Dynamic Load Balancing with Caching for Spatial Data Rendering

By Ahmet Sayar

Abstract

We have been working on creating Geo-Science grids and integrating them with Web based map and data rendering services such as WMS and Sci-Plotting. Integrating them with Geo-Science grids revealed that data rendering takes a lot of time due to querying, transforming, rendering and displaying spatial data. One can overcome these problems by caching and load balancing. However, current load balancing approaches are not suitable for rendering of variable sized and un-evenly distributed spatial data. We therefore propose a framework of "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.

Introduction

Geographic Information Services (GIS) 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.

Load balancing and caching are the first things coming to mind to provide high performance and scalability to provide service in a reasonable amount of time. However, in GIS applications using spatial data, due to the nature of the spatial data (variable sized and un-evenly distributed), general load balancing techniques can not give the expected results over the performance and scalability (see Section 1). In this document, we propose architecture of load balancing together with caching to solve this problem. Our architecture is based on the partition of the main work into smaller pieces. In order to find the best partitions sharing the work among the worker nodes, we introduced "*critical region"* concept and used caching. We do the partition on the queries sent to Map Server. Map Server behaves as failover and backup service by determining intelligently which server to assign the work to. Worker servers might be other Web Map Services (WMS) for map images or Web Feature Services (WFS) for the

vector data depending on the clients' requests. Our Web Map Services (WMS) and vector data services (WFS) are OGC (Open Geospatial Consortium) compatible services and implemented as Web Services.

In the proposed framework, caching will be considered together with the load balancing. Map Server cache the data as Image class object. Cached data is kept until the successive request comes. Map server does not create map for the region falling into the cached data, if the cached data and requested data layer names and numbers are the same. In this case, Map Server just cuts this region from the cached image and appends it to the map pieces returned to the partitioned queries.

Load balancing techniques are grouped into two, static and dynamic. Static load balancing methods divide and allocate the data pair prior to the computation process. In contrast, dynamic load balancing techniques divide and allocate work at runtime. Both techniques have the aims of increasing the scalability and high performance and, increasing the availability through disaster recovery. *We will be using dynamic load balancing in our applications.* We also use the round-robin algorithm to share the workload among the worker nodes, in case of that there are more partition than the available servers.

This paper is composed of three sections. Section 1 is the problem definition. Section 2 explains our proposed solution to the problem explained in Section 1. Section 2 starts with the explanations of some concepts in the problem domain such as "spatial data", "data rendering" and "web map server". Later, we explain the architecture about how dynamic load balancing is done through query partitioning with the help of critical regions and caching. Architecture is examined in two sub-titles. These are first, "data extraction and rectangulation" and second, "query partitioning based on critical regions". After giving problem definition and architecture for the proposed solution, we illustrate the proposed system through an application scenario. Section 3 explains the steps and framework for testing the solution on real Geo-Science Application. As Geo-Science Application, we have chosen Pattern Informatics (PI). It is a scientific simulation application for forecasting earthquakes all over the world.

1. Why do the classical load balancing algorithms not work? :

The short answer is that we don't know the workload previously. 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 1.

Web Map Services are queried based on some criteria. Bounding box values are one of these criteria defined 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. Bounding box values are in the form of (minx, miny, maxx, maxy). For example, in Figure 1.a client needs a map of the earth defined by bounding box (bbox) value of (a,b,c,d).

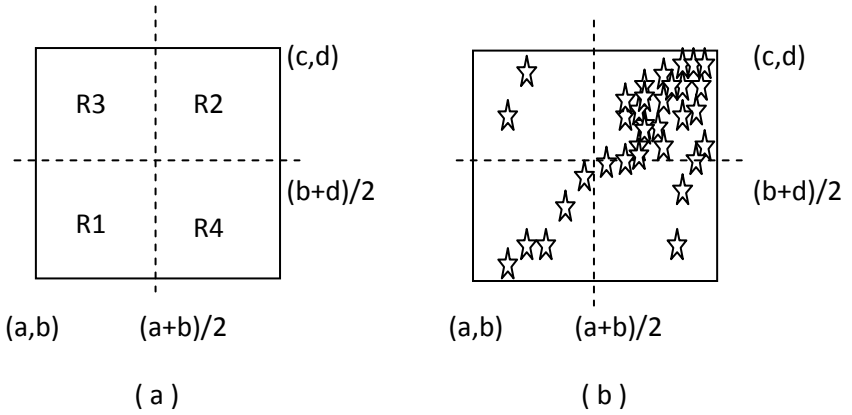


Figure 1: Classic partitioning can not share the work equally among worker servers. Server assigned the partition of "((a+b)/2, (b+d)/2), (c, d)" gets the most of the work.

2. *Proposed Solution:*

Dynamic Load Balancing through Query Partitioning with the help of Critical Regions and Caching for the variable sized and unevenly distributed spatial data

2.1. Background

System is based on query partitioning but In order to understand whole architecture we need to explain some terms. These are "spatial data", "data rendering" and "map servers". Proposed architecture is intended for solving the load balancing and caching problems in Map Servers through query partitioning with the help of pre-defined critical regions. "Query partitioning" and "critical regions" is explained in related sections.

Spatial Data: Spatial data sets are mostly in the form of vector data or raster data. Raster data is basically bitmap data such as NASA satellite images. Vector data is encoded in XML (such as GML defined by OGC as a standard representation of vector data) or kept in plain text files. In order to be rendered, vector data should have geometry elements with the coordinate values to locate on the globe. Geometry elements of the vector formats are encoded in sets of points, chains of line segments, and polygons.

Data Rendering: Data rendering is a tool for analyzing, interpreting and communicating numerical data. Rendering is the process of generating an image from a multi-dimensional description of the objects containing geometry, viewpoint, texture and lighting information. As well as being an important tool for researchers to help understand their data in a clear and concise way, data rendering can also be used to convey complex data to an audience that may have little understanding of the underlying processes represented in the image. Most people are familiar with the digital animations used to present meteorological data on television, although few can distinguish between the models of reality and the satellite images.

Web Map Server: Web Map Servers are Web Services running on a web server as a web application. Web Map Servers provide data rendering services. Possible clients to WMS are other WMS or browser based display applications. Our data rendering server is compatible to OGC standards. This enables it to be integrate-able to any other third party GIS service. It has standard interfaces and publicly available capabilities file and WSDL for the service descriptions. Besides being OGC compatible, it is Web Service based and it aims to provide high performance rendering for the Geo-Science applications. For this reason, it has dynamic load balancing and caching capabilities (*as explained in this document*). Our implementation of WMS is capable of using other standard OGC WMSs as worker servers to make dynamic load balancing. We can also easily convert our WMS into a standard WMS just by removing all the critical regions defined in its properties file. Properties file includes WSDL addresses of worker WMSs and critical region sets for the critical data.

2.2. Architecture

We think of the problem as a performance issue and take the caching and dynamic load balancing into considerations. 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 Figure 1, 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*. It will be explained in the Query Partitioning section.

Architecture is summarized in four orderly steps. These are

- Caching
- Cached data extraction through the main data and cached data ranges
- Rectangulation on un-cached data in main query ranges
- Partitioning based on Critical Regions.

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). See Figure 2.

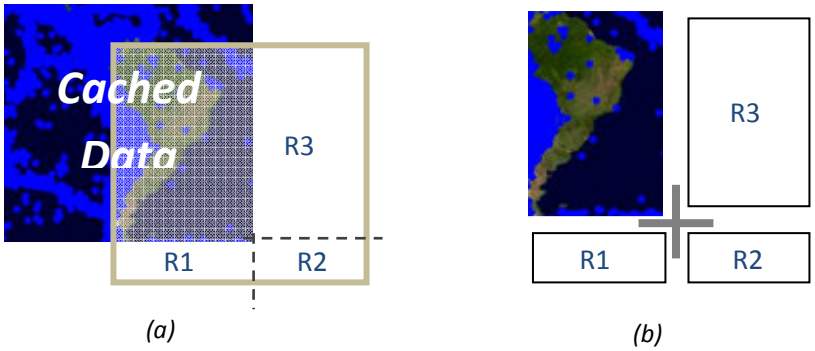


Figure 2: Illustration of the cached data extraction and rectangulation.

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. 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 cached data extraction and query rectangulation are explained in Section 2.2.1. General partitioning techniques based on critical regions are explained in Section 2.2.2.

2.2.1. Cached Data Extraction and Query Rectangulation

The methodology here is to remove the regions in the main query ranges which overlap with the cached data and creating rectangular sub-regions from the remaining main query ranges as in bboxes. After removing the cached region from the requested bbox, we rectangulate the remaining parts as much as equally. The rectangulated regions obtained here look like the requests in the category of the first time calls.

Caching and load balancing are implemented by considering the characteristics of the interactive user interfaces created for the Web Map Services. Interactive user interfaces for the Web Map Services are uniform and enable clients to query map data and geo-science data by standard map tools. Standard map tools are move, zoom-in and zoom-out. A client can not use any of these map tools together at the same time. Interfaces enable users to use these tools one at a time. Furthermore, 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. You need to keep this in mind throughout this chapter.

Before explaining the cached data extraction and query rectangulation we need to explain first how caching is done and what is cached.

Caching: In order to explain our caching techniques clearly we first need to explain the map server's related structure. Our implementation of WMS is multi-threaded, so it can serve multiple clients at the same time. 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 (see the section 2.2.2). 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.

The 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 (3) 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 2.2.2. Since rectangles obtained in this section are a kind of first time calls (i.e. do not have any cached data), they are partitioned depending on their positions to the pre-defined critical regions.

These cases are created by comparing the bboxes of the main query and cached data.

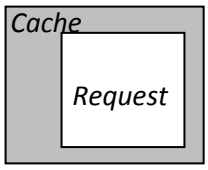
In the following sections we use bbox notation to represent main query, cached data and rectanguled areas in main query. We formulate the main query ranges as Q(minx, miny, maxx, maxy), cached data ranges as CD (minx_, miny_, maxx_, maxy) and rectangles obtained through rectangulation as R(minx, miny, maxx, maxy).

2.2.1.1. Cached data <u>covers</u> the entire main query

Request-query Q (minx, miny, maxx, maxy) and Cached data CD (minx_, miny_, maxx_, maxy)

Condition:

minx <minx_<maxx
& minx<maxx_<maxx
& miny<miny_<maxy
& minx<maxy_<maxx



Rectangulations: No need to rectangulations. We have ready to use data. No need successive cascaded calls to other feature data servers or map servers. Mapping Server does not make any successive queries, just cut the cached image data based on the query and cached data bbox values.

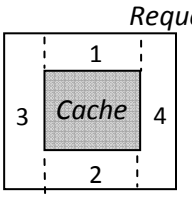
This case is encountered mostly through using zooming-in map tools interactively. Request is a successive call and cached data represents the last call.

2.2.1.2. Cached data <u>is covered</u> by the main query

Request-query Q (minx, miny, maxx, maxy) and Cached data CD (minx_, miny_, maxx_, maxy)
Condition: minx_<minx<maxx_ & minx_<maxx<maxx_ & miny_<miny<maxy_ & minx_<maxy<maxx_

Rectangulations:

R1 (minx_, maxy_, maxx_, maxy)
R2 (minx_, miny, maxx_, miny_)
R3 (minx, miny, minx_, maxy)
R4 (maxx_, miny, maxx, maxy)



This case is encountered mostly through using zooming-out map tools interactively. Request is a successive call and cached data represents the last call.

2.2.1.3. Cached Data and Main Query don't intercourse

Request-query Q (minx, miny, maxx, maxy) and Cached data CD (minx_, miny_, maxx_, maxy)
Condition:

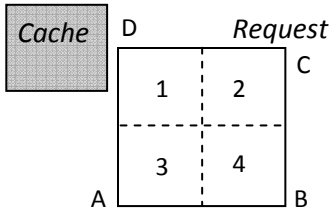
NOT (At least one point of Q is in CD) = NOT ((A is in CR) or (B is in CR) or (C is in CR) or (D is in CR))

Let's name the corners of Q as A, B, C and D.
So, each sub-condition above will be formulated as below.

(A is in CD) = (minx_ < A_x < maxx_) && (miny_ < A_y < maxy_)
(B is in CD) = (minx_ < B_x < maxx_) && (miny_ < B_y < maxy_)
(C is in CD) = (minx_ < C_x < maxx_) && (miny_ < C_y < maxy_)
(D is in CD) = (minx_ < D_x < maxx_) && (miny_ < D_y < maxy_)

Rectangulations:

R1 (minx_, maxy_, maxx_, maxy)
R2 (minx_, miny, maxx_, miny_)
R3 (minx, miny, minx_, maxy)
R4 (maxx_, miny, maxx, maxy)



This case is encountered when there is no cached data available or, there is cached data but successive calls do not intercourse with the cached data. User or client might not be doing his request through map tools such as move or zooming. He can make his request randomly and successive request might be unrelated.

2.2.1.4. Cached data partially overlap with the main query ranges - move

Request-query Q (minx, miny, maxx, maxy) and Cached data CD (minx_, miny_, maxx_, maxy)

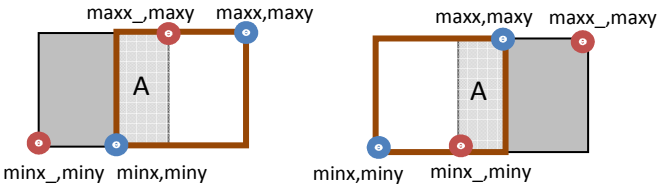
There are three different types of situations. These situations are differentiated depending on the source of possible action to create a query. So, successive queries might be created through (1) horizontal movements (left or right), (2) vertical movements (up or down) or (3) any other movements (angle movements) (left-up, left-down, right-up and right-down). In the following sub-sections we explain the load balancing strategies and partitioning algorithms by keeping the caching in mind.

In the below figures and partitioning formulas, A is the cached data falling into the main query ranges. Red box represents main query grey box represents cached data.

1. Horizontal movement

Condition: miny==miny_

Rectangulations:



Towards RIGHT (minx>minx_)

O + R(maxx_, miny, maxx, maxy)

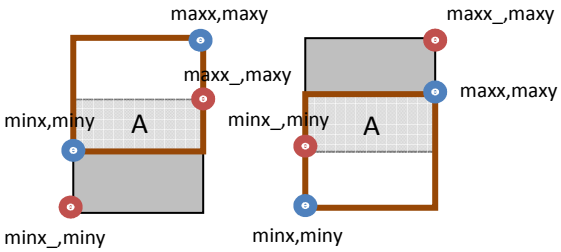
Towards LEFT (minx<minx_)

R(minx, miny, minx_, maxy_) + O

2. Vertical movement

Condition: minx==minx_

Rectangulations:



Towards UP (miny>miny_)

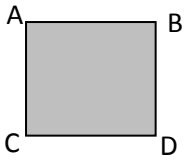
O + R(minx_, maxy_, maxx, maxy)

Towards DOWN (miny<miny_)

R(minx, maxy, maxx_, maxy_) + O

3. Any other movements

Let's name the corners of the cached data to make the following explanations clear.



A(minx_,maxy_) C(minx_,miny_)

B(maxx_,maxy_) D(maxx_,miny_)

There are four cases for the movement with angle. After the movement one of the corners A, B, C and D might be in the queried ranges. So, based on this classification we get four different cases as below:

A is in queried ranges
Figure 3.a:

O + R1(minx, miny, minx_, maxy_)
+ R2(minx_, maxy_, maxx, maxy)
+ R3(minx, maxy_, minx_, maxy)

C is in queried ranges
Figure 3.c:

O + R1(minx, miny, minx_,miny_)
+ R2(minx_, miny, maxx, miny_)
+ R3(minx, miny_, minx_, maxy)

B is in queried ranges
Figure 3.c:

O + R1(maxx_, miny, maxx, maxy_)
+ R2(maxx_, maxy_, maxx, maxy)
+ R3(minx, maxy_, maxx_, maxyy)

D is in queried ranges
Figure 3.d:

O + R1(minx, miny, maxx_, miny_)
+ R2(maxx_, miny, maxx, miny_)
+ R3(maxx_, miny_, maxx, maxy)

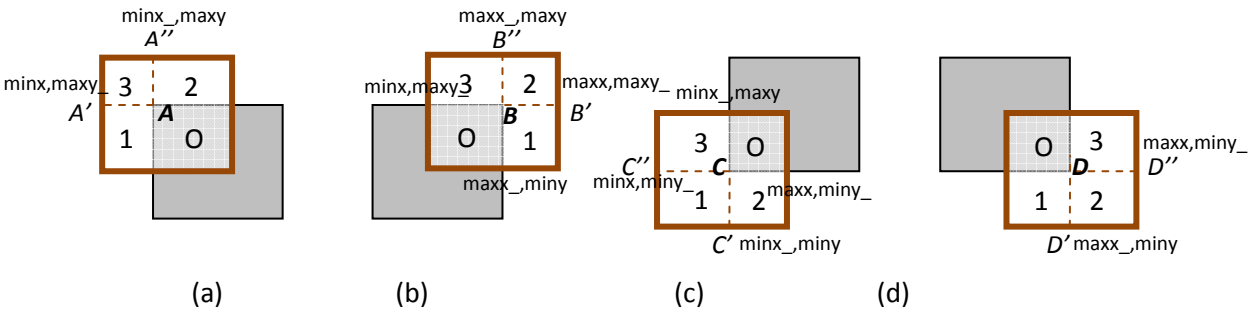


Figure 3: Possible movements with angle.

2.2.2. Query Partitioning

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. Windows are also called bbox in the document. The name bbox is used by OGC. Ranges are defined in rectangular areas which are called as bbox. 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).

We group the partitioning approaches into two general area depending on the partitions sizes:

- *Partitioning into sub-regions of equal area (Section 2.2.2.1)*
- *Partitioning into sub-regions of unequal area (Section 2.2.2.2)*

Since partitioning is based on critical regions we first explain critical statement concept.

Critical Regions: Critical region is basically data intensive region defined by a specific search criterion (See Figure 4). In the GIS case critical search criteria is the bounding box values. Therefore, critical regions in GIS are defined as data intensive regions defined by set of boundary boxes. Initially we define it in the properties file of the Mapping Server before run time manually. There are two ways to define the critical regions by this way. The easiest one is contacting the data provider's administrators and get from them. Another way is little more complicated such that Web Map Server admin pre-runs the system and make a request to data provider with the widest range of the bbox values. After getting critical data from data provider, admin defines the critical regions in smaller bboxes by looking at the returned critical data plotted on image maps. In the future, we wilt try to figure out a protocol allowing this to come from data providers (which are WFS in our application framework) in a 2-dimm array or in a text file.

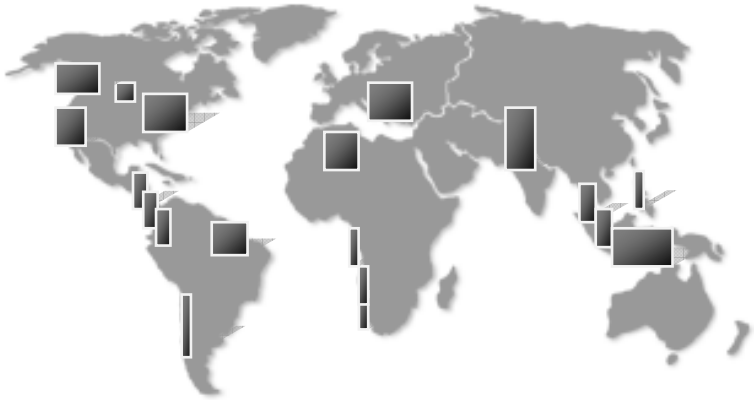


Figure 4: Globe showing predefined sample critical regions

Partitioning*:*
Partitioning is applied all the requested regions which do not overlap with cached data. There are two types of requests. These are first time calls or successive calls in a session. If it is first time call or successive call but there is no available cached data, whole query undergoes to partitionibg process. If it is successive request and there is overlapping cached data then partitioning process is applied after cached data extraction and rectangulation processes explained in Chapter 2.2.1.

Partitioning techniques are grouped into two general classes depending on the main query bbox's position to critical region bbox (see Figure 5). These are "partitioning into sub regions of equal area" and "partitioning into sub-regions of un-equal area". There are five possible cases illustrated in Figure 5: 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.

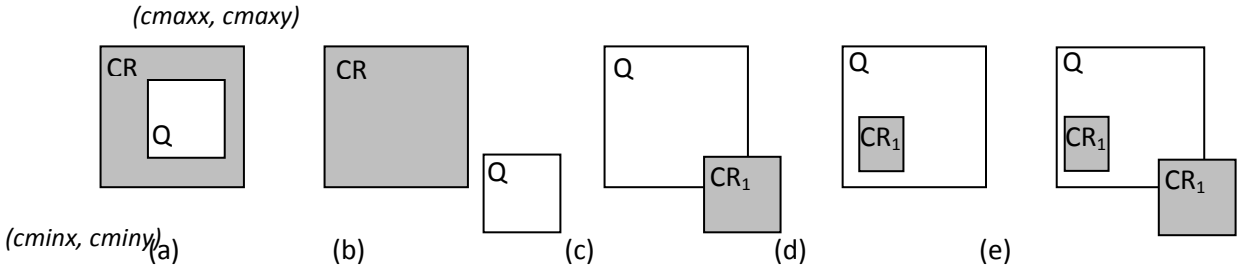


Figure 5: All possible cases of the positioning of CR (critical region) and main query request (Rq).

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 with the technique explained in Section 2.2.2.1. In case of the cases (c), (d) and (e) the main query is partitioned into sub-regions of unequal area wit the technique explained in item 2.

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).

2.2.2.1. Partitioning into sub-regions of <u>equal</u> area.

In order to formulate conditions clearly we name the corners of the main query bbox ranges as A, B, C and D. Coordinates x and y are represented as sub-indexes such as A_x

This is the simple case no need an algorithm to solve the partitioning problem. Cut the main query ranges into four equal areas. Partitioning is easy but we need to figure out the cases to apply this partitioning technique. Therefore, we define and formulate the conditions to apply this partitioning.

<u>*Condition*</u>: (Query ranges are covered by a CR) || NOT (intercourse with CR)

The main query ranges fall in to one of the pre-defined CRs.
(Query ranges are covered by a CR) = (cminx<minx<cmaxx) && (cminx<maxx<cmaxx)
&& (cminy<miny<cmaxy) && (cminy<maxx<cmaxy)

Or, query ranges do not intercourse with any critical region
NOT (intercourse with CR) = NOT ((A is in CR) or (B is in CR) or (C is in CR) or (D is in CR))

(A is in CR) = (cminx < A_x < cmaxx) && (cminy < A_y < cmaxy)
(B is in CR) = (cminx < B_x < cmaxx) && (cminy < B_y < cmaxy)
(C is in CR) = (cminx < C_x < cmaxx) && (cminy < C_y < cmaxy)
(D is in CR) = (cminx < D_x < cmaxx) && (cminy < D_y < cmaxy)

Partitioning:

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, $\frac{miny+maxy}{2}$)

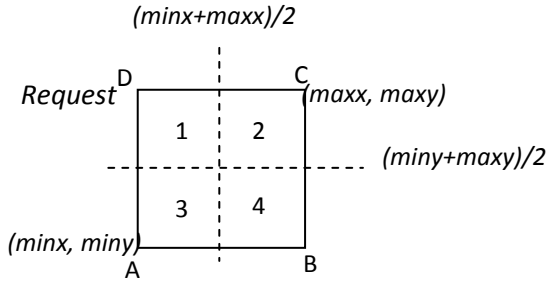


Figure 6: Partitioning into equal regions.
Corresponds to "a" and "b" in Figure 5

2.2.2.2. Partitioning into sub-regions of unequal area.

2.1. Query ranges intercourse with only **one CR**:
Sample cases are c and d in Figure 5.

Condition: (Query ranges are partially overlapping with CR) || (Query ranges cover CR)

Second part of the condition:
Condition of "Query ranges cover CR" is illustrated in Figure 7 b.
(Query ranges cover one CR) = (minx<cminx<maxx) && (minx<cmaxx<maxx)
&& (miny<cminy<maxy) && (miny<cmaxx<maxy)

First part of the condition:
(Query ranges are partially overlapping with CR) = *ELSE case of second condition defined in Section 2.2.2.1*. Condition is illustrated in Figure 7 a. In Figure 7.a, C is inside CR case is explained. At the partitioning formula we don't explain how we find X and Y coordinates.

Here we explain how to find best partition for the case of "C is inside CR":
C is the bottom-right corner name of the main query.

Best partition coordinates (X, Y): $\left(cminx + \frac{|maxx-cminx|}{2}, miny + \frac{|maxy-cmaxy|}{2}\right)$

We will be using critical regions in order to define best partition coordinates. These cases make us to generalize our formula for the partitioning. Lets say according to the critical regions and the other parameters we find out that (X,Y) is the best point to partition. So our sub-regions will be defined by the below ranges. Note that (X,Y) coordinate which is the gravity center of the critical region falls in the query range. In other case we would apply the partition defined in (a). If there is more than one critical region fallen in the query range we get anyone of them.

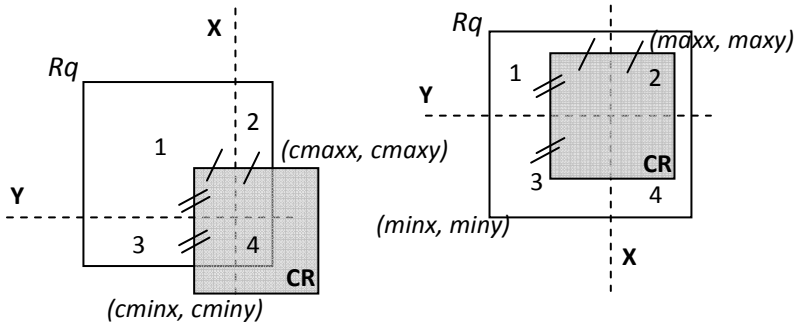
Partitioning:

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 5 *(b) Corresponds to "d" in Figure 5*

Figure 7: Two general cases of partitioning into un-equal regions. Numbers represent the partitioned area

2.2. Query ranges intercourse with **more than one CR**:
Sample cases are e and, multiple CR cases of c and d.

We mention additional situations to complete the partition definition (case e and, multiple CR versions of cases c and d). These are multiple CR cases of the conditions-techniques defined in Section 2.1 above. There are two approaches here to take in case of encountering multiple CR overlapping with the main query ranges.

First approach: 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 techniques explained in item 2.1 above.

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. We apply the techniques explained in Section 2.1. 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 in Section 2.1, 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

$(maxx - cminx) * (maxy - cminy)$ for the corner C is in the main query ranges.

Other cases are also found out similarly. See Figure 7 in order to understand the notations in the formula.

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 in Section 2.1

3. Testing the proposed system for the scalability by the real Geo-Science Application

- Earthquake forecasting Application – Pattern Informatics.

We will test our proposed framework on Pattern Informatics (PI) Geo-Science Application. PI is one of the Geo-Science projects that we have been applying our proposed grid framework on. PI is a technique developed at University of California, Davis for analyzing earthquake seismic records to forecast the locations of future large earthquakes. This forecast is based on analyzing the space-time patterns of past earthquakes to find possible locations where future large earthquakes are expected to occur. The PI method of earthquake forecasting provides a systematic, quantitative measure of variations (both increases and decreases) in seismicity and uses these variations to forecast regions likely to experience a large earthquake in the future.

Data for the past earthquakes are archived in relational database and integrated to the grid applications through our implementation of Web Feature Service (implemented by A. Galip). We use HPSearch tool to tie together and manage the distributed data sources and code.

I will first prove the integration architecture, integration of scientific grid with the map servers and plotting servers by the way of three classes of layer-overlaying.

After proving it works with common techniques of layering (which has been running just with small data), I will make stress test and see where the system halts.

I will use only caching techniques (no CR) and test the system.

Finally, I will use proposed dynamic load balancing (caching and critical regions) to improve performance.

UPDATES ARE COMING SOON …..