

# GEO-SERVICE PORTAL

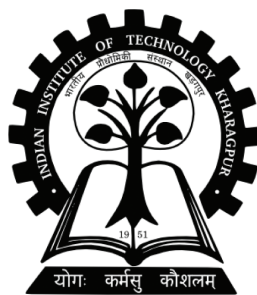
Crawling, Meta-data Discovery, Publishing & Query Orchestration  
for Spatial Data

*Thesis submitted to Indian Institute of Technology Kharagpur  
in partial fulfillment of the requirements for the award of the  
degree of*

Master of Technology  
*in*  
Computer Science and Engineering  
*by*

Deepak Punjabi  
15IT60R17

Under the guidance of  
Prof. Soumya K. Ghosh



DEPARTMENT OF COMPUTER SCIENCE AND ENGINEERING  
INDIAN INSTITUTE OF TECHNOLOGY KHARAGPUR  
NOVEMBER 2016

©2016 Deepak Punjabi. All rights reserved.

## **CERTIFICATE**

This is to certify that the thesis titled **Geo-Service Portal**, submitted by **Deepak Punjabi**, in the Department of Computer Science and Engineering, Indian Institute of Technology, Kharagpur, for the award of the degree of Master of Technology, is a record of research work carried out by him under my supervision and guidance.

The thesis fulfills all the requirements as per the regulations of this institute. Neither this thesis nor any part of it has been submitted for any degree or academic award elsewhere.

**Prof. Soumya K. Ghosh**  
**Dept of Computer Science & Engineering**

# Abstract

In the recent era of technology data is everything. One of the very useful type of data is spatial data. The information about spatial data can be found to be much useful in many fields of industry. From remote sensing, mining to doing spatio-temporal analysis to doing flood or disease spread analysis, the applications are endless. However this data is not generally publicly easy to find. Spatial data is not handled well by the the general purpose search engines. The methods to access these data are also heterogeneous and many times require licensing and using proprietary technologies. Heterogeneity of the data poses another problem of ranking and combining A central catalog service can be built to keep track of this various repositories, also the data can be made available through simple unified calling mechanisms. With the availability of large amount of heterogeneous data from different multiple sources one can do many type of spatio temporal analysis. This catalog service can then behave as central node for all available geo-spatial data available in the web. Registry can be used to publish all the information to subscribers and open web. Using this registry an efficient query processing system can be built around it to generate information needed in the user friendly mode. Query processor can find data from various heterogeneous data stores and process the data to get one result from many data sources. Various ranking and cost matrices can be deployed to make indexing of returned data feasible.

# Contents

<b>1</b>	<b>Introduction</b>	<b>1</b>
1.1	Motivation . . . . .	1
1.2	Problem Statement . . . . .	2
1.3	Objectives . . . . .	2
1.4	Organization of thesis . . . . .	3
<b>2</b>	<b>Literature Survey</b>	<b>4</b>
2.1	Spatial Data . . . . .	4
2.1.1	Classification of Spatial Data . . . . .	4
2.1.2	OGC Web Services . . . . .	5
2.2	Spatial Web Crawler . . . . .	7
2.3	Catalog Service for Geo-Spatial Data . . . . .	8
2.4	Query Orchestration . . . . .	8
<b>3</b>	<b>Spatial Web Crawler</b>	<b>10</b>
3.1	Objectives . . . . .	10
3.2	Architecture of the Spatial Web Crawler . . . . .	10
3.3	Advantages of Spatial Web Crawler . . . . .	13

3.4	Example Scenario . . . . .	13
<b>4</b>	<b>Catalog Service and Query Processing</b>	<b>15</b>
4.1	Architecture . . . . .	15
4.2	Implementation . . . . .	17
4.2.1	Crawler Module . . . . .	17
4.2.2	Database Setup . . . . .	17
4.2.3	Catalog Service . . . . .	18
4.2.4	Query Processing . . . . .	19
4.3	Results . . . . .	19
<b>5</b>	<b>Conclusion and Future Work</b>	<b>23</b>
5.1	Extension . . . . .	23

# List of Figures

3.1	Spatial web crawler architecture . . . . .	11
3.2	XML response from the geo server . . . . .	11
4.1	Architecture of Catalog Service . . . . .	16
4.2	List of relations . . . . .	18
4.3	Map of KGP BNK ROAD . . . . .	20
4.4	Map of KGP BNK ROAD . . . . .	21

# Chapter 1

## Introduction

### 1.1 Motivation

Currently available search engines are good for normal data such as text and web links, but when it comes to searching and ranking spatial data, none of the currently available search engines can provide efficient results. One of the reason for this can be that not all geo-spatial data is publicly available. Other problems is that repositories containing spatial data are not generally indexed and the metadata information about what data they contain is not easily available. Our motivation here is to build a registry service for the nation that can avail data from national government agencies as well as open source free repositories. Once the data is available various kinds of tasks can be built on top of that. One can easily perform various type of spatial queries like GetMap on the data available. Various applications can be provided the available data to cater in use in various ways. A query interface can be built upon the data available from catalog service. This query interfaces can provide various views for user specific user queries. An spatio-temporal analytics can also be done using the catalog. Change of data can be tracked. Other use cases can be finding the spread of diseases or getting the area affected by flood. Many other applications can be generated.

## 1.2 Problem Statement

The aim of this project is to build a catalog service for web to crawl, store and maintain, and publish metadata information about spatial data and it's providers and to utilize this information to perform efficient and parallel query orchestration.

## 1.3 Objectives

1. **Build a topical crawler to crawl the web and store geo-spatial metadata.**

Here the topical crawler crawls the open web, finds and filters url's found by it. This module checks whether the server provided by the url is capable of providing OGC compliant geo-spatial web services or not. If the url is found to be a geo-server it stores all the metadata information about the geo-server and about the data it contains into the database. This information then later can be utilized to query and find useful data.

2. **Build an OGC compliant catalog service to publish and search accumulated metadata.**

The aim here is to build a web service that can query and publish relevant metadata information from the database. The catalog service is real-time. Meaning the information available in the catalog changes as the time goes. This can also be explored to examine spatio-temporal nature of the data. The catalog service should be built in such a way that queries can be processed in real time. For this purpose the data should be stored in database in structured manner rather than storing it in plain json or xml files. One other crucial point for catalog service is to provide high availability. Registry is used to search as well as publish the available data.

3. **Build Query orchestration service to perform real-time query with heterogeneous data sources and cost matrices associated with them.**

One of the tasks of catalog service is to provide relevant information to the querying machine or person. The data that is stored by different repositories can be of heterogeneous form. Task of query orchestration is



to find relevant information across all available data sources and perform the query using all relevant data. If the data is available from multiple data sources different type of ranking and cost matrices can be associated to provide most relevant information to the user.

## **1.4 Organization of thesis**

This thesis is mainly organized into five chapters. The first chapter gives introduction about the subject matter, it clears the motivation and advantages of the project. It defines the problem statement and states as well as explains the objectives of the project. The second chapter namely literature survey, provides already available data about the subject and proposed solution methods. This section first explains about the data for the model to be work with. Then it explains about three core part of the project, Spatial web crawler, Catalog service and query orchestration. The next chapter is based on the spatial web crawler. It explains the architecture of the spatial web crawler and explains it's advantages. Next chapter contains implementation details about catalog service for web and query processing. It also shows example results. The final chapter concludes the morals of the thesis and creates a pathway for better optimizing and extending the proposed solution for the future work.

## Chapter 2

# Literature Survey

The literature for this project can be divided mainly into four parts. The data on which the framework is developed. How to find that data. Once the data is found, how to store that data. How to make data easily accessible to others. And at the last how to perform simple queries on the available data.

### 2.1 Spatial Data

The term Geo comes from geography. Geography stores all the information of location and shape of the object in the spatial data. Spatial data stores the relationships between these data. It can also be easily mapped to a map. Geo-server provides various kinds of functionality to this type of data. Spatial data is the data that can be mapped. Spatial data is collection of spatial object that has topology and co ordinates. Spatial data gives geographical and shape related attributes of the data. Geographical information system is used to retrieve and operate on spatial data. Different operations can be add, visualize, annotate etc. Different examples of soft- ware that offers spatial services are ESRI, Microsoft SQL Server, ERDAS imagine etc.

#### 2.1.1 Classification of Spatial Data

Different types of object under the class geometry are as below. All the major Geo-spatial service providers and vendors provide this kind classification. The primitive four data types in spatial data are point, curve, surface and Geom-

Collection.

- **Point**

Point in a map is denoted by (x, y) co-ordinates. When we see kharagpur city on a scale of India it will be seen as a point. Point can be used to denote various objects like origin, city, end point, top of the mountain etc. It denotes a single point consisting of longitude and latitude.

- **Curve**

Curve is used to denote collection of points. This can be a straight line or curve. For example, a road network can be represented with the help of line strings. Similarly, a river can be denoted as a curve. Curve can be made with help of two distinct points.

- **Surface**

A surface is representation of an area or a polygon. When kharagpur is seen in the scale of west Bengal it is seen as surface or polygon. Polygon can be made using at least three non-linear points. Every polygon has a feature called boundary.

- **GeomCollection**

Collection of basic building blocks defining a new type of geometry can be defined with the help of GeomCollection. GeomCollection is made with combining two or more geometry types.

## 2.1.2 OGC Web Services

Open Geospatial Consortium (OGC) is the worldwide standardization body for geospatial standards. OGC provides a standardized way of accessing this geospatial data. OGC provides three kinds of web services.

- **Web Map Service (WMS)** Web Map service defines a way of accessing geospatial information across all geo servers in a standard format as image. This image can be raster image or a vector image. Raster images are of type jpg, png or bmp. Vector images contains svg format extension images. It also provides a way to access metadata about the available information of the layers. This information can contain type and no of layers.

Some of the well-defined operations in this layer are GetCapabilities, GetMap and DescribeLayer. GetCapabilities operation returns capabilities of the server ie what kind of services the server offers. This operation may be useful to check if a server is geo server or not by examining the kind of services it offers. GetMap service returns map images for the queried data. multiple such layer of images can be queried and then can be overlayed on top of each other. for example satellite view, terrain view, traffic view etc. Each of this layer gives additional information to the map. DescribeLayer operation describes what are the available layers and what is the metadata about the layers.

- **Web Feature Service (WFS)** WFS allows direct access to features contained in the map. WFS uses SOAP based interface. SOAP is used to minimize the overhead into the communication and verify cross platform stability. For exchanging data between client and server WFS uses Geographical mark-up language(GML) which is based on XML. Some well-defined operations in WFS are query or get feature, which returns the feature stored on the server. We can add the feature in the repository by add feature. We can delete feature by delete feature. Also we can update feature stored in the repository by update feature command. We can also lock certain feature to disallow modification of it while sharing via lock feature. Locked resources and features wont be accessible to change to other clients. Example of feature are the objects in the map like building or petrol pump.
- **Web Coverage Service (WCS)** WCS offers multi-dimensional coverage of the geo spatial data.It can provide originally retrieved data or provide processed data. It provides spatio-temporal context to the given geographical data. For example, it can show the flow of the river changing over the span of years. Thus we can say that WCS provides richer coverage of spa-

tial data than WFS or WMS. Coverage data is mainly used for scientific calculations.

## 2.2 Spatial Web Crawler

Wenwen Li says that if a server is offering geo-spatial capabilities then it can be either of the WMS, WFS or WCS server. The geo-servers must comply to OGC standards for geospatial data. This can be checked using GetCapabilities service request. We can check whether a given server offers WMS offerings or not can be checked using special suffix string added to the url of a geo-server. More details are covered in the geo-spatial crawler part below. If the server is indeed a WMS server, It replies with an XML file containing all the data and operations it can offer. We call this as services.xml file. Once We know that the server is a geoserver many kinds of operations can be performed on it. WMS geoservers offer various operations such as getmap, describelayer, getcapabilities etc. WMS capabilities file contains all these operations mentioned above. This is to check if a particular server is geoserver or not. If we want to crawl through the open web, then we must repeat this process. For this purpose standard operation of crawlers are performed. We maintain a buffer of frontiers which divides the boundaries of crawled web and open un-crawled web. Wenwen Li also suggests to make a auto updating crawler to periodically check whether there is any addition or removal of geoserver from the existing available data. This geoservers also add new data to them and remove obsolete data from the data store. Automatic updates allows this operations to be done periodically and maintain the overall consistency of the data.

Marc Najork suggests that the task of crawling the web for WMS servers can be done parallel. He suggests to use parallel FIFO queues to carry out the operation efficiently. To check whether the URL already exists or not in the crawled set, he suggests to use efficient data structures to check set membership such as HashSet. He also suggests to use priority in crawling based on various methods such as PageRank.

Lopez-Pellicer suggests that there might be already available geoservers and catalog services on the web. We must efficiently identify this catalog services to different type of OGC compliant services. So that different kind of queries can be forwarded to different geo servers or catalog services. Dirk Ahlers suggests an

architecture to crawl and parse available geoservers and store the available data efficiently. They also suggests to index the data so there data can be retrieved efficiently. This methods is described as focused crawling.

Li, W., et al. suggest a method to semantically crawls the web. In his approach he maintains a hierarchy of geospatial objects to identify parent children relationship. For example, river is a type of waterbody. This kind of type of relationships can be used to manage and index available data and geoservers efficiently. JIANG Jun suggests similar methods for WFS servers.

## 2.3 Catalog Service for Geo-Spatial Data

Python catalog service for web offers a model architecture for implementing a OGC compliant web service catalog portal. In this model, series of module are used to invoke a hierarchy such that decomposition of the functions and use case is efficient and easy. In this architecture many of the modules can work in parallel. Important modules are crawler, WMS capabilities files, Database to store geospatial data and a server to provide interface to functions. All the functions are implemented and run via programs on a server. This wsgi server acts as a geospatial catalog which accumulates data from various other geospatial repositories and catalogs available on the open web. Using the pycsw interface many types of applications can be invoked. For example, using GetMap a user or application can get all the data from all the crawled repositories in a single place. We can know the information available for the layers and from where this information is available. The detailed implementation of this architecture is discussed in the later chapters.

## 2.4 Query Orchestration

Once the catalog service is built it can directly act as a single point for querying multiple repositories. The catalog server in-fact acts as a mediator while getting the queried data from other repositories. Different type of example queries are listed below.

- **Service metadat information**

Describes what kind of OGC compliant web services are offered by the catalog server or geo-server.

- **GetLayers**

An interface can be built to provide information about list of available layers.

- **GetOperations**

List of available operations can be provided that gives the information about the list of operations that are offered by the geoserver or catalog service.

- **GetBoundingBox**

We can get the information about the bounding box of the layers or map that covers the total area.

- **GetMap**

The map of particular layer can be retrieved.

- **DrawMap**

The map of particular layer/data can be retrieved and placed onto another map. This is useful to find area of interest in the particular map by superimposing the queried data to generic map.

# Chapter 3

## Spatial Web Crawler

### 3.1 Objectives

1. Build a spatial web crawler which crawlers through geo-servers which offers WFS based OGC compliant services.
2. Build a domain specific vocabulary(ontology) for this features which can be helpful to compare found features with wanted features.
3. Perform semantic matching of found features from crawled web-pages with given ontology for filtering the correct features and storing them in the permanent repository.
4. Perform an evaluation of the given spatial web crawler using metrics and test URL seed sets.

### 3.2 Architecture of the Spatial Web Crawler

Our crawler contains three modules for crawling spatial features through the world wide web. Some of the definition needed for understanding the working of spatial web crawler are:

- **Seed set:** a set of test URLs to initialize the queue for crawling the web. The crawler starts with the URLs given in the seed URL set.



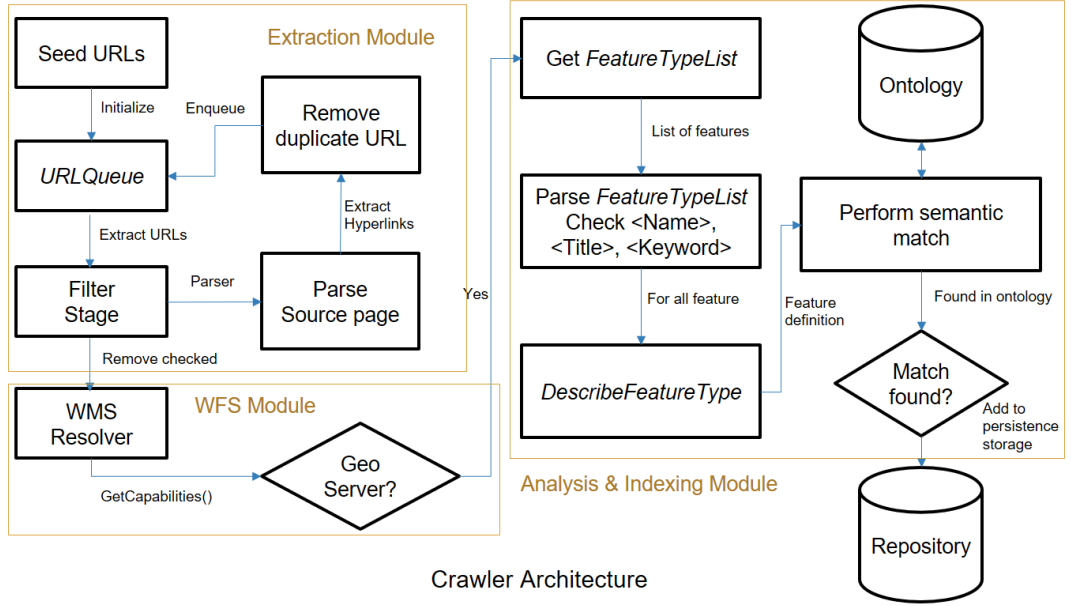


Figure 3.1: Spatial web crawler architecture

- **URLQueue:** A queue that contains set of URLs to be crawled. The crawler module takes URLs one after the another from this queue.
- **Frontiers:** a set of URLs from the URLQueue that are currently being crawled. This are called frontiers because they reside between the known web and the unknown web.
- **WMS resolver:** WMS resolver is a module that checks that if a server is WMS server or not given the URL from the URLQueue.
- **Parser:** parser is a module that downloads the webpage from the given URL and parse the HTML webpage looking for pre-specified tags and

```

<?xml version="1.0"?>
- <root>
- <FeatureType>
  <Name>prov_land</Name>
  <Title>Canadian Land</Title>
  <SRS>EPSG:42304</SRS>
  <LatLongBoundingBox maxy="83.8009" maxx="-11.9603" miny="35.8775" minx="-173.537"/>
</FeatureType>
-
- <FeatureType>
  <Name>land_fn</Name>
  <Title>US Land</Title>
  <SRS>EPSG:42304</SRS>
  <LatLongBoundingBox maxy="89.8254" maxx="179.94" miny="31.8844" minx="-178.838"/>
</FeatureType>
</root>

```

Figure 3.2: XML response from the geo server

names. After parsing it gets a set of tags and its contents. In our implementation we parse the webpage and look for the anchor tags within it and store all the hyperlinks from the crawled webpage.

- **Ontology:** semantic dictionary containing all the features its type and relationship hierarchy between features

Our crawler contains mainly three modules:

- **Extraction module**

Our algorithm starts with a set of seed URLs contained in the seed set. We initialize the URLQueue with these seed URLs. The filter stage takes URL one after the other and checks whether the given URL is already crawled or not. After filtering such URLs, they are sent to the parser. The parser downloads the webpage from the given url and parses it for finding hyperlinks contained in it. These hyperlinks again go to filter stage for finding whether the given urls are already crawled or not. After filtering these urls are added to the end of URLQueue. The filtered urls are also passed to the WFS module.

- **WFS module**

Once we have a URL for the examination, we send a GetCapabilities() request to that server. We do this by appending the request to the url. *services?REQUEST = GetCapabilities&version = 1.1.0&service = WFS* The server replies for this request. If the reply contains WFS\_Capabilities tag, then it offers WFS service. We parse the received response for finding the WFS\_Capabilities tag. If the given server is WFS server then the given url is passed to analysis and indexing module.

- **Analysis and Extraction module**

In this stage server response is parsed for the tag FeatureTypeList. FeatureTypeList contains list of features. This features are stored under the tag FeatureType. FeatureType tag contains set of keyword, title, name tags. Each of this tags are checked to see if it contains any word from the ontology. For each of such tag found, DescribeFeatureType request is appended to the url.

*?service = WFS&version = 1.1.0&request =  
DescribeFeatureType&typename = " + keyword*

Here the keyword is the name of the feature. Each of this retrieved feature is checked again the ontology, if the feature is found in the ontology then it is added to permanent storage in the repository.

### 3.3 Advantages of Spatial Web Crawler

- Allows search in web pages that are not generally searchable from the normal search engines. This is because of the spatial context awareness of the spatial web crawler.
- It provides more up-to-date results from the results. Spatial crawler is more sensitive to changes of spatial data on the web and automatically crawls through the changed features with the help of automatic update module.
- Provides improved accuracy in the search of spatial features and operations.
- Provides extra features such as bounding box of spatially crawled data and other spatial features. This bounding box can then be used for visualizing the crawled data.
- Spatial data mining and analysis kind of tasks can be performed. So that we can infer and predict new results and patterns.

### 3.4 Example Scenario

Suppose we search for a example query after building the system. Let this query be *river and snow*. Now if we search this on normal search engines like google, it will show us results containing either river or snow. But as we have information about spatial context in this crawler. Repositories have stored the information about which features are in near proximity to each other.

Other than this, we have a object ontology. Which is a hierarchical graph based on the spatial features. This graph contains generalized and spatialized features.

For example it suggests that river comes under the type stream which intern comes under the geometry type waterbody. This type of hierachical classification helps us to understand the geospatial data and their features and relationships correctly. Our query is river and snow. The crawler searches the repository for finding both features river and snow or their similar representation by doing a semantic matching over ontology. Once found it can see which is the geo server which offers both of the geo services and feature types. Based on its finding from the repository we conclude our results and send the reply back to the user.

# Chapter 4

## Catalog Service and Query Processing

Once the geo-servers are crawled from the open web, we need to organize this data into a well organized manner so that it can be retrieved efficiently. The operation and query we perform on this data must be performed in a efficient manner.

### 4.1 Architecture

- There are various data repositories available on the web. However this repositories are not indexed. Spatial web crawler crawls through open web and stores corresponding metadata information about available data into the xml files. Crawler uses *Getcapabilities()* OGC request to find if a given server is geoserver or not, if the given URL is indeed a geo-server then crawler module requests to retrieve *capabilities.xml* file from the server and stores it into warehouse. XML files contains the services information provided by the geo-servers found on the web. They are usually called services.xml. This information contains available operations, available layers and other useful information. This xml files are gathered in a folder containing xml files.
- No of spatial repositories in the web is significantly lower than other type of services and servers. It can happen that it might take a significant amount of time to gather practically good number of geo-servers. For this reason, this files can be added by the spatial web crawler or it can be

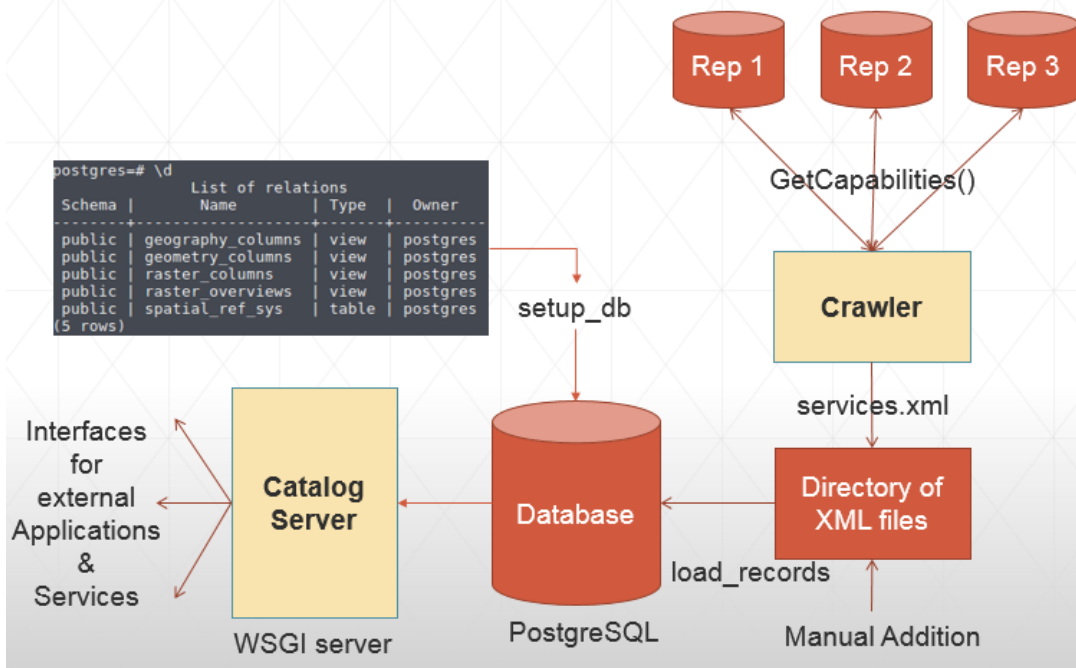


Figure 4.1: Architecture of Catalog Service

added manually.

- This xml files are then loaded into the database for faster and efficient retrieval operation. For database we can use either of MySQL, PostgreSQL or SQLite. But MySQL and SQLite does not offer inherent data structures and operations support for spatial data. SQLite is also meant for light weight databases, whereas here the no of request-reply and data transfer can be very high. For these reasons it is better to choose PostgreSQL database system to store data for the catalog service. PostgreSQL offers high scalability and offers inherent support of spatial data types and operations.
- Catalog Service is hosted on *Web server gateway interface (wsgi)* server. We can also use other deployment servers like apache. Main reason for choice of server as wsgi is that, wsgi integrates well with python. It is build on python. It is useful for request-reply type ordinary web interfaces and it is efficient for file transferring on other protocols. A python program admin.py is hosted on wsgi server, which takes the metadata information from the database and replies in response to GetCapabilities() request. When working on a particular request for a repository like GetMap(), the catalog service acts as a mediator between client and the repository on which the data is hosted. It converts the request taken by wsgi server

converts it into standard OGC compliant WMS or WFS call, and accesses data from original repository from where data is hosted.

## 4.2 Implementation

### 4.2.1 Crawler Module

- In first stage implementation, we first crawl through open web using the spatial web crawler discussed in the above chapter. The crawler checks if the given server provides geospatial services or not. If the server is a geospatial repository then it collects all the metadata information about the repository and stores it in a form of a xml files. Standard OGC compliant service calls are used for collecting metadata information about found registry. Here in the figure we can see there are three repositories which have been crawled by the crawler. The information about these repositories are stored in a xml files called services.xml. The no of geoservers and registries are very less in the open web compared to other kind of services, for this reason it might take a long time to crawl and find significant amount of registry services for our catalog service. To handle this problem, manual addition option is used. We can manually add popular geoservers and registry services to the services.xml files. We know that geoservers reply in xml response when called for a getcapabilities request. We can use this property and manually add popular geo-registries beforehand to the set of xml files. So in the summary, the first stage crawls the web and stores corresponding set of services files into one place.
- For this python based crawler module program is used. It takes a set of seed URLs, checks if the URL can build object of OGC compliant WMS or WFS object. If it can build OGC WMS object then, crawler requests for capabilities file, and stores it in set of xml files.

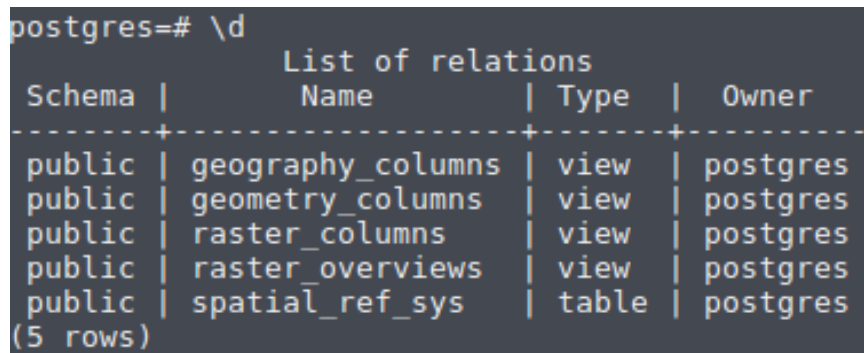
### 4.2.2 Database Setup

- In second step, These set of xml files are used to populate database. Here various type of databases can be used. I have tried databases like MySQL, SQLite and PostGreSql. In our approach I have used postgresql because

it offers geospatial properties and services inherently. Other databases do not offer inbuilt services for spatial data. Postgresql database offers postgis extension, which are a set of libraries to extend postgresql database into spatial database. For adding postgis to postgresql, go to postgresql shell `psql` and create a extension, `postgis`.

*CREATE EXTENSION POSTGIS;*

- Once this is done, import all of the `services.xml` files into the database. Create structured tables and schema for maintaining spatial data. For this purpose `setup_db` command is used from catalog service module.



```
postgres=# \d
```

List of relations			
Schema	Name	Type	Owner
public	geography_columns	view	postgres
public	geometry_columns	view	postgres
public	raster_columns	view	postgres
public	raster_overviews	view	postgres
public	spatial_ref_sys	table	postgres

(5 rows)

Figure 4.2: List of relations

- Here four tables are built to store and index spatial data efficiently. Those are, `geometry_columns`, `records`, `spatial_ref_sys` and `spatial_ref_sys.srid_seq`. This tables are then used by the catalog server to publish and query available data. When a external entity queries catalog server for Get Capabilities request, the catalog server replies with the services file which contains all the information about all the geo-servers it has crawled till now. Thus it contains all the layer information and data crawled from the internet. This makes it a single point of resource access.
- Data from `xml` files is parsed and stored into respective relations. For this `load_records` command is used from `admin.py` module.

### 4.2.3 Catalog Service

- Catalog service takes all available data from the database and creates a metadata capabilities file. It acts as a mediator between repositories which actually contains the data and the client.



- When a client requests for a query or a metadata service, catalog service checks if the database contains the information. If metadata information is requested, it replied directly to the client with the necessary files or response.

#### 4.2.4 Query Processing

- When a query or request for the data is processed by the catalog service, it acts as a mediator between client and the repository from which the data is intended. It creates a object of the required service and requests for the object on behalf of the client. Client authorizes to the catalog service and catalog service authorizes itself to the data repositories for the transfer of data. Catalog service takes the object reponse from the repository and send it in response to client in client specified format.

### 4.3 Results

Here are some results from the projects that explains and elaborates what kind of information can be get from catalog service.

- Information about service metadata : OGC:WMS  
This shows that registry is capable of providing OGC compliant Web map service type of service to the user
- List of available layers
  1. kgp:POPULATION
  2. kgp:bnk\_block\_boundary
  3. kgp:bnk\_block\_hq
  4. kgp:bnk\_district\_boundary
  5. kgp:bnk\_drainage
  6. kgp:bnk\_grampanchayat\_boundary
  7. kgp:bnk\_mouza\_boundary

## 8. kgp:bnk\_road

This shows the list of available data in the form of layers. Each layer represents data of a geographic location from a view point. Multiple layers can be overlapped on each other to better understand the data.

- Available operations

1. GetCapabilities
2. GetMap
3. GetFeatureInfo
4. DescribeLayer
5. GetLegendGraphic
6. GetStyles

Gives details about the operations that can be performed by the geoserver. For example, DescribeLayer provides information about a particular layer. GetMap returns a map of layer in multiple available formats.

- GetMap



Figure 4.3: Map of KGP BNK ROAD

Returns a map of particular layer. Can be superimposed to another map for better visualization.

- DrawMap

DrawMap overlays different map images on top of each other to better visualize and analyze the effect of desired operation. For example, it can be used to find area affected by flood, or to find division of regions based on religion.

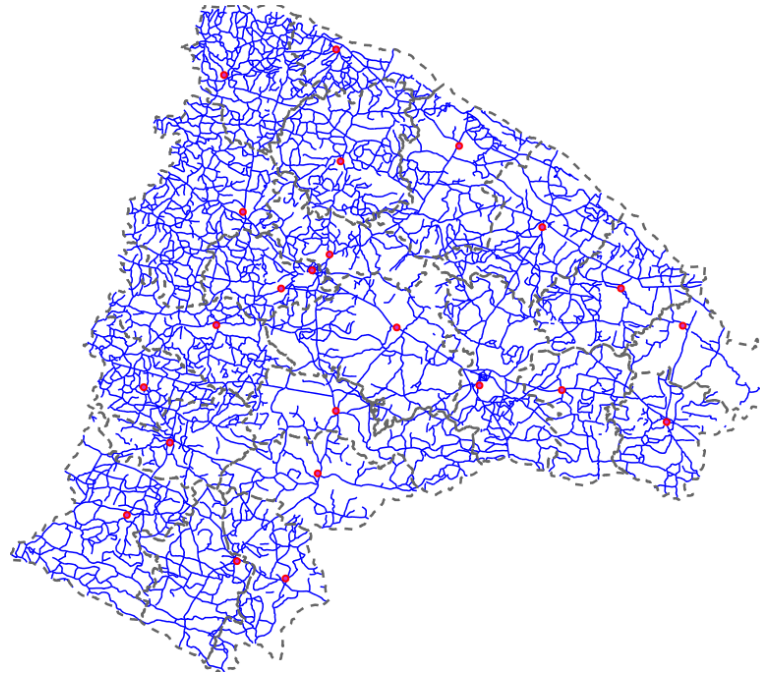


Figure 4.4: Map of KGP BNK ROAD

User can specify options to get the result image in desired format.

Options:

Layers = { kgp:bnk\_road, kgp:bnk\_block\_hq, kgp:bnk\_block\_boundary }

Width = 768

Height = 679

Format = image/png

- Information about specific layer:
  - Title — POPULATION
  - Name — kgp:POPULATION
  - Is Queryable — 1
  - Is Opaque — 0
  - Bounding Box —
  - minx — 68.52669525146484
  - miny — 8.086045265197754
  - maxx — 97.3387680053711
  - maxy — 35.8697509765625

Provides information about a particular layer, for example, name of the layer, title, bounding box information etc.

## Chapter 5

# Conclusion and Future Work

Geo-service portal acts as a underlying framework or foundation for various kind of higher level use cases. Main advantage of this registry service is that we can avail data from various repositories that has been crawled and indexed for others to use. Basic quering facilities can be provided directly on top of registry service. Using geo-spatial crawler we can get the latest geo-spatial data available on the web. Once this data is got from various geo-servers, various kinds of applications can be built on top of that. Building an OGC compliant web service catalog can also be beneficiary as already available software and services can use the registry for various kinds of services with little to no modification of their original code-base.

Geo-service portal can be used for many kind of applications. This applications and use cases include wide variety such as just getting the relevant spatial information for the application to doing in-depth spatial or temporal or spatio-temporal analysis. Other use cases can include finding the spread of the disease on large geographic area. Which of the areas can be affected by flood can also be founded using data from catalog. All this services can work if underlying foundation of catalog service is built.

### 5.1 Extension

In today's era, when creating a service reliability, scalability and availability is very crucial things to consider. If the data is hosted on one server and single server architecture is implemented then there might be a high chance of failure on high load. Also the growth of the users and query load on the server cannot

be predicted. For this reasons it would be better to consider a cloud based implementation for the above stated architecture. Cloud based implementation for crawler, registry service, and query processor can scale very well in the situations for high load. Cloud based implementation can also be good for availability because on situations of high load, infrastructure can be scaled up to cater the need of high load, in situations of low load the infrastructure can be scaled down to save power and investments. The registry itself can be replicated in the cloud at various geographical places to avail above stated high availability and reliability of the operations. Changes are also be easily and more efficiently done in the cloud because we don't have to shutdown and start the instances again. Cloud instances can be replaced in-place without shutting them down. This ensures versioning can be done of the software and eventual upgradtions and roll-out of updates can be provided without hiccups.

# Bibliography

- [1] Patil, Sonal, Shrutilipi Bhattacharjee, and Soumya K. Ghosh. *A spatial web crawler for discovering geo-servers and semantic referencing with spatial features*. International Conference on Distributed Computing and Internet Technology. Springer International Publishing, 2014.
- [2] Li, Wenwen, Chaowei Yang, and Chongjun Yang. *An active crawler for discovering geospatial web services and their distribution patternA case study of OGC Web Map Service*. International Journal of Geographical Information Science 24.8 (2010): 1127-1147.
- [3] Najork, Marc. *Web crawler architecture*. Encyclopedia of Database Systems. Springer US, 2009. 3462-3465.
- [4] Ahlers, Dirk, and Susanne Boll. *Location-based Web search*. The Geospatial Web. Springer London, 2009. 55-66.
- [5] Li, W., et al. *Semantic-based web service discovery and chaining for building an Arctic spatial data infrastructure*. Computers & Geosciences 37.11 (2011): 1752-1762.
- [6] Jiang, Jun, Chong-jun Yang, and Ying-chao Ren. *A Spatial Information Crawler for OpenGIS WFS* Sixth International Conference on Advanced Optical Materials and Devices. International Society for Optics and Photonics, 2008.
- [7] <http://geopython.github.io/pycsw-workshop/>
- [8] <https://geopython.github.io/OWSLib/>