Current Research Work
Challenges and Proposal

By Ahmet Sayar

Research Committee:

- *Prof. Geoffrey C. Fox (Principal Advisor)*
- *Prof. Randall Bramley*
- *Prof. Kay Connelly*
- *Prof. Melanie Wu*

Indiana University - April, 2007
Computer Science - Community Grids Lab (CGL)

Introduction

Geographic Information Systems (GIS) [2][9] and specifically Map Services are very crucial for the spatial decision making and situation assessment. Map Services play the key role in the situation assessment through accessing and rendering the data to create comprehensible representations. Maps are composed of layers created with spatial data sets which are mostly kept in databases or in plain text files. Geo-Science applications are earth related scientific applications and using spatial and temporal data for their simulations. Since Geo-Science applications require quick response times, the GIS services must enable accessing and processing these large data sets in a reasonable time period.

This document defines an architectural framework for integrating Map Services with Geo-Science grids [11[12]. In this framework, Geo-Science grids' requirements and spatial data characteristic introduces performance and interoperability challenges. We also take the extensibility and generality issues into considerations to create an efficient framework to be applied on any Geo-Science Grids.

In order to create extensible and general integration framework, we introduce 3-layer structured display. In the structure, bottom layers come from Map Services through coverage portrayal services (CPS) [13] or WMS, middle layer set comes from Map Services through Web Feature Services (vector data) and top layer set comes from Scientific Plotting (Sci-Plotting) Services. Since all the services are created in Web Service principles, it is possible extending the framework with any other third-party service.

In order to provide interoperability we use Open Geospatial Consortium (OGC) [1] and ISO/TC211 standards. For the data interoperability, we use OGC's standards. It defines vector data in feature collections and coverage data in any image types. Features are encoded in XML-structured Geographic Markup Language (GML) [10].

As it is very well known, using XML based data representation decreases the performance severely in distributed computing environments. In case of Geo-Science applications, because of the characteristics of the spatial data (variable sized and unevenly distributed) the situation becomes worse. In our proposition we mostly deal with these issues in order to make our proposed framework most wanted framework for Geo-science applications. We deal with the performance issues at the communication and transaction level. We do not take data and database level performance issues on which Database community work into account.

We first explain the current work (Chapter 1) and then, summarize the challenges of current work (Chapter 2). Performance issues are explained in Chapter 2.1. Finally Chapter 3 proposes a solution to the performance problems in the integration framework in order to make the framework useful and usable for Geo-Science Grids.

Before proceeding with this document, it is recommended for you to have a look at author's blog [7] and a complementary document about Map Services and GIS [16].

1. Integrating Map Services and Geo-Science Grids in SOA Principles

Integration is done at the layer level. Therefore, the Geo-Science Grids' outputs and Map Servers outputs should be represent-able in layers and synchronized as well. Our OGC compatible Map services provide their data in layers. We also use Google Map servers as Map services by embedding an intermediary middleware. This is explained in another document.

Regarding Geo-Sciences' outputs, we need to convert them into layers to integrate (overlay) them with the Map Services' layers. In order to do that we created Scientific Plotting Web Services. It is based on Dislin plotting libraries. After having wrapped them as Web Services [4], we enabled these libraries to be used by Science community.

There are three set of layers used to build a map with scientific data and information. Layer order is important to create reasonable and human interpretable maps. Please see the Figure **1**. *Layer-1* is the bottom layer created from raster data such as Google Earth and coverage data in image format (comes from Web Map Services). *Layer-2* is created from vector data such as state boundaries and seismic data coming from Web Feature Service, and *Layer-3* is created from processed data coming from simulation outputs of the Geo-Scientific applications. *Layer-3* is created at Sci-Plotting Server.

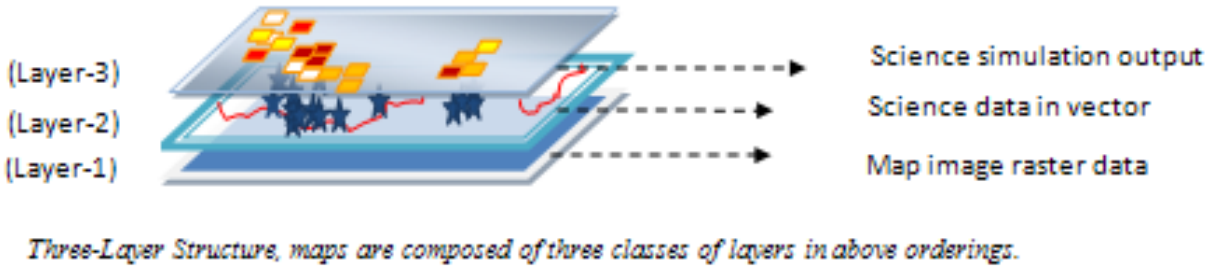


Figure 1: Integration output is three-layer structured maps. Layer-3 comes from Sci-Plotting, Layer-2 comes from WFS and Layer-1 comes from WMS or Coverage Portrayal services.

Geo-Science grids (or applications) are based on the data (called as features, spatial data or temporal data) related to the earth. They are defined by coordinates, bounding boxes, spatial reference systems, projections, geometry elements and, some other geo-related attributes and elements.

There are three different groups of layers and three main components in the architecture. Layers are explained in Figure **1**. Basic components of the system are Map Services, Sci-Plotting Services and User Portal (services in grey colors in Figure **2**). Regarding the relation of the components and the three main groups of layers; *Layer-1* and *Layer-2* come from Map Services and *Layer-3* comes from Sci-Plotting services. All the layers are displayed and map frames are animated through smart map tools in User Portal.

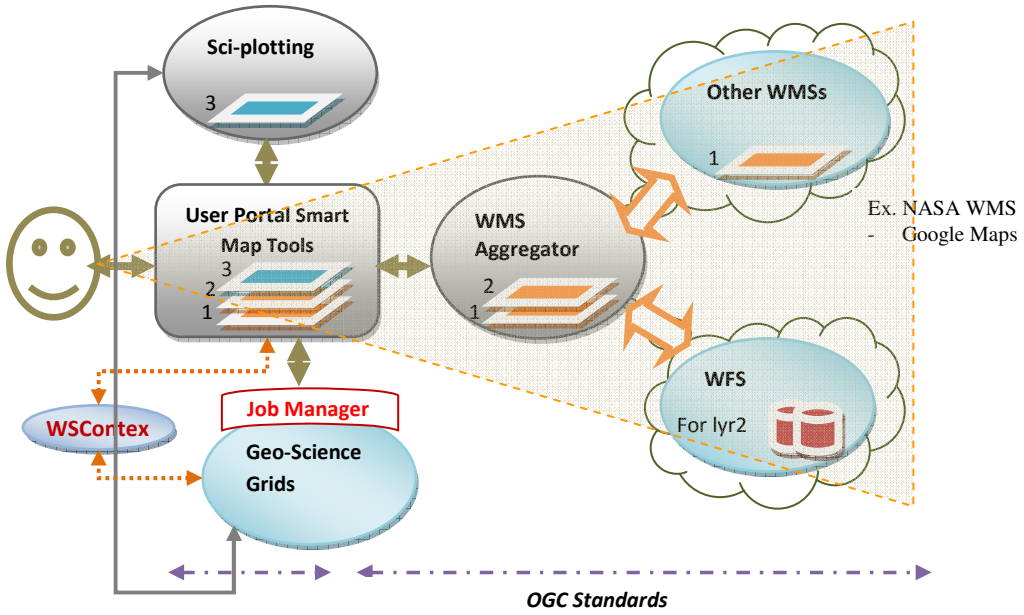


Figure 2: *Map Services and Geo-Science Grids integration framework. WFS and Job Manager (interface to Geo-Science Grids) are other CGL-Lab research projects*

*Major Components in the Framework (*Figure **2***):*

Web Feature Service (WFS) (by G. Aydin) [8] provide requested geographical information as GML feature collections. Data is kept in relational DB and upon request; WFS convert it to GML and returns. WMS interact with a Web Feature Service by submitting database queries encoded in OGC's Filter Encoding and in compliance with the OGC Common Query Language.

Web Map Service (WMS) [5][6] enables visualizing, manipulating analyzing geospatial data through maps. Map Servers typically compose maps as layers. Layers may come from distributed sources: Web Feature Services provide abstract feature representations that can be converted to images, and other Map Servers may contribute map images. WMSs can be federated and cascaded to create more detailed and comprehensible map images.

Aggregator WMS [5][9] provides all the interfaces functionalities and OGC-interoperability that any other OGC compatible WMS provides. Aggregator WMS also provides Google Maps in the image format such as jpeg which can be archived and manipulated depending on the application aims. Aggregator WMS is expected to improve the performance compared to conventional WMS. It uses load balancing and caching techniques (see Section 2 and Section 2.1) as Google Map servers do. It also provides properties set before run-time enabling different running modes such as streaming or non-streaming data transfer modes. We are going to test its performance.

Sci-Plotting Services: For the core functionality we use Dislin scientific-data plotting libraries. Dislin is a plotting library containing functions for displaying data graphically as curves, graphs, pie-charts, 3-D

color plots, surfaces and contours. All these services are wrapped as Web Services and integrated into the general visualization system as illustrated in Figure **2**.

Job Manager (by H. Gadgil) [19] and Geo-Science Grids: As a Job Manager we use HPSearch. It is simply a scripting environment for managing distributed workflows. Different Geo-Science applications (PI or Virtual California) require different set of parameters for the application users to trigger the job manager. This set of parameters and their order are defined earlier by the Job manager. Users provide required parameters through the project's user interface module deployed in to the user portal and trigger the application to run. After the application or science grids finish the task, job manager send the output link to the user waiting at the user portal. Then the user requests *Layer-3* from Sci-Plotting server which is created based on the data located at the link job manager returned.

WS-Context [20]: WS-Context specification is defined by OASIS. When multiple Web services are used in combination, the ability to structure execution related data called context becomes important. This information is typically communicated via SOAP [15] Headers. WS-Context provides a definition, a structuring mechanism, and a software service definition for organizing and sharing context across multiple execution endpoints.

When integration portal invokes the Job Manager for a specific Geo-Science Grids, Job manager starts running the chain of services and set the context in WS-Context and get a session id. This session id together with the string "ok it is started" is sent to the portal. Whenever the job is done at the Geo-Science Grids and the result is sent to a specific URL, Job manager sets the context belonging to specific id as "job is done" at the WS-Context server. Upon receiving "job is done" message through the WS-Context, portal access the URL and get the result through HTTP protocol.

2. Extensibility, Interoperability and Performance of the Framework

Title lists the challenges of the proposed integration framework explained roughly in Section 1. This chapter explains our approaches to come up with these challenges. Extensibility and Interoperability are explained shortly in the following two paragraphs. Our concern is mostly performance and scalability of the framework due to the reasons explained in Section 2.1.

Extensibility: We have created our proposed framework in Web Service principles [4]. Each service in the framework is a Web Services. Therefore, they can be used by outsiders and, any services (GIS) created in Web Service principles can be integrated to the framework. Web Services provide loosely coupling of services and extensibility.

Interoperability: In order to provide interoperability we use OGC and ISO/TC211 standards, in addition to creating services in Web Service principles. OGC defines vector data in feature Collections and features are encoded in GML. GML is an XML based data representation. As it is very well known, using XML based data representation decreases the performance severely. Geo-Science applications use large

data in size. Coding them in XML increases the size two fold or sometimes three fold. In this section we mostly deal with the performance issues and explain them in Section 2.1.

2.1. Performance and Scalability

We consider how to transfer data from WFS to WMS, how to parse and render the XML based large feature data collections efficiently and, how to create efficient communication channels.

Regarding the data transfer from WFS to WMS, we have improved the performance by implementing streaming data transfer. Streaming versions of WMS IS implemented by using NaradaBrokering publish-subscribe based messaging middleware [14]. See the explanations about Naradabrokering usage in streaming data transfer below.

Regarding the XML base large amount of data parsing, we use pull parsing techniques. We explain it in the following paragraphs.

NaradaBrokering: In case of transferring the result GML set in the form of string causes some problems when the GML is larger than some amount of size. Since the WFS returns the resulting XML document as an <xsd:string>, this has to be constructed in memory and the size will depend on several parameters such as the system configuration and memory allocated to the Java Virtual Machine etc. Consequently there will be a limit on the size of the returned XML documents. For these reasons we have investigated alternative ways for data transport and, researched the use of topic based publish-subscribe messaging systems for streaming the data. Our research on NaradaBrokering shows that it can be used to stream large amount of data between nodes without significant overhead. Additional capabilities such as reliable messaging and support for different transport protocols already inherent in NaradaBrokering show that it is a powerful yet easy to integrate messaging infrastructure. For these reasons we have developed a novel Web Map Service and Web Feature Service that integrate Open Geospatial Consortium specifications with Web Service-SOAP [15] calls and NaradaBrokering messaging system.

In case of streaming through Naradabrokering, the clients make the requests with standard SOAP messages but for retrieving the results a NaradaBrokering subscriber class is used. Through first request to Web Service called *getFeature*, WMS gets the topic (publish-subscribe for a specific data), IP and port on which WFS is streaming requested data. Second request is done by NaradaBrokering Subscriber. Even whole data is not received by WMS; WMS can draw the map image with the returned science data. This depends on the WMS's internal implementation.

Pull Parsing: In case of using pull parsing, the parser only parses what is asked for by the application rather than passing all events up to the client application as SAX parsing does. The pull approach of this parsing model results in a very small memory footprint (no document state maintenance required – compared to DOM), and very fast processing (fewer unnecessary event callbacks - compared to SAX).

Pull parsing does not provide any support for validation. This is the main reason that it is much faster than its competitors. Since all our services are OGC compatible and created in Web Service principles, we do not necessarily need validation. In OGC, services describe themselves by capability document and servers know each other by exchanging these document. If you are sure that data is valid (as in our case), or if validation errors are not catastrophic to your system, or you can trust validity of the capabilities document of the server you are in contact with, then using XML Pull Parsing gives the highest performance results. For example in WFS WMS communication case, since we know that WFS provides feature data in OGC's GML format, it is very advantageous skipping validation and using "pull parsing". Please see the article where pull parsing is compared with other leading Java based XML parsing implementations [17].

Figure **3** shows current systems's performance result improved with streaming data transfer and pull parsing techniques mentioned so far.

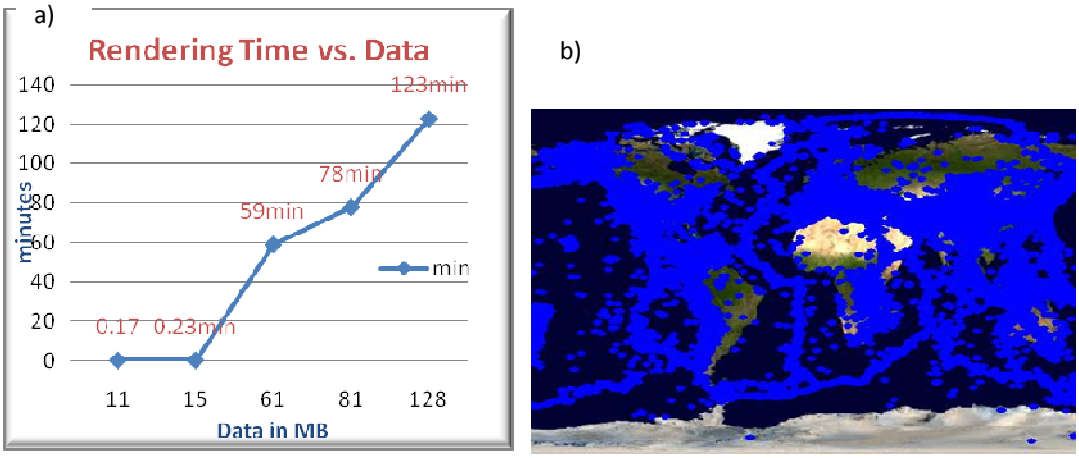


Figure 3: (a) Performance result of the current system. (b) sample output consisting of Layer-1 and 2. See Figure 1.

This figure teaches us valuable lessons in terms of the capabilities and limits of our implementation. From the below result we draw following conclusions. First, for small data payloads (less than 15MB) the response time is acceptable. However for larger data sets (more than 20MB) the performance decreases sharply and the response time is relatively long. Second, there exists a maximum threshold for the amount of data to be transported from WFS to WMS and rendered at WMS.

Current system test above shows that the performance is still not enough in order to meet Geo-Science grids' performance requirements. As you see, if the spatial data is over 20MB, integration framework is not feasible to use. Time column (y) in the Figure **3** (a) includes querying, transforming, rendering and displaying spatial data. In other words,

$$time_{(at\ the\ column)} = time_{(map\ is\ shown)} - time_{(client\ makes\ request\ for\ the\ map)}.$$

One can overcome these problems by caching and load balancing. However, the current load balancing approaches are not suitable for rendering of variable sized and un-evenly distributed spatial data. We therefore propose a framework of "dynamic load balancing with caching" overcoming performance and

scalability problems in rendering of distributed spatial data in order to make our map and plotting services feasible to use in Geo-Science grids (see Section 2.1).

However, implementing dynamic load balancing on processes such as spatial data rendering is not easy. Since we do not know the workload previously, the classic load balancing algorithms do not work for the variable sized and unevenly distributed data. The work is partitioned into independent work pieces, and the work pieces are of highly variable sizes. It is not possible to estimate the size of total work at a given worker server. Partitioning of feature data (represented in sets of polygons, line strings and points) is hard due to the spatial nature, varying sizes of the feature collections, and uncertainty about the query location. Problem is illustrated in Figure **4**.

Web Map Services are queried based on some criteria. Bounding box is the key criteria in the query. Load balancing and the partitioning are based on the values of the "*bounding box*" criteria. Bounding box values define the location of the places whose map images are going to be created. Archived spatial data are kept in WFS with their bbox values. Bounding box values are in the form of (minx, miny, maxx, maxy). For example, in Figure **4** (a) client needs a map of the earth defined by bounding box (bbox) value of (a,b,c,d).

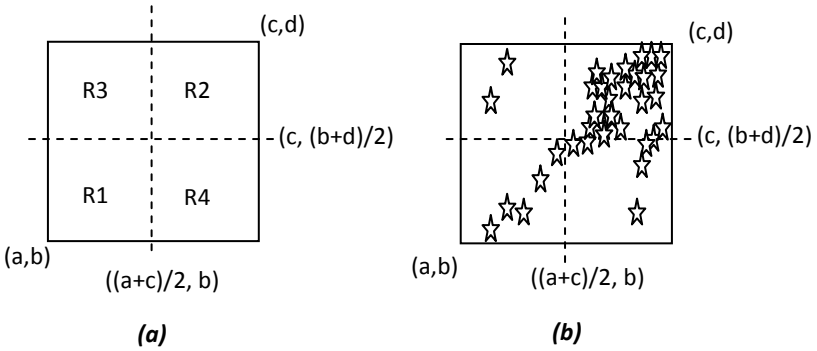


Figure 4: *Classic partitioning can not share the work equally among worker servers. Server assigned the partition of "((a+b)/2, (b+d)/2), (c, d)" gets the most of the work.*

We propose an innovative solution to the problem illustrated above in Section 2.1.

3. Dynamic Load Balancing with Caching for Spatial Data Rendering

In order to overcome the challenge explained in Section 2.1, we propose an innovative dynamic load balancing and caching on variable sized and un-evenly distributed spatial data rendering through introducing the critical regions concept. Please see [18] for more information.

Dynamic load balancing and caching definitely increases the performance but, it is not easy to share the workload equally among the worker servers once the spatial data is variable sized (depending on the location – defined in bbox) and un-evenly distributed over selected location in bounding boxes.

3.1. Brief Architecture

We take the caching and dynamic load balancing into considerations together. Our main concern in load balancing is the determining the best partition of the application data after utilizing cached data. Caching helps us to prevent redoing the jobs of querying and rendering for the data requested before. Partitioning the query ranges into equal sizes (as it is shown in) does not help to share the work to the worker map server equally. In order to solve this problem, we utilize from the *critical regions (CR)*. It will be explained in the Query Partitioning section.

Architecture is summarized in three orderly steps. These are

1. *Caching*
2. *Cached-data extraction and Rectangulation on un-cached data over the query ranges*
3. *Partitioning on rectangles (such as R1,2,3 in* Figure **5***) based on CR.*

In order to make these concepts more clear, let's give a concrete example: Example data is map image consists of NASA satellite base maps and earthquake seismic records (in blue dots). See Figure **5**.

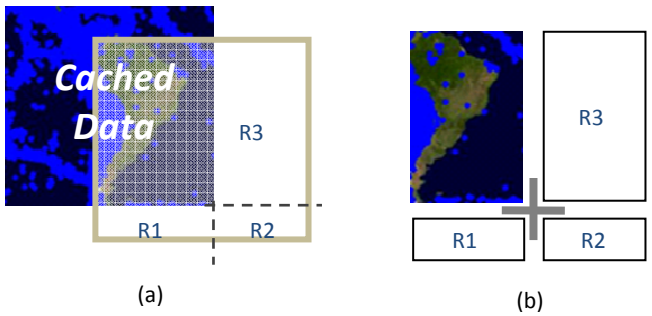


Figure 5: Illustration of the cached data extraction and rectangulation.

We get overlapped region of the data (as map image) through cached data extraction process, and remaining region in main query is partitioned through rectangulation process. According to OGC standards in GIS domain, queries are created with location parameter and location is defined in bounding box formats. Bounding box is a formula defining the region as a rectangle through coordinates of bottom left corner and top right corner. Ex Q(minx, miny, maxx, maxy).

Rectangles (R1, R2 and R3) go through the partitioning processes based on their positions to the critical regions (see Section 3.1.3). We append overlapped cached map image with the images returned to queries obtained through partitioning of the R1, R2 and R3. Partitioning techniques are applied on the query ranges when no cached data available (such as in case of first time calls) or applied on rectangles obtained by rectangulation process (such as R1, R2 and R3).

Caching and critical regions concepts are used in different domains for different purposes. Under the consideration of our proposed integration framework, caching is explained in Section 3.1.1 and critical region concept is explained in Section 3.1.3.

3.1.1. Caching

In order to explain our caching techniques clearly we first need to explain the way Map Servers work. Our implementation of WMS is multi-threaded, so it can serve multiple clients at the same time and provides data in the MIME type image format. Therefore, if we store the cached images in local file system it will cause trouble because of that all the threads share the local file system. In order to prevent this, we keep cached images as class objects. Whenever map server needs to use the cached data, converts it into image for a specific client without confusion because of that each client (browser) has its own thread and each thread has its own instances of the classes.

Caching will be utilized just by the successive requests. In other words, cached image will be kept till the next request comes. In order for the successive request to utilize the caching, its layer numbers and names should be the same, otherwise it will be counted as first time request and caching will not be utilized. According to our caching implementation, we don't need to use any metadata or tool to see if the image is already in hand. There will be only one object (an image class object) cached in the system for one thread (for a session). We do not need any cache space in the local file system.

3.1.2. Cached data extraction and Rectangulation

The methodology here is to remove the regions in the main query which overlap with the cached data and then, create rectangular sub-regions from the remaining main query in the form of bboxes. After removing the cached region from the requested bbox, we rectangulate the remaining parts as much as equally. Since we get rid of the cached data, the rectangulated regions obtained here look like the first time calls. In order to be able to utilize from the cached data according to the below algorithms, we assume the layer numbers and names are the same. In other case, successive call will be counted as first time call.

The bbox ranges of cached data and main query can be positioned to each other in four possible ways. These are (1) cached data covers main query, (2) cached data is covered by main query, (3) cached data and main query don't overlap and (4) they overlap partially. We explain data extraction and query rectangulation techniques for each group in the same order given above. Depending on their positions to each other, rectangulation techniques change. We basically explain the removal of parts in main query ranges which overlap with the cached data and creating the new bbox(es) for the remaining parts. After having obtained rectangles in bboxes, we make them go through partitioning process explained in Section 3.1.3. Since rectangles obtained in this section are a kind of first time calls (i.e. do not have any cached data – already extracted), they are partitioned depending on their positions to the pre-defined critical regions.

Why we need to make rectangulation: Our services are OGC compatible and implemented in Web Service principles. They accept the requests in predefined XML-structured queries such as *getMap* Web Service for WSM and *getFeature* Web Service for WFS. Queries to WMS and WFS are actually window shape range queries. Range queries are formulated in bbox by OGC standards. After extraction of cached data falling in main query range, the remaining part needs to be converted to rectangular shapes in order to create sub-queries in bbox to get data from WFS. This is why we make rectangulation after cached data extraction from queried-region. Please see the Figure **5** for the sample rectangles obtained through rectangulation process.

3.1.3. Query Partitioning through Critical Regions

For technical details in implementation level, see my paper here
http://complexity.ucs.indiana.edu/~asayar/proposal/loadBalancing5.pdf

The main goal of the query partitioning is partitioning a query into sub queries based on bbox values in order to share the total work among the worker notes as much equal as possible. However, the nature of the spatial data does not allow this. In order to overcome this problem, we use the critical regions defined in bboxes.

Query partitioning is applied on the range queries. Range queries (also known as spatial selection or window-queries) are used to select all line segments or points intersecting with a specified rectangular window. Range queries are represented in bbox by OGC [1]. Bbox is defined with the coordinates of bottom-left corner (minx, miny) and top-right corner (maxx, maxy). Queries are created based on these coordinate values. According to OGC standard specifications bbox values are defined in the form of (minx, miny, maxx, maxy).

Critical region is basically data intensive region defined by bbox for a critical data (See Figure **6**). For example for the Pattern Informatics application, earthquake seismic data is the critical data. Critical regions are defined as a set of bounding box values. For the time being we define it manually but for the future, we are trying to figure out dynamic and autonomous techniques. We have two approaches to create CR manually. In first approach, we get the list from the WFS providing critical data by contacting database admin. Since they maintain the data and provide upon request they know the characteristic of the data much more then any one else. The second approach is Map Server oriented. Map Server admin pre-run the system and make a request to data provider with the widest range of the bbox values. After getting the critical data and overlaying it on the base-map, admin analyses the created map and defines the list of critical regions. Critical regions are put into the properties file and Map Service is rerun.

Here is the sample critical region consisting of three bboxes for earthquake seismic data in California State. This definition is put into properties file of Map Service (manual definition of CRs).

Seismic_data = -124.23,31,-123.19,33.20: : -122.77,32.94,-122.05,33.01.: -114.95,34.30,-112.01,36.44

This bbox set is perfectly fine when the zoom-level is good to show California state but for more deeper or more higher zoom level it wont work as it works for California zoom level. Again as a short solution, Map Server admin sets the bbox values based on a zoom level works ok for any specific Geo-Science applications. For example above sample set of bboxes are good for PI California applications. We will be handling these issues by creating more efficient techniques to create and maintain CRs in the future.

Partitioning (binary):
There are two groups of requests from the partitioning point of view. These are first time requests and successive requests. Upon first request from the client (or successive call with no cached data available), the main query is divided into four equal parts (binary partitioning – partitioning into equal area). On successive requests of the client (with cached data available), cached data overlapping with the second query is extracted, following which the remaining part of the second query is rectangulated. Each rectangle obtained is then partitioned individually.

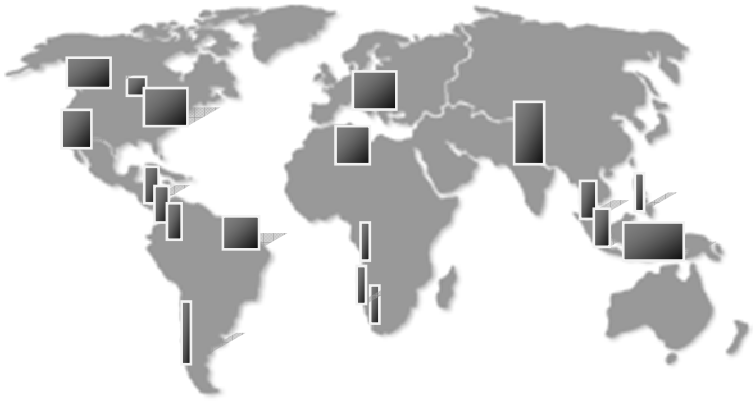


Figure 6: *A predefined set of sample critical regions*

We define our proposed partitioning technique as binary partitioning. By the term "binary" here, we mean at each level the region defined by bbox is divided into 2^n rectangles. More specifically our partitioning algorithm is defined as two-level (n=2) binary decomposition. So, at the end of partitioning process we obtain $2^2 = 4$ equal or un-equal regions for a region.

Partitioning techniques are grouped into two classes based on the sizes of the sub-regions. These are "*partitioning into equal area*" and "*partitioning into unequal area*". There are five possible cases illustrated in Figure **7** undergoing one of these two general partitioning techniques. These techniques are briefly explained in the following paragraphs.

Analysis of Figure **7**: In case-a main request's bbox falls in a critical region, in case-b there is no overlapping region between none of bbox, in case-c some of the bbox fall in a critical region, in case-d main query bbox covers a critical region and in case-e main query bbox and critical region bboxes overlap partially and totally at the same time. If there are more then one critical regions overlapping with main query bbox partially or totally, we firs look for totally overlapping CRs, then look for partially overlapping CRs. If there is at least one totally overlapping CR (such as d and e), then we ignore the partially overlapping CRs and partition the query according to covered CR.

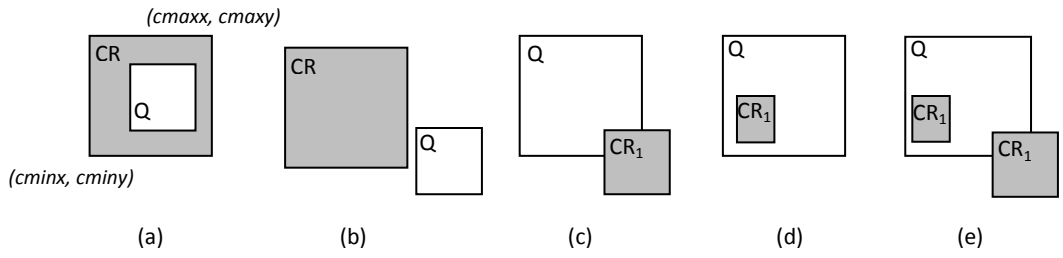


Figure 7: *All possible cases of the positioning of CR (critical region) and main query request (Q).*

In case of query ranges are positioned as in (a) and (b) against the critical regions, it is partitioned into sub-regions of *equal area*. In case of the cases (c), (d) and (e) the main query is partitioned into sub-regions of *unequal area*. In order to make explanation of the partitioning techniques more clear we name the critical region bbox as CR (cminx, cminy, cmaxx, cmaxy). As it is used before the main query is defined as Q (minx, miny, maxx, maxy).

Generalized partitioning techniques:

1. *Equal area partitioning:*

Partitions:

P_1 (minx, miny, $\frac{minx+maxx}{2}$, $\frac{miny+maxy}{2}$)

P_2 ($\frac{minx+maxx}{2}$+1, $\frac{miny+maxy}{2}$+1, maxx, maxy)

P_3 (minx, $\frac{miny+maxy}{2}$, $\frac{minx+maxx}{2}$, maxy)

P_4 ($\frac{minx+maxx}{2}$, miny, maxx, $\frac{miny+maxy}{2}$)

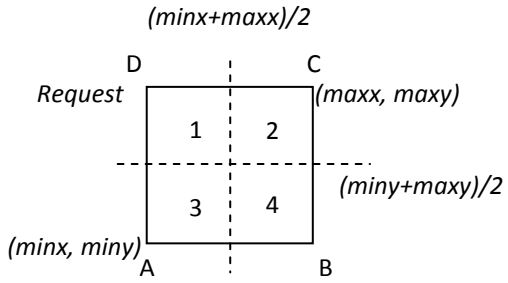


Figure 8: *Partitioning into the equal regions. Corresponds to (a) and (b) in* Figure 7

2. *Un-equal area partitioning:*

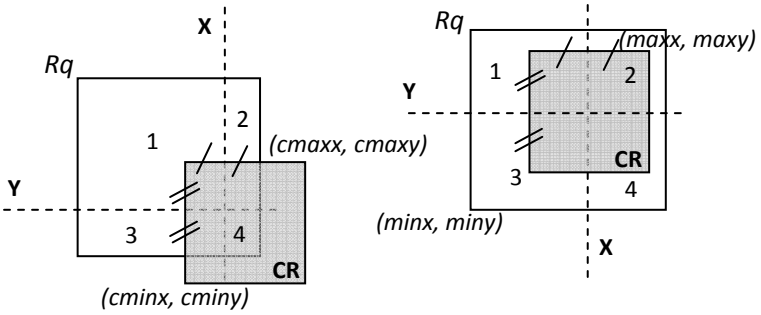
Partitions:

P_1 (minx, miny, X, Y)

P_2 (X+1, Y+1, maxx, maxy)

P_3 (minx, Y, X, maxy)

P_4 (X, miny, maxx, Y)



(a) Corresponds to "c" in Figure **7** *(a) Corresponds to "d" in* Figure **7**

Figure 9: *Two general cases of partitioning into un-equal regions.* Numbers represent the partitioned area. *Corresponds to (c), (d) and (e) in* Figure 7.

In case of a query overlapping more than one CR:

We mention additional situations to complete the partition definition (case e and, multiple CR versions of cases c and d). Techniques applied in cases of encountering cases "a" and "b" do not change depending on the multiple CR overlapping.

There are two approaches here to take in case of encountering multiple CR overlapping with the main query ranges.

First approach: we take the first occurrences of CR in cases of c and d. In case of e, take the first occurrences of CR that falls in main query bbox and ignore all the other CRs which are partially or fully covered by the main query ranges. This approach reduces the partition problem to a level which can be solved by the *un-equal* area partitioning technique.

Second Approach: We first find out the areas of each overlapping regions. We pick the CR with the largest overlapping area with the main query bbox, then, we apply the techniques explained above for one CR. Detailed steps are explained below:

Step 1: Find out all the CRs intercourse with main query ranges by using the techniques explained above, and put them in a list.

Step 2: Find out one CR from the list which are of the largest overlapping area. In other words, find out CR the one with the largest value of

$$(maxx - cminx) * (maxy - cminy)$$

This formula is specifically for the case pictured in Figure **9** (case a). Areas for the other cases are also found out similarly.

Step 3: apply the partitioning techniques over the selected CR in step 2. The technique applied here is base on the cases of one CR explained above

REFERENCES

[1] OGC (Open Geospatial Consortium) official web site http://www.opengeospatial.org/

[2] GIS Research at Community Grids Lab, Project Web Site: http://www.crisisgrid.org.

[3] Ahmet Sayar Sayar's project web site "Grid Map tools and Web Map Services page" available at http://complexity.ucs.indiana.edu/~asayar/gisgrids/

[4] Booth, D., Haas, H., McCabe, F., Newcomer, E., Champion, M., Ferris, C., and Orchard, D. "Web Service Architecture." W3C Working Group Note, 11 February 2004. Available from http://www.w3c.org/TR/ws-arch.

[5] Jeff De La Beaujardiere, OpenGIS Consortium Web Mapping Server Implementation Specification 1.3, OGC Document #04-024, August 2002.

[6] Kris Kolodziej, OGC OpenGIS consortium, OpenGIS Web Map Server Cookbook 1.0.1, OGC Document #03-050r1, August 2003.

[7] Ahmet Sayar's blog page available at http://ahmetsayar.blogspot.com

[8] Vretanos, P. (ed.), Web Feature Service Implementation Specification (WFS) 1.0.0, OGC Document #02-058, September 2003

[9] Ahmet Sayar, Marlon Pierce, Geoffrey Fox OGC Compatible Geographical Information Services Technical Report (Mar 2005), Indiana Computer Science Report TR610

[10] Simon Cox , Paul Daisey, Ron Lake, Clemens Portele, Arliss Whiteside, Geography Language (GML) specification 3.0, Document #02-023r4., January 2003.

[11] Fran Berman, Geoffrey C, Fox, Anthony J. G. Hey., Grid Computing: Making the Global Infrastructure a Reality. John Wiley, 2003.

[12] Foster, I. and Kesselman, C., (eds.) The Grid 2: Blueprint for a new Computing Infrastructure, Morgan Kaufmann (2004)

[13] Jeff Lansing., OWS1 Coverage Portrayal Service (CPS) Specifications 1.0.0, Document #02-019r1 February 2002.

[14] Pallickara S. and Fox G., "NaradaBrokering: A Distributed Middleware Framework and Architecture for Enabling Durable Peer-to-Peer Grids" ACM/IFIP/USENIX International Middleware Conference Middleware-2003, Rio Janeiro, Brazil June 2003

[15] Don Box, David Ehnebuske, Gobal Kakivaya, Andrew Layman, Dave Winer., Simple Object Access Protocol (SOAP) Version 1.1, May 2000,.

[16] Ahmet Sayar, Research status report available at http://complexity.ucs.indiana.edu/~asayar/proposal/summaryOfResearchWork.pdf

[17] Sosnoski, D. "XML and Java Technologies", performance comparisons of the Java based XML parsers. Available at http://www-128.ibm.com/developerworks/xml/library/x-injava/index.html

[18] Ahmet Sayar, Technical report "Dynamic Load Balancing with Caching for Spatial Data Rendering" http://complexity.ucs.indiana.edu/~asayar/proposal/loadBalancing5.pdf

[19] Harshawardhan Gadgil, Geoffrey Fox, Shrideep Pallickara, Marlon Pierce, Robert Granat, Proceedings of the IEEE/ACM Cluster Computing and Grid 2005 Conference, CCGrid 2005, Cardiff, UK. See also HPSearch Web Site, http://www.hpsearch.org.

[20] Bunting, B., Chapman, M., Hurlery, O., Little M., Mischinkinky, J., Newcomer, E., Webber J, and Swenson, K., Web Services Context (WS-Context) Specification, Version 1.0, July 2003.