# *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 services. Integrating these services with Geo-Science grids applications 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 to overcome performance and scalability problems in rendering of distributed spatial data.

## *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 the key elements in the problem domain. The key concepts are basically "spatial data", "data rendering", "web map server", "critical regions" and "caching". In this section we explain how query partitioning is done with the help of critical regions and caching. Proposed dynamic load balancing techniques will be changing depending on the type of calls (first time calls -section 2.2.2 or successive calls -section 2.2.3). Section 3 explains the steps and framework for testing the solution on real Geo-Science Application. Our chosen application is Pattern Informatics (PI) for earthquake forecasts.

## *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).

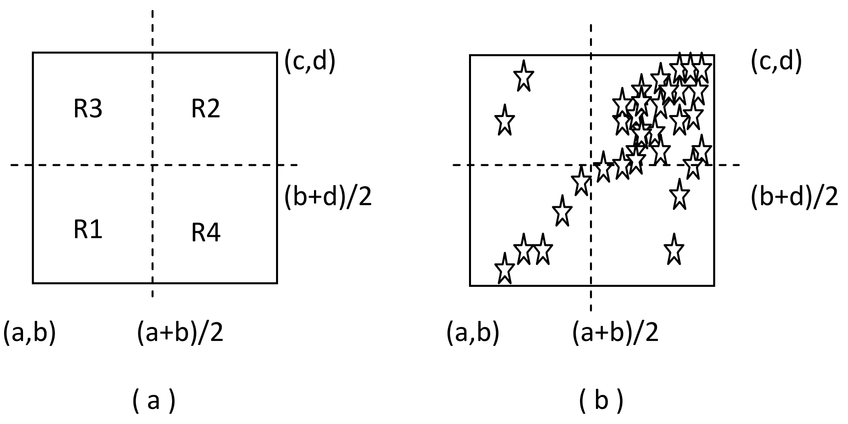


*Figure 1: Classic partitioning can not share the work equally among worker servers. Server assigned the partition of "( (a+b)/2, (b+d)/2 ), (c, d)" gets the most of the work.*

## 2. Proposed Solution: Query partitioning with the help of critical regions and caching

### 2.1. Definition of the Key Elements

System is based on query partitioning but In order to understand whole architecture we explain the key elements in this section. These are "spatial data", "data rendering", "map servers", "critical regions" and "caching". Proposed architecture is for solving the load balancing and caching problems in Map Servers through query partitioning with the help of pre-defined critical regions. "Query partitioning" is the main concept to implement dynamic load balancing algorithm defined in Section 2.2.1.

***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.

***Critical Regions:*** Critical region is basically data intensive region defined by a specific search criterion (See Figure 2). 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 value. 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.

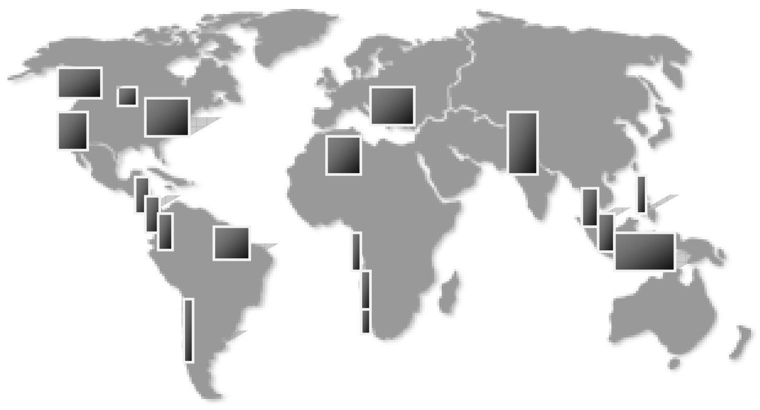


*Figure 2: Globe showing predefined sample critical regions*

***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.

## ***2.2. Architecture of Dynamic Load Balancing with Caching via query partitioning***

We think caching and dynamic load balancing all together. Our main concern in load balancing is the determining the <u>best partition</u> of the application data. Partitioning into equal areas 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 *<u>critical regions</u>*.

Since the architecture is based on finding out the best partition, we first explain what the query partitioning is. Later, we explain all the cases of the range queries in the architecture and how to find the best partition in these cases. All the possible cases are grouped into two. These are load balancing for the first time calls which does not utilize cached data (see Section 2.2.2), and load balancing for the successive calls which utilize the cached data (see Section 2.2.3).

### ***2.2.1. 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 in general. Classification of the partitioning techniques is based on the usage characteristics of the cached data. In some cases map server needs to use cached data in some other cases don’t. There are three groups of the partitioning techniques as summarized below:

(1) *Partitioning with no usage of cached data*
    a. *Partitioning into sub-regions of equal area (Section 2.2.1.1)*
    b. *Partitioning into sub-regions of <u>un</u>equal area (Section 2.2.1.2)*
(2) Partitioning with the usage of the cached data *(Section 2.2.1.3)*

### *2.2.1.1. Partitioning with no usage of cached data*

Partitioning techniques explained in this section are used in two cases.
First case: Techniques are used for the first time calls.
Second case: If it is a successive call and falling into one condition defined in Section 2.2.1.2, then server cuts out cached data and partition the remaining part in main query ranges with the techniques explained in Section 2.2.1.2.Since there is no cached data overlapping with these partitions obtained from the section 2.2.1.2, and they are in the form of bbox, they are handled by the techniques explained here.

Partitioning (with no usage of the cached data) techniques are grouped into two general classes depending on the main query bbox's position to critical region bbox (see Figure 3). 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 3: 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), the we ignore the partially overlapping CRs.

c, d and e cases might have multiple CRs. If there is more than one critical region whose gravity-points coordinate falls in the query range, then choose the one whose gravity-point is the closest to the gravity-point of the query bbox. Gravity point for the bbox is calculated as the coordinate values of the mid point of the rectangular bbox.

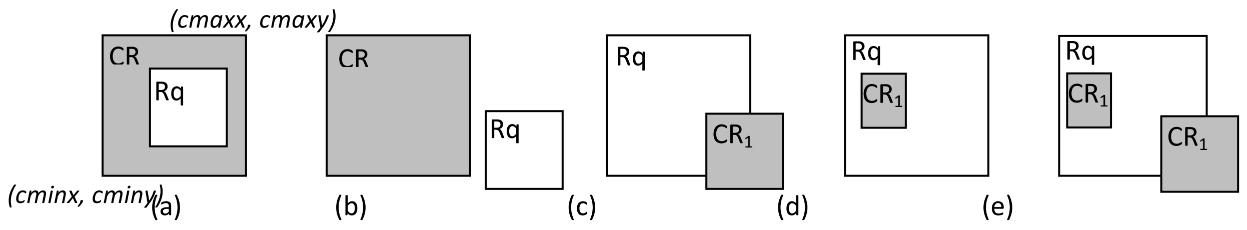


*Figure 3: All possible cases of the positioning of CR (critical region) and main query request (Rq).*

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 item 1. In case of the cases (c), (d) and (e) 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). Remember main query is defined as QR (minx, miny, maxx, maxy).

*1. Partitioning into sub-regions of equal 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 $A_x$

Condition: (Query ranges are totally Inside the CR) || NOT (Overlapping with CR)

(Query ranges are totally Inside the CR) = (cminx<minx<cmaxx) && (cminx<maxx<cmaxx)
&& (cminy<miny<cmaxy) && (cminy<maxx<cmaxy)

NOT (Overlapping 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)

In these cases we find the center of the rectangular bbox and cut the bbox into four geographically equal sub-regions. See the Figure x.

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

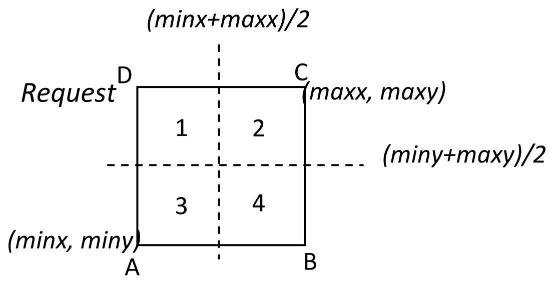


*Figure x: Partitioning into equal regions. Corresponds to "a" and "b" in Figure 3*

*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 3.

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

(Query ranges are partially overlapping with CR) = *ELSE case of condition defined in item 1*. Condition is illustrated in Figure y a.

Condition of "Query ranges cover one CR" is illustrated in Figure y b.
(Query ranges cover one CR) = (minx<cminx<maxx) && (minx<cmaxx<maxx)
&& (miny<cminy<maxy) && (miny<cmaxx<maxy)

Here, CR's bbox is defined as (cminx, cminy, cmaxx, cmaxy)

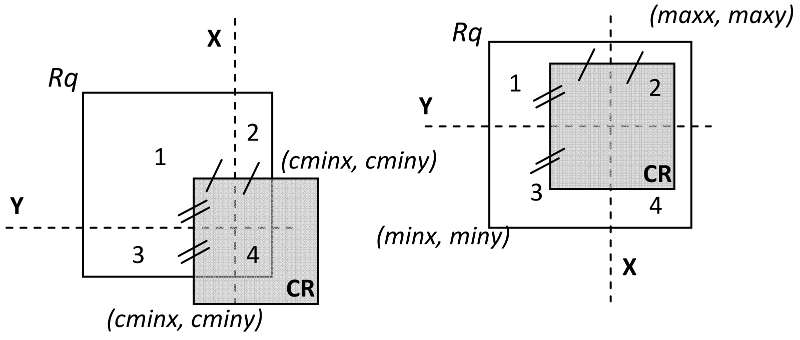
We will be using critical regions in order to define best partition coordinates. These cases make us to generalize our formula for the partitioning. Lets say according to the critical regions and the other parameters we find out that (X,Y) is the best point to partition. So our sub-regions will be defined 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.

$P_1$ (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 3* *(b) Corresponds to "d" in Figure 3*

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

*How to find (X, Y) coordinate to complete the partition:*

$X = minx\_ + \frac{|maxx - minx\_|}{2}$ $Y = miny + \frac{|maxy - maxy\_|}{2}$

2.2. <u>Query ranges intercourse with **more than one CR**:</u>

Sample cases are e and, multiple CR cases of c and d.

We mention here, additional condition 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 item 2.1 above.

There are two approaches here to take:
<u>First approach</u>: 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.

<u>Second Approach</u>: We first find out the area of overlapping region. We pick the CR with the largest overlapping area with the main query bbox.

Step 1: Find out all the CRs intercourse with main query ranges by using the techniques explained in Item 2, 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 2X x 2Y value.

Area of overlapping = 4XY     X and Y are calculated in item 2.1.

#### *2.2.1.2. Partitioning with utilizing of the cached data.*

*Example cases are Figure 7.a, b, c, d.*

Cached data and main query ranges can be positioned in three possible ways. Depending on their positions to each other, partitioning techniques change. The possible positions are listed in Table 1. We disregard the critical regions in this section. 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 partitions in bboxes, we make them go through another partitioning process explained in Section 2.2.1.1. Since partitions obtained in this section are a kind of first time calls (no cached data), they are re-partitioned depending on their positions to the pre-defined critical regions.

When cached data bbox partially or totally overlap with the query bbox, map server does not create map image for the cached part of the queried data. It directly reads from the cached map image with the help of bbox of the cached data and ranges defined in main query. I explain the algorithm below. The part of the map image falling in cached data is merged with the image parts returned from the worker map servers as an answer to the partitioned queries. We grouped the cases into three as listed in Table 1. These cases are created by comparing the bboxes of the main query and cached data. Main query range was defined earlier as R(minx, miny, maxx, maxy). Let say cached data bbox ranges are defined as CDR (minx_, miny_,maxx_,maxy).

| ***a.*** **Cached data covers the entire main query ranges** | | ***b.*** **main query covers all the cached data ranges** | | **c. Partially overlapping** |
|---|---|---|---|---|
| *Figure 8.a: Do nothing case. Cut from the cached data*<br><br>**Condition**:<br>minx<minx_<maxx &<br>minx<maxx_<maxx &<br>miny<miny_<maxy &<br>minx<maxy_<maxx<br><br>**Partitions**:<br>No partitions. Mapping Server does not make any successive queries, just cut the cached image data based on the query and cached data bbox values. | | *Figure 7.d: Partition for uncovered parts + cut from the cache*<br><br>**Condition**:<br>minx_<minx<maxx_ &<br>minx_<maxx<maxx_ &<br>miny_<miny<maxy_ &<br>minx_<maxy<maxx_<br><br>**Partitions**:<br>R1(minx_, maxy_, maxx_, maxy)<br>R2(minx_, miny, maxx_, miny_)<br>R3(minx, miny, minx_, maxy)<br>R4(maxx_, miny, maxx, maxy) | | Figure 7.a, b, and c<br><br>**Condition:**<br>See Section 2.2.1.3.1<br><br>**Partitions**:<br>See Section 2.2.1.3.1 |

*Table 1: Partitioning with utilizing of the cached data is grouped into three. Partially overlapping case is explained in the following chapter.*

*2.2.1.2.1.* Cached data partially overlap with the main query ranges (Figure 7.a, b, and c)

The methodology here is to remove the regions in the main query ranges which overlap with the cached data and creating rectangular sub regions as bboxes from the remaining main query ranges. After removing the cached region from the requested bbox, we partition the remaining parts as much as equally. The partitions 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 and load balancing algorithm explained in Section 2.2.3 is going to be applied. You need to keep this in mind throughout the following explanations.

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 after horizontal movements (left or right), vertical movements (up or down) or 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.

Cached data is handled introduced through movement actions and zooming actions. Movement actions are also grouped into angle movement, horizontal movement and vertical movement.

*A. Movement Actions:*

In the below figures and partitioning formulas, O is the cached data falling into the successive query ranges. O is cut from the cached data and appended to the returned images which are defined in query ranges R.

- 1. Horizontal movement (miny==miny_)
  Partitioning after cached data (O):

| Towards RIGHT (minx>minx_) | Towards LEFT (minx<minx_) |
|---|---|
| Figure 4.a: | Figure 4.b: |
| O + R(maxx_, miny, maxx, maxy) | R(minx, miny, minx_, maxy_) + O |

- 2. Vertical movement (minx==minx_)
  Partitioning after cached data (O):

| Towards UP (miny>miny_) | Towards DOWN (miny<miny_) |
|---|---|
| Figure 4.c: | Figure 4.d: |
| O + R(minx_, maxy_, maxx, maxy) | R(minx, maxy, maxx_, maxy_) + O |

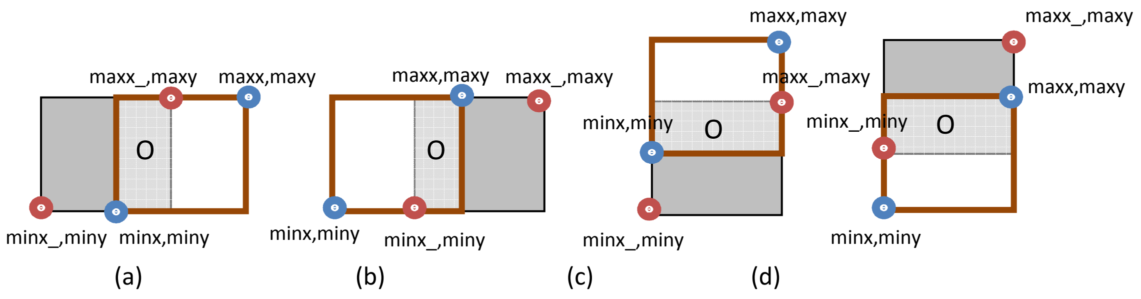
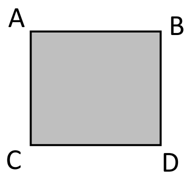


*Figure 4: Possible horizontal and vertical movements.*

- 3. (any other movements) Movement with angle
  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 5.a:

O + P1(minx, miny, minx_, maxy_)
+ P2(minx_, maxy_, maxx, maxy)
+ P3(minx, maxy_, minx_, maxy)

C is in queried ranges
Figure 5.c:

O + P1(minx, miny, minx_,miny_)
+ P2(minx_, miny, maxx, miny_)
+ P3(minx, miny_, minx_, maxy)

B is in queried ranges
Figure 5.c:

O + P1(maxx_, miny, maxx, maxy_)
+ P2(maxx_, maxy_, maxx, maxy)
+ P3(minx, maxy_, maxx_, maxyy)

D is in queried ranges
Figure 5.d:

O + P1(minx, miny, maxx_, miny_)
+ P2(maxx_, miny, maxx, miny_)
+ P3(maxx_, miny_, maxx, maxy)

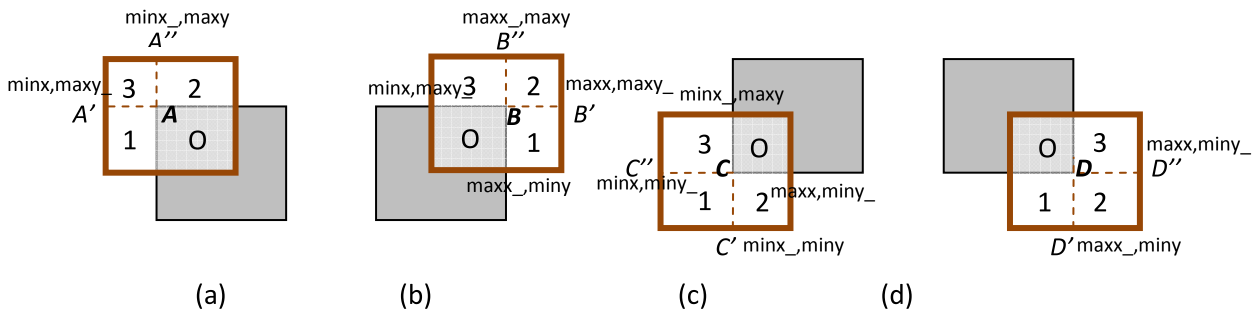


*Figure 5: Possible horizontal and vertical movements.*

We group the requests into two, first time calls and successive calls. First time calls are done without caching on data. Successive calls utilize the caching. Therefore, dynamic load balancing algorithm will be changing depending on the request type.

## *B. Zooming In-Out Actions*

We first utilize the caching (see Section 2.1 for Caching), if caching is not enough to meet the requested data, we make bbox partitioning for the remaining regions in accordance with the load balancing algorithms. There are two generalized cases. In first case caching doesn't meet the requested data as shown in Figure 7. Figure 7 shows three possible cases of load balancing with caching. (a) represents partially overlapped region, (b) represents fully overlapped region and (c) represents the cases with no overlapped regions. In (c) application can not utilize from the load balancing. In the second case, (do nothing case shown in Figure 8.a), caching meets the requested data. In other words, successive request's bbox is inside the cached image's bbox

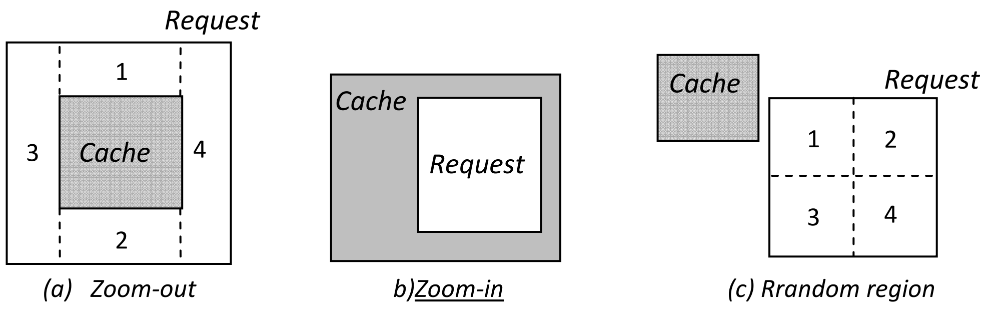


*Figure 8: Utilizing of the cached data.*

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.

**UPDATES ARE COMING SOON …..**