Journal of Information Engineering and Applications
ISSN 2224-5782 (print) ISSN 2225-0506 (online)
Vol 2, No.1, 2012

Concept for a Web Map Implementation with Faster Query Response

M. A. Bashar[1*] Monirul Islam[2] M. A. Chowdhury[2] M. P. Sajjad[2] M. T. Ahmed[3]

1. Department of Computer Science and Engineerig, Comilla University Comilla, Bangladesh
2. Structured Data System Limited (SDSL), Lalmatia, Dhaka-1207, Bangladesh
3. Department of Information and Communication Technology, Comilla University Comilla, Bangladesh

* E-mail of the corresponding author: basharcse@gmail.com

Abstract

Vector data and in particular road networks are being used in many application domains such as in mobile computing. These systems would prefer to receive the query results very quickly. Lots of research is going on to make the query response faster. One technique is to compress vector data so that they can be transferred to the client quickly. If we look different compression technique that are used to make the response faster, we will see that some of them do not make the response fast enough and some of them make response fast but very complex to implement. We report the concept for the implementation of a web map with a simple compression technique to send query response to the client, and found it making response fast. We have used some open source/free components to make the development quick and easy. This paper may work as a guide line for quick implementation of a web map.

Keywords: Web Map, PostGIS, Geoserver, GeoWebCache, Compression.

1. Introduction

Consisting of thousands of points and line segments representing various geographical features, this data requires a significant amount of time to be generated at the server side and is even more time-consuming to be rendered at the client side. The common practice to address the performance/storage issue in GIS applications is to send a raster image, which is a rendition of requested geospatial data at the server side, to the client. A new map was generated every time users panned to a different location or changed zoom level (since there are an infinite number of combinations of map extents and zoom-levels it was impossible to generate those maps in advance).

Google map developers had broken with that tradition [1]. They opted for a solution where maps could be produced in advance and served as small tiles for assembling into one big image at user end. The advantage of this approach is consistency of appearance and graphical quality of the map (which was rare prior to release of Google Map!) and, probably more important, enormous scalability that could be achieved. There is no need for server side processing to generate maps and individual map tiles are much smaller than the whole map presented at the user end, so they are able to be delivered and displayed much faster. The trade off was a big effort up front to generate nice looking maps and the need to fix zoom levels rather than allowing a continuous zoom, as is the case with the traditional approach. Pre-designed map tiles approach is brilliant for speed and performance but it does not allow for dynamic map content.

Transmitted raster image is only a visual representation of the geospatial data and hence does not include the geometric objects and corresponding metadata. Instead, the objects are rendered in the image and some of the corresponding metadata is superimposed as limited labels. Any subsequent query (e.g., nearest neighbor query) in the client results in another handshake with online server which hosts the original dataset. That is, the client application cannot manipulate the results for further processing. A solution to this problem is sending the original query result in vector data format to the client to enable further interaction with it and to preserve the metadata. This way, server sends the

vector data instead of the raster image to the client that enables users to further interact with query results (e.g., to select a specific geometric object or to issue a query based on returned results). An example of such approach is recently taken by Yahoo! Maps Beta [2] which allows users to highlight different sections of a path (i.e., different line strings). Such level of interactivity can greatly improve user experience with the system and enable more sophisticated query analysis on the client [3], [4].

It is important to compress road vector database so that we can send them to client side very quickly. A careful study of road vector databases in general and such query results in particular reveals that the data returned to the client is highly repetitive in nature and shows strong correlation. Such data behavior suggests finding a way to reduce this redundancy. By finding the difference of successive point of road we can eliminate a huge amount of redundancy from the data before compressing it. Also the nature of road vector database suggests us that if we use gzip we will achieve a huge amount data compression. After doing these two steps we send data to the client and achieve a faster response from server in practice. Novelty of our compression technique is that it is simple and works fine in real life.

2. Analysis of Different Compression Techniques

The problem of vector data compression has been addressed by two independent communities:

- Data compression community that takes numerical approaches embedded in compression schemes to focus on the problem of compressing road segments [5]–[7]. The advantages of these methods lie in their simplicity and their ability to compress a given vector data up to a certain level. However, the drawback of using a pure data compression approach is that it ignores the important spatial characteristics of vector data inherent in their structure. A successful approach must be devised with the rendering process and user's requested level of details in mind as this is the final deliverable of the entire process. Also none of the above references study the efficiency of their approaches with regards to overall response time.
- GIS community that uses hierarchical data structures such as trees and multi-resolution databases to represent vector data at different levels of abstraction [8]-[19]. With most of these methods, displaying data in a certain zoom level requires sending all the coarser levels. Furthermore, using generalization operators introduces the issue of choosing which objects to be displayed at each level of abstraction. While both approaches have their own merits, neither of these techniques blends compression schemes with hierarchical representation of vector data. Furthermore, most of the above approaches do not perform empirical experiments with bulky real-world data to study the effect of performing such compression techniques on the client, transmission and server times.

One of the most well-known line generalization and simplification schemes proposed in the GIS community is the Douglas-Peucker algorithm [12]. The idea behind this algorithm is to propose an effective way of generalizing a linear dataset by recursively generating long lines and discarding points that are closer than a certain distance to the lines generated. More specifically, this algorithm takes a top down approach by generating an initial simplified poly-line joining the first and last poly-line vertices. The vertices in between are then tested for closeness to that edge. If all vertices in between are closer than a specified tolerance, $\varepsilon > 0$, to the edge, the approximation is considered satisfactory. However, if there are points located further than ε to the simplified poly-line, the point furthest away from the poly-line will be chosen to subdivide the poly-line into two segments and the algorithm is recursively repeated for the two generated (shorter) poly-lines.

The Douglas–Peucker algorithm delivers the best perceptual representations of the original lines and therefore is extensively used in computer graphics and most commercial GIS systems. However, it may affect the topological consistency of the data by introducing self-intersecting simplified lines if the accepted approximation is not sufficiently fine. Several studies propose techniques to avoid the above issue known as self-intersection property [17]-[19]. For instance [17] proposes two simple criteria for detecting and correcting topological inconsistencies and [18] present an alternative approach to the original Douglas-Peucker algorithm to avoid self-intersections without introducing more time complexity. Finally, based on Saalfeld's algorithm [19] propose an improvement to detect possible self-intersections of a simplified poly-line more efficiently. However, due to high cost of performing the proposed generalization in real-time [19] suggest a pre-computation of a sequence of topologically

consistent representations (i.e., levels of detail) of a map which are then progressively transmitted to the client upon request. There is a nice approach [20] which mainly differs from Zhou and Bertolotto [19] in that they do not store multiple representations of data at different levels offline and they perform the entire process of generating user's requested level of detail on the fly. Another important difference is that Zhou and Bertolotto [19] propose sending the coarsest data layer to the user initially and then progressively transmitting more detailed representations of the same data to the client. However, [20] only send the single desirable level of detail requested, to the user. Therefore, as the need for having more levels of detail increases, use of their proposed progressive transformation increases the amount of data being transferred redundantly. Furthermore, as opposed to Zhou and Bertolotto [19], [20] do not focus extensively on topology preservation; however, it is important to note that their aggregation does not affect topology at all. Also the finest level of detail will contain the original data and thus completely preserves topology. For all other levels of detail their visual investigation of the results show strong indication that they do preserve topology.

Another work based on the Douglas–Peucker algorithm is the work of Buttenfield [11] in progressive vector data transmission over the web. Similar to Zhou and Bertolotto [19], Buttenfield proposes a hierarchical simplification preprocessing where each packet stores the representation of a complete vector at some intermediate level in a tree. Packets are constructed using the Douglas–Peucker algorithm for different levels of detail and based on the user request; the entire row of a certain height is transferred over the network. Buttenfield argues that "transmission time and disk space remain as two significant impediments to vector transmission". Again [11] does not report any experimental evaluation of the proposed system and it is only tested on single poly-line packets contained in a small geographic database and the efficiency of this method in dealing with real-world databases is yet to be studied.

A hybrid aggregation and compression technique for road network databases are discussed here [20]. It is a nice approach in vector data compression which is integrated within a geospatial query processing system. It uses line aggregation to reduce the number of relevant tuples and Huffman compression to achieve a multi-resolution compressed representation of a road network database. Problem with this technique is that it is complex system. Complexity associate with this system weeks its applicability in many simple and quick development approach.

3. OUR COMPRESSION TECHNIQUE

Analyzing different compression technique, their advantage, disadvantage and implementation complexity we could not accept them for our purpose. We have come to use a simple and fast enough compression technique - we find a reference point and for other points we just calculate the successive difference instead of full value and represent them as comma separated. Then outcome is compressed by gzip [21] and finally send to the client over the network. If we look at road vector databases in general and query results in particular, we see that the consecutive points (latitude and longitude) are very near to each other. Sending full, say 8 digit long point (latitude and longitude) is redundant. Let us consider 5 consecutive points' latitudes are 27.312312,27.312316, 27.312323,27.312325,27.312328. Sending these 5 points' latitudes require 40 digits to send but we can send this information segment with 27.312312,4,7,2,3 without any difficulty, which means only 12 digits are enough for sending the information segment. Hence we can eliminate a huge amount of redundant data. This technique is very easy and straightforward. Again gzip is based on the DEFLATE algorithm [22], which is a combination of LZ77 [23] and Huffman coding [24]. As a compression technique that significantly compresses such highly correlated data, we use gzip coding and it is a free software application. In fig. 1 we are showing actual and compressed files size for map data of Pretoria, South Africa.

4. IMPLEMENTATION

To make the development faster and cost effective we use different open/free components. Generic architecture of our implemented system is as follow –

At the bottom of this architecture is a database (PostGIS). Application servers (GeoServer and GeoWebCache) are in the middle. On the top of this architecture there is a user interface layer. The database and GeoServer interact via SQL (with Open Geospatial Consortium standard spatial

extensions). The applications servers and user interface layers interact via standard web encoding (JSON) over an HTTP transport. Again GeoServer and GeoWebCache interact over HTTP. GeoWebCache and user interface layers interact via standard web encoding (images) over HTTP. Components in this architecture works as follow:

- PostGIS: This database can answer spatial queries as well as standard attribute queries. It is certified as compliant with the OGC "Simple Features for SQL" specification. PostGIS is an extension to the PostgreSQL object-relational database system which allows GIS (Geographic Information Systems) objects to be stored in the database. PostGIS includes support for GiST-based R-Tree spatial indexes, and functions for analysis and processing of GIS objects [25]-[27]. In addition, PostGIS adds types, functions and indexes to support the storage, management, and analysis of geospatial objects: points, line-strings, polygons, multi-points, multi-line-strings, multi-polygons and geometry collections. As a spatial database, PostGIS can store very large contiguous areas of spatial data, and provide read/write random access to that data. This is an improvement over old file-based management structures that were restricted by file-size limitations and the need to lock the whole files during write operations. The spatial SQL functions available in PostGIS make analyses possible. Complete manual for PostGIS is given here [26], [27]. We can summarize functionality of PostGIS extension as follows –
  - It adds a “geometry” data type to the usual database types (e.g. “varchar”, “char”, “integer”, “date”, etc).
  - It adds new functions that take in the “geometry” type and provide useful information back (e.g. ST_Distance (geometry, geometry), ST_Area (geometry), ST_Length (geometry), ST_Intersects (geometry, geometry), etc).
  - It adds an indexing mechanism to allow queries with spatial restrictions (“within this bounding box”) to return records very quickly from large data tables.

  The core functionalities of a spatial database are easy to list: types, functions, and indexes. What is impressive is how much spatial processing can be done inside the database once those simple capabilities are present: overlay analyses, re-projections, massive seamless spatial tables, proximity searches, compound spatial/attribute filters, and much more.
- Geoserver: This map/feature server provides standardized web access to underlying GIS data source [5]. GeoServer provides an HTTP access method for geospatial objects and queries on those objects. GeoServer presents spatial data (tables in a database) as feature collections, and allows HTTP clients to perform operations on those collections:
  - Render them to an image, as an attractive cartography product.
  - Apply a logical filter to them and retrieve a subset, or a summary.
  - Retrieve them in multiple formats (KML, GML, GeoJSON).

  A document with comprehensive guide to all aspects of using GeoServer which covers topics from initial installation to advanced features is available here [28].
- GeoWebCache: It is tile server and can intelligently store and serve map tiles using standard web protocols for requests and responses [25]. Like GeoServer, GeoWebCache is a protocol gateway. GeoWebCache sits between tiled mapping components (like OpenLayers, Google Maps and Microsoft Virtual Earth) and rendering engine in GeoServer. Tiled map components generate a large number of parallel requests for map tiles, and the tiles always have the same bounds, so they are prime candidates for caching. GeoWebCache receives tile requests, checks its internal cache to see if it already has a copy of the response, returns it if it does, or delegates to the rendering engine (GeoServer) if it does not. GeoWebCache documentation is available here [29].

Client-side: Client in general has two main functionalities: It enables the user to specify his/her query and corresponding parameters and more importantly processes and renders the query results back to the user. We implement three different steps on the client side that enable processing and displaying the data: gzip decompression, reconstruction and rendering. These steps together, convert the compressed data received from the server to a meaningful and displayable format. Two different types of clients were used in our system: Heavyweight client and Lightweight client. In a heavyweight or slow client, besides the resulting vector data, additional data such as raster or satellite images are sent and displayed

on the client. In a lightweight or fast client, only the desired vector data and nothing more are sent and rendered on the client.

5. Conclusion

Transmitted raster image is only a visual representation of the geospatial data and hence does not include the geometric objects and corresponding metadata. Client cannot do any interaction with raster image but many applications such as navigation requires farther interaction with query result. A solution to this problem is sending the original query result in vector data format to the client to enable further interaction with it and to preserve the metadata. In that case it is important to compress query result so that we can send them to client side very quickly. Analyzing different compression technique, their advantage, disadvantage and implementation complexity we have come to the decision that some compression technique do not make the response fast enough and some make response faster but their complexity to implement makes their

References

[1] Online at http://all-things-spatial.blogspot.com/2009/06/ingenuity-of-google-map-architecture.html accessed December 31, 2011

[2] Online at http://maps.yahoo.com accessed January 06, 2012

[3] Cai Y, Stumpf R,Wynne T, TomlinsonM, Chung DSH, Boutonnier X, IhmigM, Franco R, Bauernfeind N (2007), "Visual transformation for interactive spatiotemporal data mining", Knowl Inf Syst 13(2):119–142 ISSN 0219-1377. doi:10.1007/s10115-007-0075-5

[4] Shahabi C, Kolahdouzan MR, Safar M (2004), "Alternative strategies for performing spatial joins on web Sources", Knowl Inf Syst 6(3):290–314. ISSN 0219-1377. doi:10.1007/s10115-003-0104-y

[5] Akimov A, Kolesnikov A, Fränti P (2004), "Reference line approach for vector data compression", *ICIP*, pp 1891–1894

[6] Shekhar S, Huang Y, Djugash J, Zhou C (2002), "Vector map compression: a clustering approach", *Voisard A, Chen SC (eds) ACM-GIS, ACM*, pp 74–80 ISBN 1-58113-591-2

[7] Zhu Q, Yao X, Huang D, Zhang Y (2002), "An efficient data management approach for large cybercity gis", *ISPRS*

[8] Ai T, Li Z, Liu Y (2003), "Progressive transmission of vector data based on changes accumulation model", SDH, Leicester, *Springer*, Berlin, pp 85–96

[9] Bertolotto M, Egenhofer MJ (2001), "Progressive transmission of vector map data over the world wide Web". *Geoinformatica* 5(4):345–373 URL http://citeseer.ist.psu.edu/bertolotto01progressive.html

[10] Bertolotto M, ZhouM (2007), "Efficient line simplification for web-mapping", *International journal of web engineering and technology*, special issue on web and wireless. GIS 3(2):139–156

[11] Buttenfield B (2002), "Transmitting vector geospatial data across the internet", In: *GIScience '02: proceedings of the 2nd international conference on geographic information science*, London, UK, Springer, Heidelberg, pp 51–64. ISBN 3-540-44253-7

[12] Douglas DH, Peucker TK (1973), "Algorithms for the reduction of the number of points required to represent a digitized line or its caricature", *Can Cartogr* 10(2):112–122

[13] Han Q, Bertolotto M (2004), "A multi-level data structure for vector maps", In: *GIS '04: proceedings of the 12th annual ACM international workshop on geographic information systems*, New York, NY, USA, ACM Press, pp 214–221 ISBN 1-58113-979-9. doi:10.1145/1032222.1032254

[14] Paiva AC, da Silva ED, Leite FL Jr, de Souza Baptista C (2004), "A multiresolution approach for internet gis applications", *In: DEXA Workshops, IEEE Computer Society*, pp 809–813 ISBN 0-7695-2195-9

[15] Persson J (2004), "Streaming of compressed multi-resolution geographic vector data",

Geoinformatics, Sweden.

[16] Puppo E, Dettori G (1995), "Towards a formal model for multi-resolution spatial maps", In: *Egenhofer MJ, Herring JR* (eds) SSD, volume 951 of Lecture Notes in Computer Science, Springer, Heidelberg, pp 152–169. ISBN 3-540-60159-7

[17] Saalfeld A (1999), "Topologically consistent line simplification with the douglas-peucker algorithm", Cartogr Geogr Inf Sci 26(1):7–17

[18] Wu ST, Márquez MRG (2003), "A non-self-intersection douglas-peucker algorithm", In: *SIBGRAPI, IEEE Computer Society*, pp 60–66. ISBN 0-7695-2032-4

[19] Zhou M, Bertolotto M (2005), "Efficiently generating multiple representations for web mapping", In: *Li K-J, Vangenot C (eds) W2GIS*, volume 3833 of Lecture Notes in Computer Science, Springer, Heidelberg, pp 54–65. ISBN 3-540-30848-2

[20] Ali Khoshgozaran, Ali Khodaei, Mehdi Sharifzadeh, Cyrus Shahabi (2008), "A hybrid aggregation and compression technique for road network databases", *Knowledge and Information Systems*, Volume 17, Issue 3, Pages 265-286

[21] Online at http://en.wikipedia.org/wiki/Gzip accessed December 15, 2011

[22] Online at http://en.wikipedia.org/wiki/DEFLATE accessed January 04, 2012

[23] Online at http://en.wikipedia.org/wiki/LZ77_and_LZ78 accessed January 04, 2012

[24] Online at http://en.wikipedia.org/wiki/Huffman_coding accessed January 04, 2012

[25] Online at http://opengeo.org/publications/opengeo-architecture/ accessed

[26] Ramsey, P., Refractions Research Inc, "PostGIS Manual". Online at http://www.dcc.fc.up.pt/~michel/TABD/postgis.pdf accessed January 06, 2012.

[27] Ramsey, P., Refractions Research Inc, "INTRODUCTION TO POSTGIS", Online at http://2007.foss4g.org/workshops/W-04/PostGIS%20Workshop.doc. accessed October1, 2011

[28] Online at http://docs.geoserver.org/1.7.x/en/user/ accessed October1, 2009

[29] Online at http://geowebcache.org/trac/wiki/Documentation accessed October1, 2009

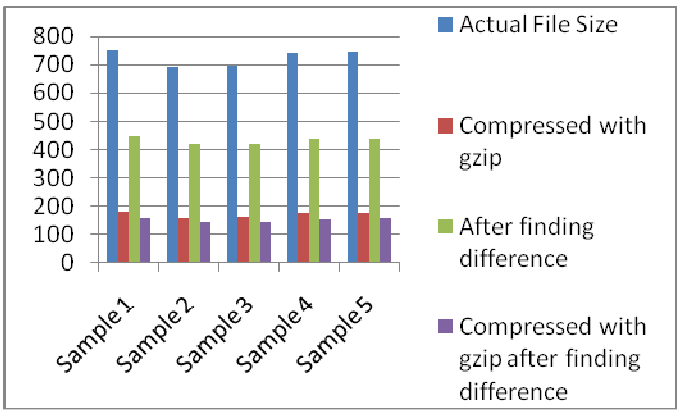

Fig.1:

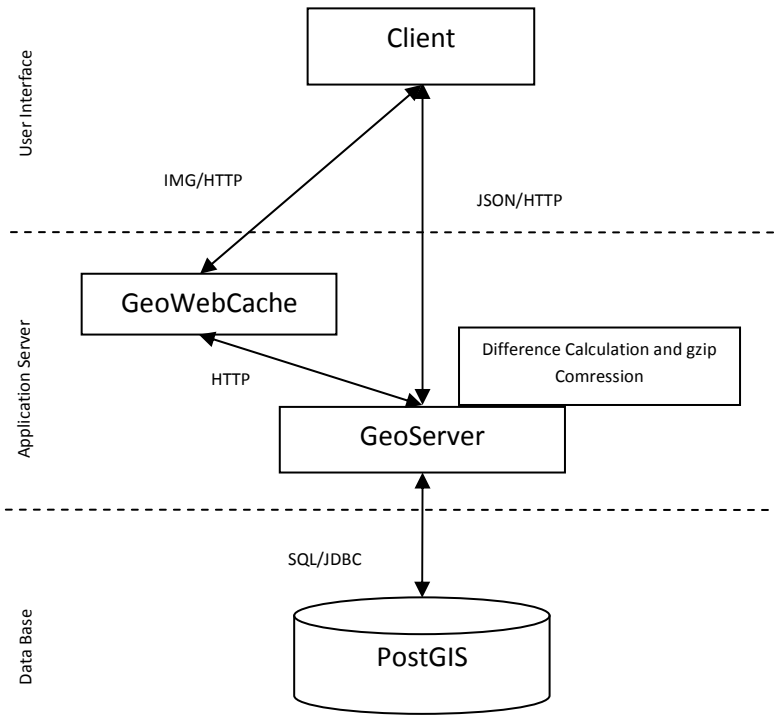

Fig. 2: Architecture of our implemented web map.

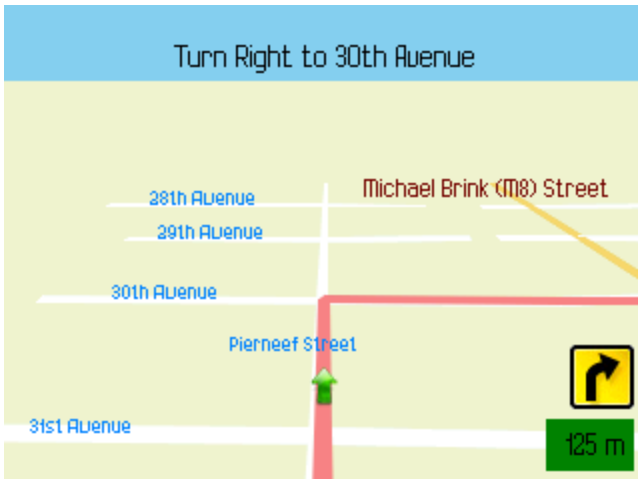

Fig 3: Navigation in J2ME client (shot 1).

Fig 4: Navigation in J2ME client (shot 2).