads @ GI_Forum '11: Proceedings of the Geoinformatics Forum Salzburg, Wichmann, p. 62–71.

Myrick, D.T. and Horel, J.D. 2006. Verification of Surface Temperature Forecasts from the National Digital Forecast Databes over the Western United States. Weather and Forecasting 21(5): 869-892.

Nganje, W.E., Bangsund, D.S., Leistritz, F.L., Wilson, W.W., and Tiapo, N.M. 2002. Eds. Canty, S.M., Lewis, J., Siler, L., and Ward, W. 2002 National Fusarium Head Blight Forum Proceedings. US Wheat and Barley Scab Initiative. http://scab.pw.usda.gov/pdfs/forum_02_proc.pdf#page=288 (accessed December 5th 2011). 275-281.

Pavan, W., Fraisse, C.W., and Peres, N.A. 2011. Development of a Web-based Disease Forecasting System for Strawberries. Computers and Electronics in Agriculture 75(1): 169-175.

Ruth, D.P., Glahn, B., Dagostaro, V. and Gilbert, K. 2009. The Performance of MOS in the Digital Age. Weather and Forecasting 24(2): 504-519.

Stainer, D. Introduction to NoSQL Databases. 2010. Presentation. San Diego NoSQL Meetup. Aug 2010. http://www.slideshare.net/dstainer/introduction-to-nosql-databases

Stonebraker, M., Bear, C., Cetintemel, U., Cherniack, M., Ge, T., Hachem, N., Harizopoulos, S., Lifter, J., Rogers, J., and Zdonik, S. 2007. One size fits all? Part 2: Benchmarking studies. In Proc. of CIDR Conference. Jan 2007. 173-184.

Stonebraker, M. and Cetintemel, U. 2005. One Size Fits All: An Idea Whose Time has Come and Gone. In Proc. of the 21th International Conference on Data Engineering. Apr 2005. 2-11.

Strassberg, G., Jones, N.L., Maidment, D.R. 2011. Arc Hydro Groundwater: GIS for Hydrogeology. ESRI Press. Redlands, CA. 2011.

المنارة للاستشارات
www.manaraa.com

Tao, Z., Malvick, D., Claybrooke, R., Floyd, C., Bernacchi, C.J., Spoden, G., Kurle, J., Gay, D., Browersox, V., and Krupa, S. 2009. Predicting the Risk of Soybean Rust in Minnesota Based on an Integrated Atmospheric Model. International Journal of Biometeorology 53(6): 509-521.

Weigel, R.S., Lindholm, D.M., Wislon, A., and Faden, J. 2010. TSDS: High-Performance Merge, Subset, and Filter Software for Time Series-like Data. Earth Science Infomatics 3(1-2): 29-40.

Williams-Woodward, J. 2010. 2009 Georgia Plant Disease Loss Estimates. The University of Georgia Cooperative Extension. http://www.caes.uga.edu/applications/publications/files/pdf/AP%20102-2_1.PDF (accessed December 5th 2011).

Xiao, Z. and Liu, Y. (2011). Remote sensing image database based on NOSQL database. *Geoinformatics, 2011 19th International Conference on*, pp.1-5, 24-26 June 2011.

Zender, C.S. 2008. Analysis of Self-describing Gridded Geoscience Data with netCDF Operators(NCO). Environmental modeling and Software 23(10-11): 1338-1342.

Zhao , J., Wang, Y, and Zhang, H. 2011. "Automated batch processing of mass remote sensing and geospatial data to meet the needs of end users", *Proc. IGARSS*, pp.3464 -3467.

المنارة للاستشارات
www.manaraa.com

Appendix A

Tables in a PostgreSQL Database for Gridded Crop Disease Forecasting

www.manaraa.com

Tables in a PostgreSQL Database for Gridded Crop Disease Forecasting

Datasets (metadata about datasets)

datasets		
	recid	serial unique
	dataset	text
	meta_dsc	text containing json string

Variables (metadata about variables)

variables		
	recid	serial unique
	dataset	text
	var	text
	meta_dsc	text containing json string

Cells of National Grid

national_grid		
	recid	serial unique
	cellid	integer
	dataset	text
	x	integer
	y	integer
	lon	float
	lat	float
	cell_shape	

Validation Data (National)

validation_data		
	recid	serial unique
	dataset	text
	var	text
	vldtime	timestamp without timezone
	float_array	2 dimensional array of double precision floats
	text_array	2 dimensional array of text

Forecasted Data (National)

forecasted_data		
	recid	serial unique
	dataset	text
	var	text
	reftime	timestamp without timezone
	vldtime	timestamp without timezone
	float_array	2 dimensional array of double precision floats
	text_array	2 dimensional array of text

Cells of Regional Grid

rational_grid		
	recid	serial unique
	parent_cellid	integer
	region	text
	dataset	text
	x	integer
	y	integer
	lon	float
	lat	float

Models

models		
	recid	serial unique
	model	text
	crops	array of text
	crop_disease	text
	meta_dsc	text containing json string

Model Parameters

model_params		
	recid	serial unique
	model	text
	output_parameter	text
	meta_dsc	text containing json string

Regions

regions		
	recid	serial unique
	region	text
	upperlat	double precision float
	lowerlat	double precision float
	upperlon	double precision float
	lowerlon	double precision float
	meta_dsc	text containing json string

Regional Crop Mask

crop_masks		
	recid	serial unique
	region	text
	crop	text
	regional_crop_mask	2 dimensional array of boolean

Derived_Params

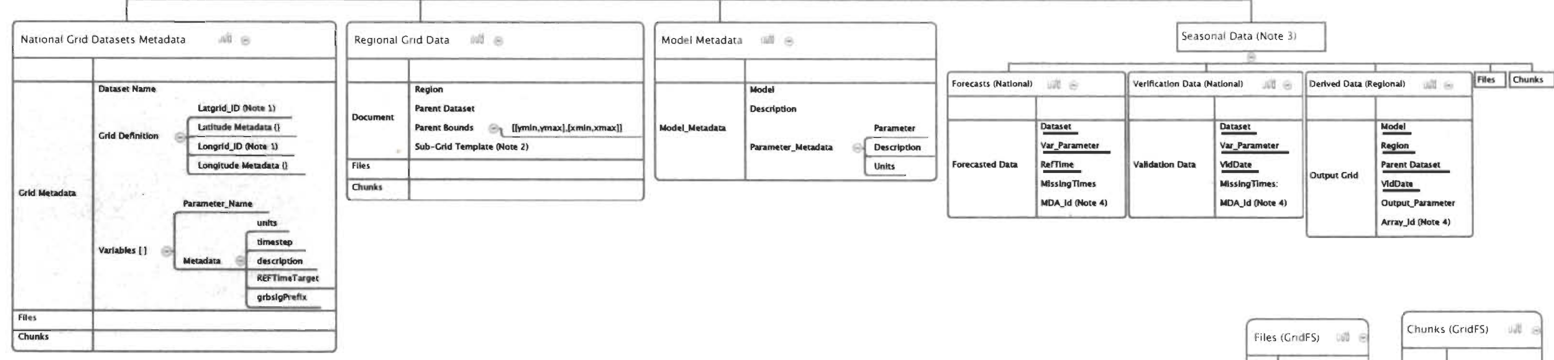
derived_param_data		
	recid	serial unique
	region	text
	crop	text
	model	text
	parent_dataset	text
	parameter	text
	vlddate	date
	float_array	2 dimensional array of double precision floats
	text_array	2 dimensional array of text

المنارة للاستشارات
www.manaraa.com

Appendix B

Document Hierarchy in a MongoDB Datastore for Gridded Crop Disease Forecasting

www.manaraa.com



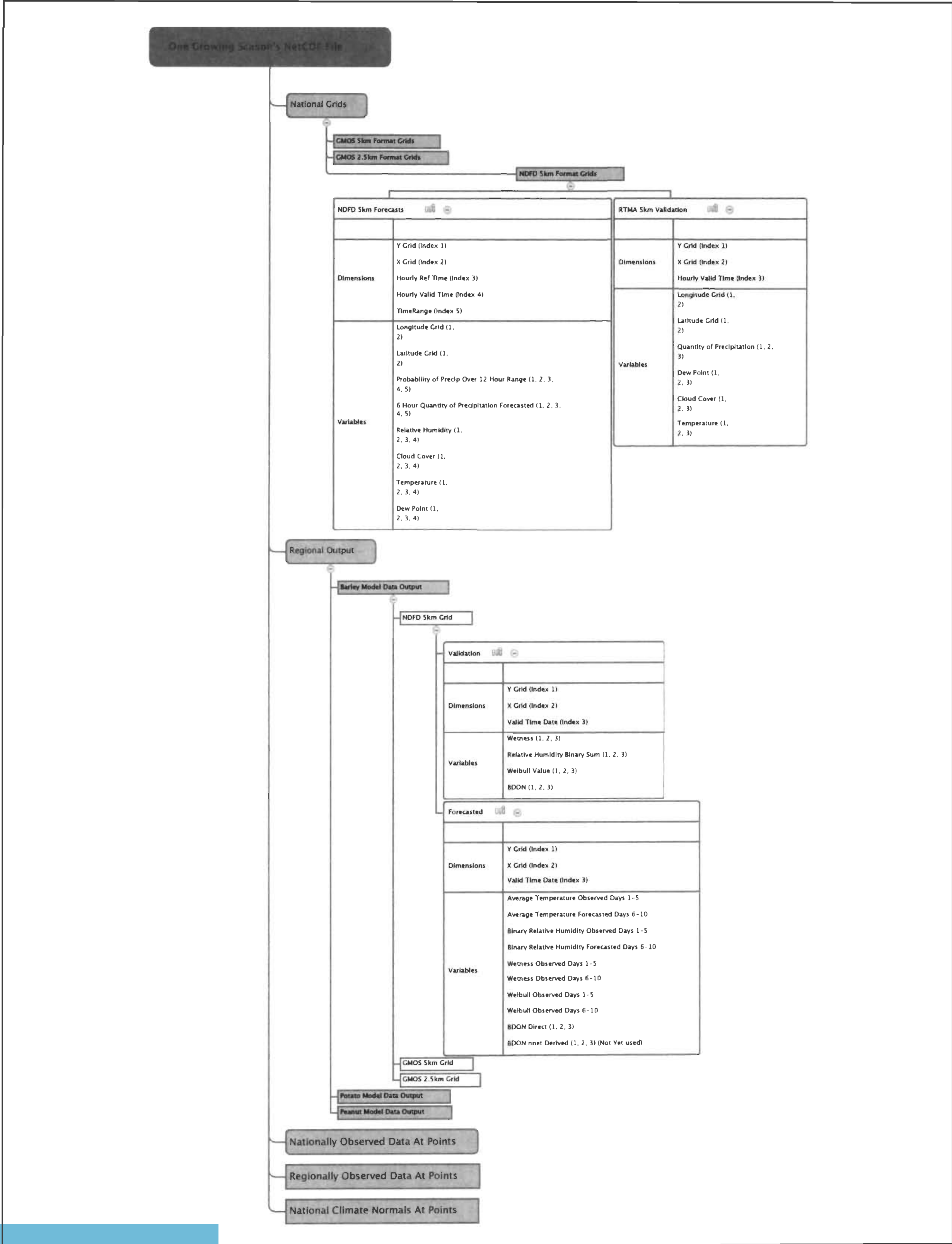
المنارة للاستشارات
www.manaraa.com

Appendix C

One Growing Season's NetCDF File

المنارة للاستشارات

www.manaraa.com



www.manaraa.com