A PREFETCHING METHOD FOR INTERACTIVE WEB GIS APPLICATIONS

A THESIS SUBMITTED TO
THE GRADUATE SCHOOL OF NATURAL AND APPLIED SCIENCES
OF
MIDDLE EAST TECHNICAL UNIVERSITY

BY

SERDAR YEŞİLMURAT

IN PARTIAL FULFILLMENT OF THE REQUIREMENTS
FOR
THE DEGREE OF MASTER OF SCIENCE
IN
COMPUTER ENGINEERING

FEBRUARY 2010

Approval of the thesis:

A PREFETCHING METHOD FOR INTERACTIVE WEB GIS APPLICATIONS

submitted by **SERDAR YEŞILMURAT** in partial fulfillment of the requirements for the degree of **Master of Science in Computer Engineering Department**, **Middle East Technical University** by,

Prof. Dr. Canan Özgen _____
Dean, Graduate School of **Natural and Applied Sciences**

Prof. Dr. Müslim Bozyiğit _____
Head of Department, **Computer Engineering**

Assoc. Prof. Dr. Veysi İşler _____
Supervisor, **Computer Engineering Dept., METU**

Examining Committee Members:

Prof. Dr. Özgür Ulusoy _____
Computer Engineering Dept., Bilkent University

Assoc. Prof. Dr. Veysi İşler _____
Computer Engineering Dept., METU

Assoc. Prof. Dr. Şebnem Düzgün _____
Mining Enginering Dept., METU

Assoc. Prof. Dr. Pınar Karagöz Şenkul _____
Computer Engineering Dept., METU

Asst. Prof. Dr. Tolga Can _____
Computer Engineering Dept., METU

Date: 03/02/2010

I hereby declare that all information in this document has been obtained and presented in accordance with academic rules and ethical conduct. I also declare that, as required by these rules and conduct, I have fully cited and referenced all material and results that are not original to this work.

Name, Last Name : Serdar Yeşilmurat

Signature :

ABSTRACT

A PREFETCHING METHOD FOR INTERACTIVE WEB GIS APPLICATIONS

Serdar Yeşilmurat
M.S., Department of Computer Engineering
Supervisor: Assoc. Prof. Dr. Veysi İşler

February 2010, 120 pages

A Web GIS system has a major issue of serving the map data to the client applications. Since most of the GIS services provide their geospatial data as basic image formats like PNG and JPEG, constructing those images and transferring them over the internet are costly operations. To enhance this inefficient process, various approaches are offered. Caching the responses of the requests on the client side is the most commonly implemented solution. However, this method is not adequate by itself. Besides caching the responses, predicting the next possible requests of the client and updating the cache with the responses for those requests provide a remarkable performance improvement. This procedure is called “prefetching”. Via prefetching, caching mechanisms can be used more effectively and efficiently. This study proposes a prefetching algorithm called Retrospective Adaptive Prefetch (RAP). The algorithm is constructed over a heuristic method that takes the former actions of the user into consideration. This method reduces the user-perceived response time and improves users’ navigation efficiency. The caching mechanism developed takes the memory capacity of the client machine into consideration to adjust the cache capacity by default. Otherwise, cache size can be configured manually. RAP is compared with 4 other methods. According to the experiments, this study shows that RAP provides better performance enhancements than the other compared methods.

Keywords: WEB GIS, Cache, Prefetching, Performance

ÖZ

ETKİLEŞİMLİ WEB CBS UYGULAMALARI İÇİN ÖN YÜKLEME YÖNTEMİ

Serdar Yeşilmurat
Yüksek Lisans, Bilgisayar Mühendisliği Bölümü
Tez Yöneticisi: Doç. Dr. Veysi İşler

Şubat 2010, 120 sayfa

Web CBS uygulamaları için harita verilerini istemci uygulamalarına sunmak çok önemli bir sorun teşkil etmektedir. Birçok CBS servisi coğrafi verilerini PNG ve JPEG gibi temel resim formatlarında sağladıklarından dolayı bu resimleri üretmek ve internet üzerinden iletmek maliyetli işlemlerdir. Bu randımansız süreci iyileştirmek için çeşitli çözüm önerileri sunulmuştur. Kullanıcı isteklerine verilen cevapları istemci tarafında saklamak (önbelleğe almak) en yaygın çözüm olarak karşımıza çıkmaktadır. Ancak, bu yöntem de tek başına yeterli değildir. Verilen cevapları saklamanın yanında bu yanıtları kullanarak sonraki olası istekleri tahmin edebilmek ve önbelleği bu isteklerin cevapları ile güncellemek kayda değer bir performans iyileştirmesi sağlayacaktır. Bu yönteme "önyükleme" denilmektedir. Önyükleme sayesinde önbellekte saklama mekanizmaları daha etkili ve verimli bir şekilde kullanabilmektedir. Bu çalışma kullanıcının önceki işlemlerini dikkate alan deneyimsel bir yöntem üzerine kurulu bir önyükleme algoritması önermektedir. Bu yöntem uygulamanın kullanıcı isteklerine verdiği tepki süresini azaltmakta ve kullanıcının harita üzerinde gezinmesini daha etkin hale getirmektedir. Geliştirilen mekanizma önbelleğin boyutunu (elle ayarlanabilir olmakla beraber) varsayılan olarak istemci makinesinin belleğini dikkate alarak ayarlamaktadır. Önerilen algoritma 4 farklı metod ile karşılaştırıldı. Deneylerin sonucuna göre bu çalışmada sunulan metodun diğer yöntemlere göre daha iyi performans sağladığı görülmektedir.

Anahtar Kelimeler: WEB CBS, Önbellek, Önyükleme, Performans

To My Beloved Family,

ACKNOWLEDGEMENTS

The author of this study presents his gratefulness to:

… his supervisor Assoc.Prof. Dr. Veysi İŞLER for his positive criticism and guidance;

… his colleagues whom he will be always feeling privileged to work with;

… his company; MilSOFT Software Technologies Inc. for assigning him to a GIS project called PiriMap, and therefore conducing to his selection of thesis topic;

… especially his parents for their support, tolerance, patience and always being there for him.

TABLE OF CONTENTS

APPENDICES

LIST OF FIGURES

FIGURES

LIST OF TABLES

TABLES

LIST OF ABBREVIATIONS

AVG	Average
BBOX	Bounding Box
BXML	Binary Extensible Markup Language
CPU	Central Processing Unit
DBMS	Database Management System
DGN	Design format
DBMS	Database Management System
GML	Geographic Markup Language
HCBP	Hilbert Curve Based Prefetch
JPL	Jet Propulsion Laboratory
JVM	Java Virtual Machine
LRU	Least Recently Used
OGC	Open Geospatial Consortium
PKM	previous-k-movements
RAP	Retrospective Adaptive Prefetch
SP	Simple Prefetch
SRS	Spatial Reference System
STD DEV	Standard Deviation
TC	Tile Cache
URL	Uniform Resource Locator
VAR	Variance
WMS	Web Map Service

CHAPTER 1

INTRODUCTION

With the growth in internet usage, various beneficial data are presented on internet and "Geographic data" is certainly one of the most demanded information among all. Geographical Information Systems (GIS) introduce methods and environments to visualize, manipulate, and analyze these geospatial data and Web GIS provides a way for utilizing these web-based geographic data via standard web service definitions.

The nature of the geographical applications requires seamless integration and sharing of spatial data from a variety of providers. To solve the interoperability problems, the Open Geospatial Consortium (OGC®) has introduced standards for the Web GIS services. OGC® is an international industry consortium which defines specifications to provide interoperable solutions that "geo-enable" the web. It has variety of contributors from different areas such as private industry and academia to create open and extensible software application programming interfaces for GIS.

Several standards and discussion papers are presented about the web services utilized for transferring geospatial data over Web. Web Feature Service (WFS) defines interfaces for data access and manipulation operations on geographic features over Web [1]. Geospatial data processed by this service are in vector format. Web Coverage Service (WCS) retrieves the geospatial data as "coverages" which are described as digital geospatial information representing space-varying phenomena [2]. Besides these two standards, OGC® published another web service standard in Web GIS domain called Web Map Service (WMS) [3]. WMS is an international specification introduced by this consortium for serving and consuming dynamic maps on the Web. Throughout this study, two WMS servers are used as the services to provide geospatial data. One server is located in same machine with the test software (local) and the other is located on Web (remote).

Via WMS, a client can request spatial data of a location on earth in different layers, styles, dimensions and data formats. For example, a WMS client may want to obtain the 300x300 pixel raster map of the city s(he) is living with the street labels on it and labels are displayed in red. Raster map and street labels are produced as different layers by the WMS server. “Red” defines the style and “300x300 pixels” is the dimension of the map given as a part of the request. Also, the client may select different image formats supported by the WMS server, such as JPEG, PNG or GIF. Namely, there are many combinations for the client to generate a WMS request to fulfill his/her requisitions of spatial information. More combinations mean more different types of requests that should be responded by the WMS services.

In a Web GIS, the granularity of geographic data can be either a whole map or a small fragment of it which is called a tile. Loading an entire map at once and navigating on it is acceptable only if the size of the data is small. Tiling is an alternative; the map is partitioned into equally sized segments and served by tiles. To accommodate the client’s navigation efficiently, common-used tiles are cached in advance and while the client is viewing one of them, the next candidate tiles are prefetched. So, the question is how these candidate tiles are predicted.

In web, fast responses depend on less process on server side. Less process is basically achieved by the caching and prefetching policies. The motivation behind this study is to develop a prefetching method for increasing the performance of Web GIS and responding the client requests in an acceptable period of time in a reasonable way.

The map to be displayed is represented as grids where each cell is a tile and passing from one cell to another is considered as a transition among these tiles. A transition defines a cost which depends on loading a data which is not already in the cache. Namely, a transition neglects fetching the preexisting data. Navigation over a map contains two main operations; moving from one tile to another and changing the zoom level. So, the grid distribution over the map shall take these two options into

consideration. In other words, which tile will be prefetched is determined by predicting the behaviors of the client and this behavior can be one of those two navigation factors. Each grid implies the next move of the client over the map.

Since, there is no standard way of predicting the user's behavior while navigating on the map, exact solution for prefetching may never exist. This is where heuristics step in. Finding the best approximate future moves is a general issue in global optimization problem. In this study, by analyzing and ranking the former moves of the client, the next possible tiles that will be requested from the server are prefetched beforehand. In other words, former states entirely capture all the information that could influence the future progress. Instead of a deterministic process, future states are reached through a probabilistic process. In this context, by implementing a heuristic evaluation method, this study aims to find the best probable data set to be prefetched.

The remainder of the document is organized as follows. Chapter 2 surveys related works on prefetching and other performance optimization techniques. Chapter 3 explains the proposed method in detail and compares it with the 2 most commonly used methods by stating the advantages and disadvantages of all. In Chapter 4, the test bed developed for executing the experiments and obtaining the simulation results is explained. Chapter 5 summarizes the experimental results. Finally, Chapter 6 gives the conclusions and future work.

CHAPTER 2

RELATED WORK

In the evolution of Web GIS, despite the remarkable improvements, there are several issues yet to be resolved. The first issue is "considering the Web GIS only as Internet mapping". Basic functions of an interactive map are panning, zooming and querying. However; there are many other features GIS can offer; such as, GIS analysis like buffering, network analysis etc. Evolution of Web GIS on this concept is a forthcoming challenge. Another issue is security. Java applets, ActiveX Controls and plug-ins are downloaded from the Internet and executed in the local machine of the client. The security of these applications shall be assured. The third issue is charging of these services. The information published on the Internet can be charged or served as free of charge. The choice between these two options shall be made. If the user will be charged, the price amount and how to charge this fee shall be decided. Besides all the problems above, the most important issue of the Web GIS is performance. Since the GIS data are large in volume, it takes long time to transfer them over internet. This introduces a big problem especially for the machines with slow internet connections. The issue of slow performance can be solved in two ways:

- increasing the speed of Internet connection
- developing more efficient Web GIS programs

The speed of Internet connection is improving with faster modem and faster communication connections. This fast Internet connection will make the current GIS data transfer on the Internet faster. However, relying on the improvement in Internet connection speed should not be adequate. Because, designing efficient Web GIS applications will make it feasible to execute these programs even on slower connections. Today, there are remarkable number of studies and researches about how to improve the performance and accessibility of Web GIS.

2.1 General Solutions

OGC®, the leading standardization institute over GIS, introduced some standard services to provide interoperability on distributing GIS data. WMS [3] is one of these standards and as expressed in the introduction section, its main goal is providing flexibility of requesting arbitrary number of map layers in an arbitrary bounding box with different styles. However, every good thing comes with its price. WMS is expected to generate geographic map image on the fly as it is demanded. Because of this reason, it does not scale well. Even for a "single request at a time" condition, it takes a few seconds to serve the cartographic layers. In order to solve this issue, OGC® defined a WMS extension called "Tiled WMS" [4] which utilizes tiling. Tiles are pre-rendered images with specific bounding boxes and scales. By using this technique, a WMS service can serve the requested map image as tiles without making any extra process on the server side. This mechanism is considered as highly scalable and illustrated by Google Maps. The operation defined by Tiled WMS is called *GetTile* and it is used as an alternative to the original WMS request, *GetMap*. The main disadvantage of *GetTile* request is loosing the flexibility of requesting any location on earth by giving a bounding box. Also, the client should know the corresponding row and column number of the tile s(he) requests. However this is not a big issue, because Tiled WMS has another operation called *DescribeTile* that provides a description of each tile served by the WMS service. Nevertheless, *GetMap* request still exists for the clients who require flexibility over scalability.

There is another OGC® originated study on improving the performance of Web GIS. But this time, it is not a standard but a "best practice". Best practice documents are considered as non-mature suggestions over an earlier published specification. This best practice document is an extension over another existing OGC® standard. This interoperability standard defines a feature-encoding format called GML (Geography Markup Language) [5]. GML is the suggested format by OGC® for transferring geographic features over Internet. There are major problems of this format. First one arises from the structure of the GML format. Since GML is an XML-based format, it contains redundant bulky data that slows down the data transfer. Another problem is

the cost of traversing the XML structure and the conversion of text-based numerical coordinate values. Compressing the GML data with general methods like GZIP can increase the transfer speed. However, it does not eliminate the traversing and conversion costs. To address these issues, OGC® defined a new data transfer format called BXML (Binary XML) encoding [6]. BXML encoding mirrors the in-memory XML node representation as a sequence of node-equivalent "tokens". The elements of an XML document are defined by these tokens and bulky XML tag representation is avoided by the integer indexing method. Also, assigning the structure size of the structures in advance at the head of their byte sequence helps the applications to parse the data efficiently. Since the coordinate values kept in BXML structure, they can be read directly into an array in memory. By this way, costly parsing and conversion operations are not needed anymore. Namely, all the disadvantages of GML are eliminated by the BXML encoding.

Jay Ratcliff and Kevin Shaw proposed some design strategies to improve performance of GIS Web Services [7]. They discussed the importance of four design-decision issues in their study. Firstly, they compared the synchronous services with asynchronous ones. A GIS web service client is often another application or a cascading map server. Since the cascading web servers prefer to fetch the data in parallel, asynchronous service approach is needed. By providing asynchronous services, the client is not interrupted by the computation time on the server side and s(he) can deal with other operations. Another design strategy is making a decision between fine-grained services and coarse-grained services. Although the object-oriented design expects you to choose fine-grained services for flexibility, for Web GIS services, coarse-grained services should be preferred. The reason for that choice is obvious. The communication overhead caused by the requests for assembling a map that compose of thousand of features is not acceptable. However, if one (or few) coarse-grained service handles the request, the map can be served in a shorter period of time. Data transmission is another issue in Web GIS design. Format of the data transferred over web is crucial for the performance. Ratcliff and Shaw are in favor of binary formats and give DGN as an example for the file format. They mention that for the same spatial information, the sizes of GML documents are often ten times

larger than DGN files. The main disadvantage of binary formats is the interoperability issue. Because, GML is recognized as a data transfer format for geographic data and most of the Web GIS applications obey this format. On the other hand, binary formats are generally used for data transmission between the services in intranet networks. But, as I mentioned earlier, if BXML becomes a standard specification besides GML, this interoperability issue can be resolved.

In another study [8], some optimization techniques are given to improve the overall performance of Web GIS applications. These techniques are "data simplification, relative coordinates, static maps, multiresolution, compression and on-demand loading". *Data simplification* process identifies the map information that can be omitted and removed before the transmission. Since map coordinates are set with high precisions, the number of points a geographic feature contain to represent itself is too large. For smaller scales, most of these points can be removed without disrupting the general shape of the feature. This can be achieved by arranging the precision of the simplification. However, it should be considered that more performance means less accuracy and quality. In figure below, it can be seen that the accuracy of the geographic feature is reduced as the precision is chosen low.

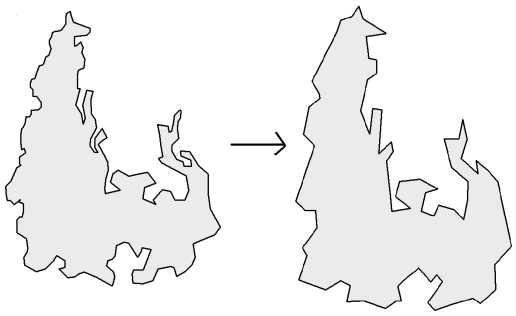

Figure 1: The appearance of a lake feature after simplification process

Relative coordinates express the coordinates of each point in a polygon or polyline according to the previous point. Absolute coordinates specify each point independently according to a fixed coordinate system. Via relative coordinates, the volume of the data transmitted for a geographic feature reduces. The disadvantage of this approach is changing the rendering strategy of a geographic feature. Instead of

drawing the geometry of the feature directly, in this approach, translating each point of the geometry according the previous point to obtain the absolute coordinates is needed. According to this study, the trade-off between the absolute and relative coordinates is in favor of relative coordinates. In *static map* approach, loading the map data statically as soon as they are needed is recommended, because Web GIS applications generally store their data in a spatial database system. Establishing a connection and loading the data is a time consuming process. By *multiresolution*, each map data are represented in different levels of detail. By this way, for instance, the streets will not appear when the zoom level focuses on a country, hence the performance is improved. As mentioned earlier, *compression* is an important factor to reduce the size of the data transmitted and it consequently improves the overall performance. *On-demand loading* is defined as the creation of the map according to the type of user request. If some layers are not inside the user's area of interest, those layers are filtered and not displayed. Also, the scale of the map may not be appropriate for some map layers. To avoid the map pollution, some map layers are discarded, because displaying those layers is not meaningful. A complex vector map that represents the planning of a city should not be displayed in the country zoom levels.

Walker, Pham and Maeder constructed a framework over Bayesian Network utilized for keeping the probabilistic relationships between the task analysis and datasets [9]. Task analysis is basically the operations performed by the GIS user. Datasets include the spatial, non-spatial and relational information used during these tasks. A Bayesian Network is generated with all these data and relationships between them and only the certain datasets are loaded according to the query made by the user. Namely, instead of loading the entire map data, the layers that fulfill the user requirements are fetched from the GIS server. Also, the learning process for a Bayesian Network is dynamic update of probabilities. The user may intervene to the process by rejecting some of the selected layers or adding layers manually. Thus, the casual probability of a layer as a result of the user query is updated according to these interventions for the future analysis.

Haixia Zhao and Ben Schneiderman developed a light-weight Java Applet called YMap to introduce the improvements they proposed for Web GIS applications [10]. They focused on the problem of slowness in rendering the map image on client-side and proposed an image-based technique to overcome this problem. Their solution is specific to Web choropleth maps and instead of vector data, the map data transferred over network is in compressed GIF format. A geographic data in a choropleth map is represented with a single color. So, from the color of the pixel that the user's mouse is over, the geo-referencing data can be extracted from the database on server side. By this way, the map displayed on browser can be used interactively. Also, in this technique, the map is broken down into equally sized smaller segments and when the user performs a pan or zooming operation, only the segments inside the current view are requested instead of the whole map data. According to their comparisons with a Web GIS package that uses vector geographic data, the loading and rendering performance of YMap came out better. As a further improvement, they recommended two other methods too:

- caching different levels of the map data as they are requested
- prefetching the next possible segment to be loaded by using a separate thread

The amount of geospatial data is increasing constantly. Earth observation satellites send large amounts of data everyday. This introduces the problem of development of computer systems utilized for storing, managing and distribution of these huge data. Coddington, Havick and James implemented a prototype system for providing access to this data and services to process the data [11]. The active digital library developed enables retrieval of remote data and processing the remote data to construct the map data the user is interested in. In many research and decision support applications, this Web-based distributed GIS system infrastructure seems to have great promise.

Neville Churcher proposed a method for displaying always the whole extent of a geographic area on screen [12]. By implementing a fisheye view technique, even though the focus point and/or the zoom level on the area of interest changes, all the geospatial objects in the map area fits inside the view extent. This method is based on distortion oriented presentation. Namely, the objects around the focus point are

displayed larger than their original sizes. On the other hand, to fit the whole map extent inside the view extent; the objects away from the focus point are displayed smaller then their original sizes. By this way, the objects around the mouse location can be observed in great details. This method provides a feasible workaround for isolating the geographic information in interest from the geospatial object clutter.

2.2 Prefetching Methods in GIS Domain

Dong-Joo Park and Hyoung-Joo Kim present a new prefetching policy for retrieving large objects in Web GIS applications [13]. They make an assumption on how the user accesses the geographic objects on the map. For instance, instead of accessing the objects in the Web browser randomly, it is very likely that the user chooses the neighbor objects around a certain central object which is called "callback object". Namely, the access pattern of the user is determined according to the "spatial locality". Under this assumption, they propose a Hilbert Curve based prefetch algorithm. For the efficiency of the prefetch policies, they saw the necessity of employing a clustering method to reflect the spatial locality. Since the Hilbert Curve is one of the most efficient clustering methods in literature, it seems wise to use this method for prefetching. Hilbert Curve is used for selecting a set of candidates based on the current callback object. Such candidates will have more probability to hit the client cache when the next callback object is requested. In this approach, map is divided by the Hilbert Curve and each cell is assigned with the values of that Hilbert Curve as shown in the figure below:

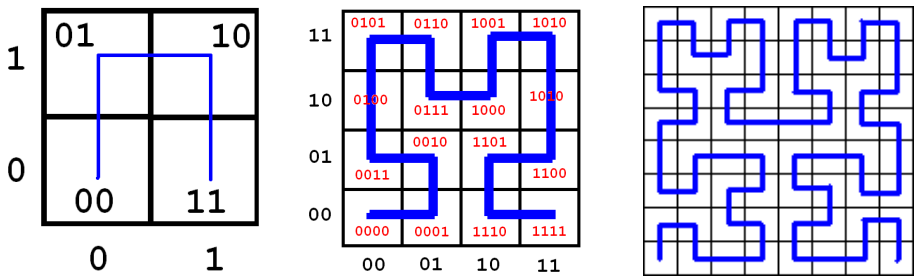

Figure 2: The first three steps of Hilbert Curve

They defined an array of Hilbert Curve values. Each object in the Hilbert Curve is placed in this array with their Hilbert Curve values. The candidates are selected from the left and right side of the callback object in the array. The number of the candidates is determined statically or dynamically. Let's assume that the location of the callback object is "*i*" and callback object is represented as "*O*". In static way, a value called "window size" is chosen and if we say window size is "7", the objects O_{i-3} ... O_{i+3} are selected as candidate objects. In dynamic approach, the window size is adjusted by the "spatial locality by distance" (SLD). The candidate objects are still selected from the left and right side of the callback object, but this time, instead of taking static number of objects, all the objects whose distance from the callback object is smaller than the SLD are chosen as the candidate objects. Since the number of candidates can vary as the SLD value changes, the window size is dynamic. Also, they defined a formula to determine an appropriate SLD value. Through the experimental results, they show that the performance of the dynamic scheme is close to the static scheme; on the other hand, the network traffic of the former is lower than the latter.

Dong Ho Lee et al. propose two prefetching techniques [14]. One of these methods is probability-based and it assumes that the location of the tile is significant while predicting the next navigation. According to this assumption, if the tile is located near to the upper border of the view extent, then the user is likely to navigate to an upper tile than the lower one. This technique takes the zooming levels of the current view extent into consideration too. The probabilities of all navigations are calculated and top "*t*" (number of tiles to prefetch) tiles with the highest transition probabilities are selected. However, the reasoning behind the probability ranking logic is not given. Especially, the next zooming move cannot be predicted according to the location of the current tile just like the neighborhood tiles.

The second technique proposed can be considered as more heuristic than the previous method. It is called *previous-k-movement* approach. In this method, rather than the current position, former actions of the user determines the next movement.

A Neighbor Selection Markov chain (NSMC) is built for obtaining the neighbor selection probabilities that can be applied to a general tile. A state on NSMC presents a sequence of direction vectors that gives the tile selection history. A direction vector denotes a move from one tile to another. The direction vector on a 2-dimensional space is shown below:

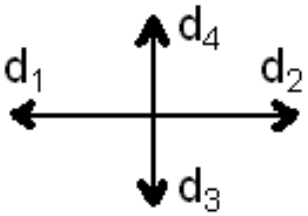

Figure 3: Direction vectors at dimension 2

According to this representation, moving to east and then north is expressed as (d2, d4). An edge on NSMC is the probability to reach the next tile from the current tile. By using this NSMC, predicting the next tile is deciding the next direction vector from the current state. To determine the probabilities, that study utilizes Hilbert Curve in their experiments. Transitions from one tile to another are presented on Hilbert Curve. Each pattern on Hilbert Curve keeps the history of a tile and the next tile to predict from the current tile is determined by locating the pattern on Hilbert Curve and choosing the tile that comes next on Hilbert Curve. The number of direction vectors that forms the pattern is configurable and taken as 3 in experiments.

The figure below shows the prediction of the next move from the current state on Hilbert Curve. The first state denotes that the user's past 3 moves contains direction vectors d4, d2 and d3 respectively. The pattern formed by these direction vectors stays on the left-bottom corner of the Hilbert Curve. Then, the next move is predicted as d2, the last direction vector in state 2. The states 3 and 4 give the next two predictions after that move if the prefetching of d2 is actually the tile the user will request.

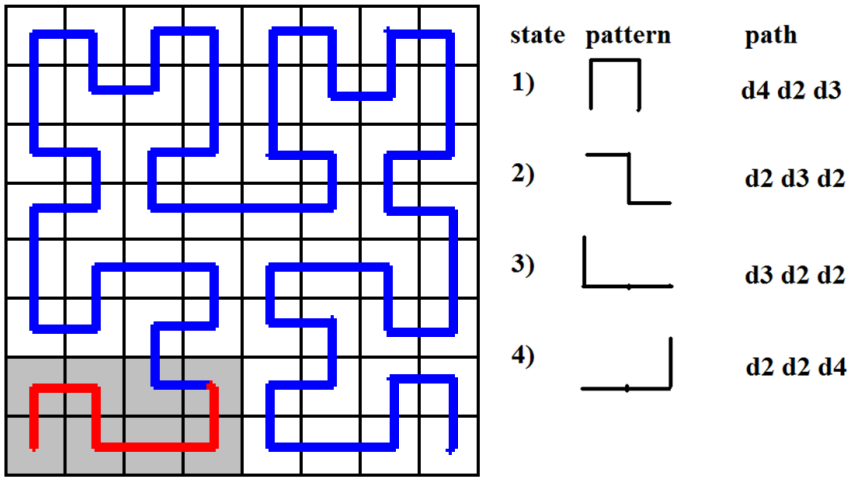

Figure 4: Tile prediction over Hilbert Curve for given patters

As it is seen, the usage of NSMC is limited to representing the states and transitions as a graph. After all, Markov Chain does not rely on previous states to predict the next state according to its definition. According to Markov Chain, the current state itself gives adequate information about the next state will be passed.

In this method, the probabilities of previous navigations are put on a Hilbert Curve and next tiles are predicted according to patterns on this curve. The main difference between this method and Hilbert Curve based prefetch algorithm [13] is; this method makes predictions by using the past *n* moves before a specific tile instead of just using that specific tile. Also, this method does not predict the zooming levels from the current tiles. Since, prefetching tiles on a zooming level is much more expensive than just prefetching the neighborhood tiles, not considering zooming levels in prefetching process is a deficiency of this method.

2.3 Prefetching Methods in Other Domains

Han et al proposed a prefetching method for improving the performance of navigational applications such as XML applications, GIS and CAD/CAM systems [15]. It is based on two notions; type-level access locality and type-level access

pattern. Type-level access locality means the types of object accessed by the applications repeatedly. Type-level access pattern is the pattern of objects that appear repeatedly. Their method reduces the number of fetches and therefore increases the performance of the application. Han, Loh and Whang examined the method proposed by Han et al [16] and offered an enhanced version that decreases the number of disk accesses as well. There are two important factors deciding the prefetching performance; the number of object fetches and the number of disk accesses in server. They claim that reducing the number of fetches does not provide a satisfactory performance improvement unless the number of disk accesses does not change. So, they proposed a method for minimizing the number of disk accesses. The method creates materialized views that reflect the type-level access patterns. These views are used by the algorithm for minimizing the number of disk accesses when prefetching the objects from the database. According to their experiments, the number of disk accesses is reduced by up to 33.0 times and performance is improved by up to 21.4 times.

The algorithm finds the best iterative pattern view among all the views for the type-level access patterns by computing the cost of each candidate view and finding out the one with the minimum cost. They define the cost of a view as the number of disk pages occupied by the view. Namely, the smaller page size means less cost. This prefetching method can be seen as an optimization for fetching data from database, but not exactly a prefetching method. Because, creating views is a known technique for dealing with a structure consists of data distributed to separate tables. This method provides a technique for constructing views for the navigational patterns and hence reducing the number of disk accesses via these views instead of database tables. However, this method may not be a proper prefetching solution for Web GIS applications. Most of the GIS data served on web are raw raster maps. The non-spatial attributes of these map data (such as city names, population etc…) are usually kept in database tables. However, prefetching mentioned in that study does not rely on non-spatial data. Han et al only considered navigational applications on Object-Relational DBMSs (ORDBMS). Non-spatial data can be stored on an ORDBMS, but the same thing is not valid for spatial data. The DBMS utilized for spatial data is

called Spatial DBMS and creating views on Spatial DBMSs is not mentioned in that study. Besides, Spatial DBMSs are not good candidates for creating views to use in that method, because of the method proposed does not handle spatial map data, but only non-spatial data. Nevertheless, reducing the disk accesses is a crucial optimization. Although it is not suitable for prefetching map data for web GIS applications, this method can be considered as a wise choice to optimize the performance of other navigational applications.

Handling the load coming to Web server is one of the most important issues in the current server architecture over web. To overcome this issue, dynamic load balancing is a fundamental need on the server side. Within the concept of a study [17], an agent is designed on the dispatcher of a Web Server cluster for optimizing connection to the Map Server. The purpose of this agent is predicting the load and selecting the most proper Web Server in a Web Server cluster. To achieve that, the load of each server must be computed. The formula for computing the load to select the best suited server is given as:

$$Total\_Load = (Wcpu \times cpu\_load) + (Wmem \times mem\_load) + (Wdisk \times disk\_usage) + (Wconn \times map\_server\_connection)$$

$$Wcpu + Wmem + Wdisk + Wconn = 1$$

In this formula, *map_server_connection* is defined to divide total connection between a Web Server cluster and Map Server by number of connections between each Web Server and Map Server. In that study, the weights are chosen as ¼, ¼, 0 and ½ respectively. According to the results, in a cluster with five servers, the agent provides better performance for Web GIS services than the previous normal Web GIS architecture. The traffic on each server is decreased significantly. However, we can say that this agent strategy will be more useful and efficient for the large networks with several Web Server clusters with a shared Map Server.

Another study provides a prefetching method specialized on interactive walkthrough applications [18]. A transition from one cell to other causes loading the data necessary to construct the potential view set. To avoid the stall during this process, a

prefetching method is developed. Within the scope of the prefetching, a graph representation of the view cells and transitions among them is generated. A general probabilistic heuristic algorithm called Simulated Annealing [19] is utilized to optimize prefetching the scene data over the generated graph.

OpenLayers (http://openlayers.org) is a JavaScript based library for displaying dynamic map data on map browser. By integrating any standard OGC WMS server, the map data in that server is retrieved as the user navigates on the browser window. The data are requested from the map server as tiles. Tiles obtained from the server combined together and form the view extent of the map the user currently navigating on. Via the tile mechanism, OpenLayers caches the tiles on client side. Also, it provides a simple prefetching mechanism. According to this prefetching method, tiles around the view extent are requested from the map server when the user requests the tiles inside the view extent. A parameter called *bufferSize* is used for determining the number of levels to prefetch. For instance, if *bufferSize* is 2, 2 tiles for each tile on the edge of the view extent are prefetched as shown in the figure below:

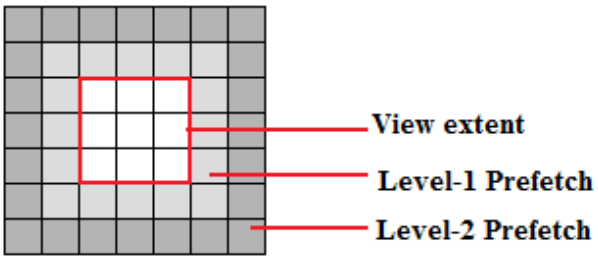

Figure 5: Tiles prefetched by OpenLayers when bufferSize is 2

CHAPTER 3

METHODOLOGY, CURRENT APPROACHES, AND THE PROPOSED PREFETCHING ALGORITHM

As mentioned in the previous sections, the basic issue of the Web GIS applications is the size of the data transferred over network. WMS can be considered as the standard interface definition of today's most popular Web GIS applications. In a WMS server, the data format presented as a response of a valid map request is a bitmap image. PNG is the most preferred image format, because it supports alpha-transparency which is not provided by JPEG image format. Also, the compression algorithm of PNG is more efficient than the other image formats that also support transparency (such as GIF). The size of the PNG images transferred over the network can be considered as acceptable with the help of compression. However, the client application that receives these bytes converts them into a format that the client application understands. For instance, a transparent PNG image with 300 x 300 pixels dimension can be compressed to 1% of the original size. But, the client application written in Java language has to convert these bytes into a `BufferedImage` instance to process. The raster representation of this PNG image in `BufferedImage` instance consumes 360000 (= 300 x 300 x 4) bytes in memory which is considered as a large scale memory consumption for an object instance in Java language.

In this study, "300 x 300 pixels" is chosen as the dimension of a tile requested from a WMS server. Most of the Web GIS client applications (such as Google Earth) combine and print the consecutive tiles to portray a map location on the screen. Namely, more than one tile is needed to represent a location on client screen.

The figure below displays the sample screenshot of the test application used in this study to simulate the user's behavior. In total 12 WMS requests have to be made to obtain the tiles needed for generating the view extent. In the client machine's

memory, these tiles consume nearly 4 MB of space. Also, for each tile request, a significant time delay occurs because of the construction of the tile image on server side and transferring the bytes of the resultant image over the network. From this point, memory consumption and time delays will be taken as the basic factors that affect the performance of a Web GIS application.

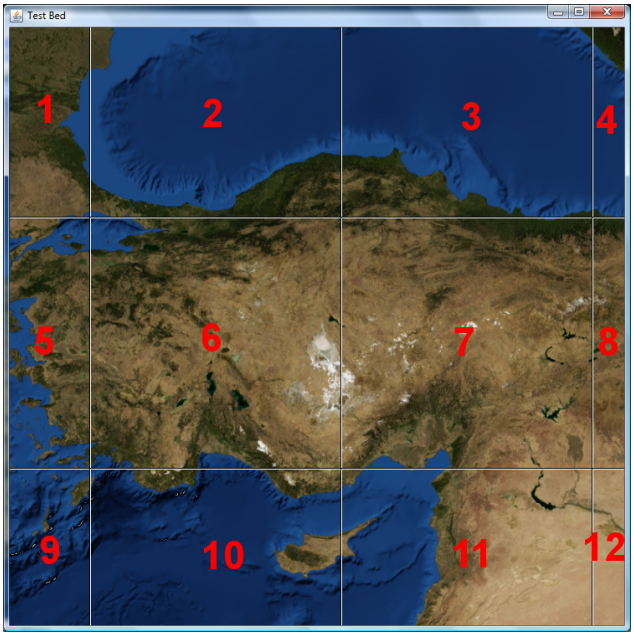

Figure 6: Tiles visible in view extent

This study compares the results of the proposed algorithm with two commonly used methods and also Hilbert Curve Based Prefetching (HCBP) Algorithm [13] and previous-k-movements (PKM) method [14] given in Related Work section. Besides HCBP and PKM, 3 methods are implemented as part of this study:

1. Caching the tiles requested from the WMS server: Tile Cache (TC)
2. Prefetching neighbor tiles of a requested tile: Simple Prefetching (SP)
3. Proposed algorithm: estimating the next possible moves of the client in heuristic fashion and prefetching the tiles that represent those moves: Retrospective Adaptive Prefetching (RAP)

3.1 Caching the tiles requested from the WMS server

The library that is used for developing the test application is "SwingX-WS" (https://swingx-ws.dev.java.net). The methodology used for handling the client requests by this library is the most commonly used strategy in Web GIS world. In this approach, each tile obtained from the WMS server is put in a cache. The cache does not contain any logic except cleaning itself when the memory reaches its limit. With the help of this cache, no request is made to the WMS server for the formerly visited tiles on the map. As a result of this optimization, the overhead of a multiple server request for the same tile is prevented.

For a client navigation that requests 12 tiles to construct the whole map image on screen (which is called view extent), 12 requests are made to the WMS server. The client application creates a separate thread for each tile request and executes those threads. As a design decision of the application, only 4 threads can run at the same time and when a thread ends its execution one of the other 8 threads is taken from the queue and executed as well. When the user navigates to another location on map before the execution of the entire 12 threads completes from the previous navigation, on the contrary of the expected behavior, the left threads are not removed from the thread queue and continue to be executed with the new threads created for the new tile requests.

The advantages of this method are:

- If the client wants to revisit the locations s(he) passed earlier, no requests are made to the WMS server and the desired tiles are retrieved from the cache.
- The tile requests are distributed to multiple threads. Namely, the client application is not blocked until all the responses are obtained from the WMS server. Also, first and foremost, for the client machines with multi-core processors, the performance of the application increases in direct proportion to the processor core number. For a quad-core machine, 12-tile request takes almost 3 to 4 times less than a regular single core machine.

- Beside prefetching methods, in this method, the memory is not used aggressively. It is a known fact that prefetching methods are not perfect. Because of that, some of the prefetched tiles may never be requested by the user but consumes significant space in memory.

The disadvantages of this method are:

- There is no replacement policy for the cache. When the client application is out of memory, the cache is entirely cleaned and the application returns to its opening state.
- The size of the cache is not configurable. It is limited to the size of the heap space of the application. Because of that, as the cache size increases, the overhead on Garbage Collector of Java increases and the overall performance of the application decreases.
- There is no logic in the cache. The cache is only filled with all the tiles requested by the client. Namely, the cache is designed only for keeping the previous moves of the client but not the possible next moves. Since, most users do not frequently revisit the former locations; the efficiency of the cache reduces as the cache grows.

3.2 Prefetching neighbor tiles of a requested tile

Besides caching the tiles, many of the Web GIS applications apply another optimization. In this optimization, client application prefetches the neighbor tiles of each requested tile. By this way, since the tiles that represent the next navigation of the client are already in the cache, seamless navigation is provided to the client. As seen in the figure below, a tile has 8 neighbor tiles. So, by applying this method, instead of 1 tile, 9 tiles are requested from the WMS server.

Figure 7: Neighbors of a tile

Most of the Web GIS applications start prefetching the neighbor tiles after receiving all the tiles in the view extent. For instance, a JavaScript based WMS client library called *OpenLayers* (see Related Work section) has a run parameter called "*buffer size"*. Buffer size determines the level of the neighbors that will be retrieved. Namely, if the buffer size is 2, instead of 8 tiles, 24 tiles are requested from the WMS server (as seen in the figure below).

Figure 8: 2-level neighbors of a tile

This method can be improved by prefetching the tiles that will be requested if the user zooms in/out. For 1 zoom level, the number of tiles will be prefetched is 9. So, as seen in the figure below, for both zooming in and zooming out, the number of tiles that will be requested for the center tile is 9 and in total, 18 tiles are prefetched.

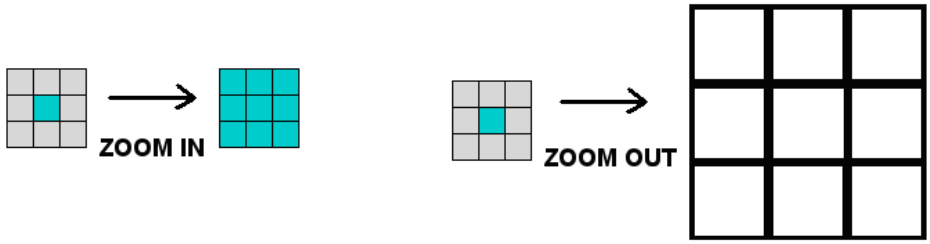

Figure 9: Tiles prefetched for one level zoom in and zoom out

The reason for requesting 9 tiles instead of 1 for each tile is very obvious. Either you zoom in or zoom out, the next level tile moves to another cell in the view. For the tiles that are close to the edges of the view extent, most of the time, the next level tile stays out of the view extent. To visualize what is meant by this explanation, see the figure below. If the client application requests 1 next level tile for each tile (since neighbor cells of the next level tile would not have been requested), there will be gaps between each next level tile. Also, since the view extent will contain only the next level tiles of the center tile, navigating to any direction after zoom operation will result to see the empty tiles.

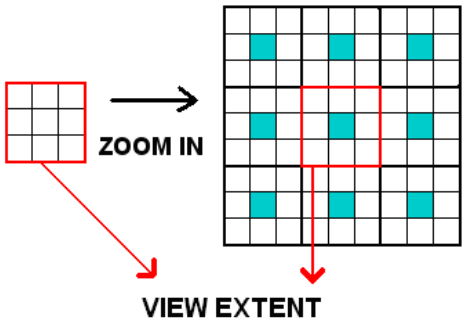

Figure 10: Tiles requested for 1-level zoom in, when requested tile number for each tile is 1

On the other hand, as it can be seen in the figure below, if the requested tile number is 9, there will not be any empty tile and prefetching 1-level zoom in/out will result to seamless navigation.

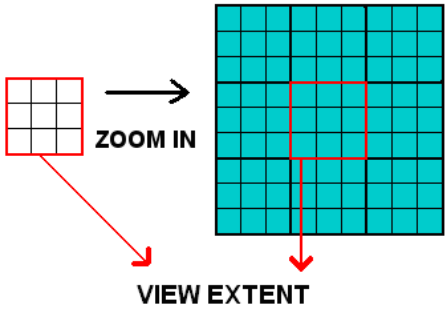

Figure 11: Tiles requested for 1-level zoom in, when requested tile number for each tile is 9

“more tiles requested in the present” means “less tiles to request in the future”. However, it is also valid that “more tiles requested in the present” means “more traffic in network” and more traffic decreases the overall performance of the application. Because of this reason, determining the number of tiles to be prefetched is very crucial. Requesting next level tiles of the current tile multiplies the number of requests to the web server by 2. Because of this reason, most of the Web GIS applications do not prefer to prefetch next zoom levels, but only prefetches the neighbor tiles of the current tile in the same zoom level.

For the experiments in this study, zoom out levels are discarded during the prefetching process, because requesting 9 tiles for 1-level zooming brings extreme burden to the system. Prefetching for both zoom levels will result in 18 tile requests to WMS server at once and this will lock the system unnecessarily. So, there is no need to prefetch both zoom levels when comparing this method with other two algorithms. This choice can be validated with the performance and hence the widespread usage of this method.

All the results and the comparisons of each method are given in Chapter 5.

The advantages of this method are:

- Since the adjacent cells of a tile are retrieved during the request of that tile, the tiles needed to load for the next navigation are already present in the cache.
- The process of prefetching for a tile is handled in a separate thread, not the main thread. By this way, especially for multi-core machines, the performance overhead is reduced to minimum.
- The size of the cache is configurable. By default, cache size is limited to the memory reserved for the client application. But, the user can change this value in unit of tile number. Namely, if the user enters 1000 in the configuration file of the client application, it means maximum 1000 tiles can be saved in the cache. When the maximum tile number is reached, cache

replaces the new tiles with old one according to LRU replacement policy. According to this policy, firstly, the tile that accessed earliest is removed from the cache.

The disadvantages of this method are:

- The memory consumed is more than the other two methods.
- Because of the first disadvantage, memory reaches to its maximum capacity sooner than the other methods. So, Java Garbage Collector is obligated to run more frequently and this causes to reduce the overall performance of the client application.
- If the prefetching process contains the zoom levels too;
    - Memory consumption increases dramatically. If 9 tiles fit in the view extent, 9 tiles for each tile are requested for 1-level zoom prefetching. In total, 81 tile requests are made. In comparison to prefetching only neighbor tiles (8 tiles retrieved), the burden over the WMS server is multiplied by 10 times.
    - Number of threads increases with the number of tiles to prefetch. Although a thread pool is utilized for limiting the number of threads that can be executed at the same time, the number of threads that should run to complete the prefetching process is always same (81). So, when the navigation changes before the prefetching is over, the remaining tiles to be prefetched are discarded. Namely, the number of tiles prefetched is less than the number of tiles that forms a whole zoom level. Since some tiles are not retrieved, the effect of this prefetching method decreases.

3.3 RAP: Retrospective Adaptive Prefetch

This study proposes a prefetching algorithm that optimizes the previously explained prefetching method by injecting a heuristic behavior. The pseudo-code of the algorithm is given in APPENDIX–A.

The prefetching algorithm presented in this study uses a heuristic approach for retrieving the next possible tiles of a given tile according to the user's former navigations. Number of former navigations (history or depth) is a configurable parameter. As depth increases, the accuracy of the result increases as well. The number of tiles to prefetch is not affected with the depth number. The only factor that changes the number of tiles to be prefetched is the pattern of the former navigations in history. Namely, if the user stays in a stable path during navigation, the algorithm assumes that s(he) will probably keep on moving on the same path in the next moves. Otherwise, for instance, if the user makes random moves on map, the estimation of the algorithm will not be effective and hit ratio of the prefetching cache will reduce.

The proposed algorithm has a grading formula for determining the number of neighbor tiles with their relative locations according to source tile. The formula can be given as below:

In the next step of the algorithm, all these values are combined and divided into total differences. Namely, for instance, each *Easting* has a "difference value" which is normalized according to the hit ratio obtained by the prefetched tiles and those tiles are calculated with the help of that *Easting* value. In the formula above, difference value of *Easting* is given as "($h_e[\ i + 1\ ] - h_e[\ i\ ]$)". At first, hit ratio of each *Easting* has a value of 1. The reason for that is giving that move a chance to affect the prefetching process more, because it is the most current move available in the history. The final step of determining whether to prefetch tiles in a direction is calculating the ratio of total *Eastings* to total differences. By this way, the effect of each Easting is evaluated for the prefetching. If absolute value of the result is equal

to or over 1, it means the application will prefetch some tiles left or right of the source tile. The same process is valid for *Southing* too.

On the other hand, evaluation of *Zooming* is a little bit different than the others. *zoomFactor* value for each move in history is a decimal number. This value is the proportion of the move's order in the history to the total size of the history. The number obtained is the effect of that zooming move to the prefetching process. Namely, the effect of a zooming moves decreases in direct proportion to its order in the history list. By this way, redundant prefetching requests to the WMS server for the zooming moves are prevented. For instance let's assume that history size is 5. If the moves in the history are "east, east, east, east and zoom-in", the *zoomFactor* value of each move will be as given in the table below:

Table 1: Calculation of *zoomFactor* (sample 1)

History Order (older-to-new)	**Navigation**	**zoomFactor calculation**	**zoomFactor**
1	east	0 * (1 / 5)	0
2	east	0 * (2 / 5)	0
3	east	0 * (3 / 5)	0
4	east	0 * (4 / 5)	0
5	zoom-in	-1 * (5 / 5)	-1
Zooming			-1

After 3 moves to west the table will be as given below. In that case, the effect of zoom-in in history is decreased from 1 to 0.4 and *zoomFactor* became -0.4. The round value of *zoomFactor* is 0, and then we can say that there is no need for prefetching another zooming level.

Table 2: Calculation of *zoomFactor* (sample 2)

History Order (older-to-new)	**Navigation**	**zoomFactor calculation**	**zoomFactor**
1	east	0 * (1 / 5)	0
2	zoom-in	-1 * (2 / 5)	-0.4
3	west	0 * (3 / 5)	0
4	west	0 * (4 / 5)	0
5	west	0 * (5 / 5)	0
Zooming			-0.4 → 0

For the history view below; although, zoom-in actions of the user are left behind in the history, their effect still continuous. Zooming value of -0.6 is rounded to -1, which means there is still a chance that the user will zoom-in in the next move.

Table 3: Calculation of *zoomFactor* (sample 3)

History Order (older-to-new)	**Navigation**	**zoomFactor calculation**	**zoomFactor**
1	zoom-in	-1 * (1 / 5)	-0.2
2	zoom-in	-1 * (2 / 5)	-0.4
3	west	0 * (3 / 5)	0
4	west	0 * (4 / 5)	0
5	west	0 * (5 / 5)	0
Zooming			-0.6 → -1

The sign of Easting determines the direction of prefetching. “1” means retrieving tiles on the right (east) side of the source tile and “-1” means retrieving tiles (west) side of the source tile. The same logic is valid for *Southing* too. “1” means tiles below the source tile and “-1” means tiles above the source tile on map. As seen in the table below, for all 8 combinations of Easting and Southing, tile number to prefetch is either 1 or 3.

Table 4: Neighbor tiles to prefetch according to Easting and Southing

TILES TO PREFETCH	EASTING	SOUTHING
	1	0
	-1	0
	0	1
	0	-1
	1	1
	1	-1
	-1	-1
	-1	1

Determining zooming in or out is made according to the sign of *Zooming* value. If *Zooming* is “-1”, 1-level-zoom-in of center tile; if *Zooming* is “1”, 1-level-zoom-out of center is prefetched.

The prefetching mechanism is notified when the center of the view extent changes. If the action of the user on the map does not change the center tile, there is no need to prefetch any tiles. By the help of the notification mechanism, redundant requests to WMS server are prevented. In each center change notification, the top-left tile of the view extent is saved. Top-left tile is the reference tile for finding the tiles to prefetch. Also, the numbers of horizontal and vertical tiles in the view extent are used during calculations. The top-left tile of the view extent in figure below is the one that has a circle inside. The number of horizontal tiles (*numWidth*) is 4 and the number of vertical tiles (*numHeight*) is 3.

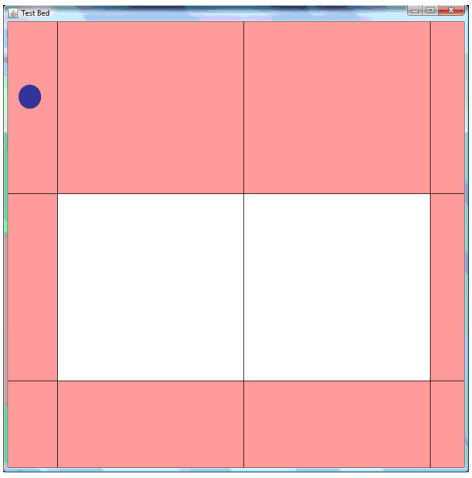

Figure 12: Top-left tile and tiles on the edge of the view extent

In this method, instead of initiating threads for each tile in the view extent, only one thread is created for all tiles on the edge of the view extent. Since, their neighbor tiles are still inside the view extent, the tiles that are in the center of the view extent are not significant for prefetching process. The tiles on the edge of the view extent in the figure above are represented with darker filling. No prefetching is done for the 2 tiles in the center. Starting from the top-left tile, a distinct number (starting from 1) is assigned to each tile on the edge clockwise. In the figure below, the numbers assigned for each tile on the edge of the view extent can be seen. The view extents are drawn with thicker borders around them. For the view extent on the left, there are 8 tiles are on the edge. For the view extent on the right, this number is 10.

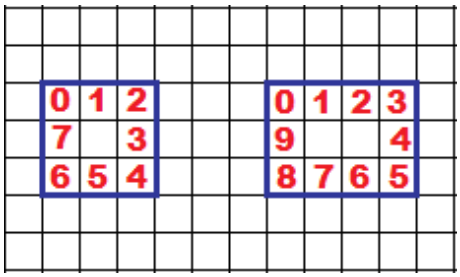

Figure 13: Tiles on the edge of the view extent

These numbers are used for indicating the order of tiles to prefetch. As show in the table below, for 8 directions, the order of prefetching is given.

Table 5: Prefetching order for each direction

DIRECTIONS	**TILE ORDER**
EAST	3 , 2 , 4
WEST	7 , 6 , 0
NORTH	1 , 0 , 2
SOUTH	5 , 4 , 6
NORTHWEST	0 , 1 , 7 , 2 , 6
NORTHEAST	2 , 3 , 1 , 4 , 0
SOUTHWEST	6 , 7 , 5 , 0 , 4
SOUTHEAST	4 , 5 , 3 , 6 , 2

The tiles to prefetch for each tile on the edge and for each direction are already known by the algorithm. This is the only section of the algorithm where pre-processing is applied. In the beginning of the application, prefetching order tables for all possible *numWidth* and *numHeight* values are calculated.

According to the prefetching order table above, if the *Easting* is 1 and *Southing* is 0, 1 tile on the right of the source tile is prefetched. So, for the view extent below, the source tiles are 2, 3 and 4. As a result, the tiles on the right side of these 3 tiles are prefetched (tiles X, Y and Z in the figure below). If the *Easting* is -1 and *Southing* is 1, it means the direction is southwest and the source tiles are 5, 6 and 7. 6 darker tiles on the left and down side of the source tiles in the figure below are the candidates for prefetching (tiles with letters from A to G).

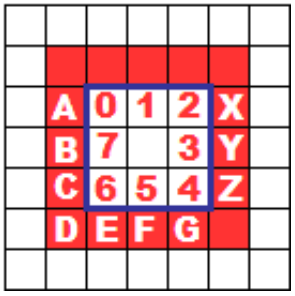

Figure 14: Tiles to prefetch in southwest direction

The order is important, because, for each direction, the location of tiles to be prefetched is in the same direction of the navigation. For instance, for southwest direction, the first source tile is 6, because the tiles to be prefetched for this tile is C, D and E and those tile are placed in the direction of the next navigation; southwest. Tiles A, B, F and G are prefetched after C, D and E. By this way, if the prefetching process is canceled because of the navigation change of the user, the tiles that will be requested first will be present in the prefetching cache.

Different than the previous algorithm (prefetching neighbor tiles of a requested tile), the proposed algorithm does not retrieve the zoom level tiles of "every" tile in the view extent, but only the center tile. For instance, for 9 tiles in view extent, instead of 81 tiles, only 25 or 9 tiles are prefetched for the center tile. The number 25 is chosen if at least one of *numWidth* or *numHeight* is greater than 3. Otherwise, prefetching 9 tiles will be adequate to fill the next view extent. By this way, the number of tiles prefetched is reduced for performance optimization. In the following two figures, the tiles prefetched for the center tile in 1-level zoom in and out conditions are given.

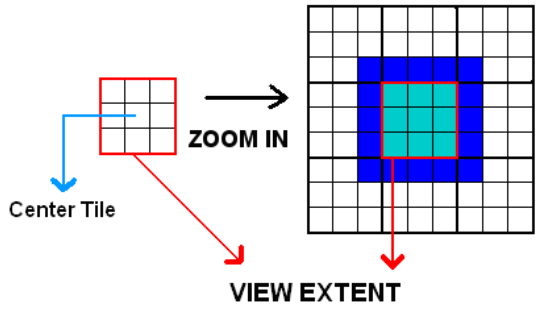

Figure 15: Tiles requested for zooming in center tile

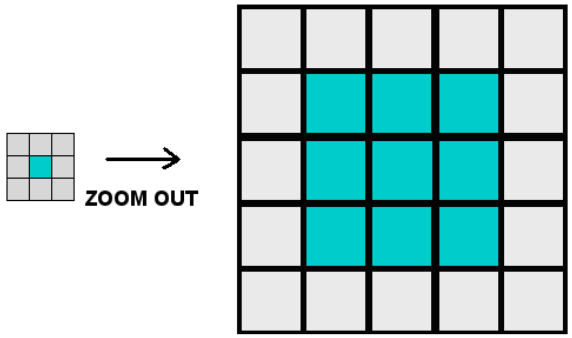

Figure 16: Tiles requested for zooming out center tile

Execution of the algorithm over a sample scenario

In this section, the processes executed in the proposed algorithm are explained over a sample scenario. The scenario contains 7 navigation actions of the client and the method of finding out the neighbor tiles to prefetch is clarified. Depth of the list where the former navigations are kept is selected as 5.

The figure below demonstrates the navigations. The cell in “2, 2” is the center of the initial view extent which consists of 9 tiles. The square around the last tile represents the view extent of the final state. 2 bright arrows shows the first two navigations of the user which are removed from the history because they are expired. By expired, it is meant that 5 (history depth) navigations are passed after these two tiles and because of that they are no longer needed for calculations.

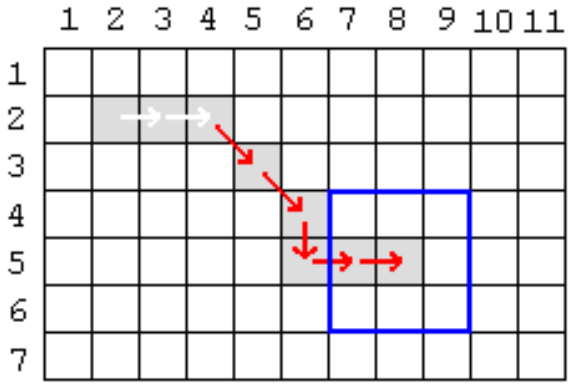

Figure 17: Sample navigation scenario when history depth is 5

In the table below, each step of grading formula execution is given. There is no column for *Zooming*, because the navigation steps do not contain any zoom in/out action.

Table 18: Execution of grading formula for the sample scenario when history depth is 5

Step	**Previous Tile**	**Current Tile**	**Hit Ratio**	**Easting**	**Southing**
1	2 , 4	3 , 5	0.692	(5 – 4) * 0.652 = 0.652	(3 – 2) * 0.652 = 0.652
2	3 , 5	4 , 6	0.652	(6 – 5) * 0.652 = 0.652	(4 – 3) * 0.652 = 0.652
3	4 , 6	5 , 6	1	(6 – 6) * 1 = 0	(5 – 4) * 1 = 1
4	5 , 6	5 , 7	1	(7 – 6) * 1 = 1	(5 – 5) * 4 = 0
5	5 , 7	5 , 8	1	(8 – 7) * 1 = 1	(5 – 5) * 1 = 0
Total				3.304	2.304
Avg				3.304 / 4 = 0,826 -> **1**	2.304 / 3 = 0,768 -> **1**

The data near “Avg” row are the final *Easting* and *Southing* values used for calculating the tiles to prefetch. Avg for *Easting* is calculated as dividing total *Eastings* into total *Easting* differences between each data in history. The divider in this table is not 5, but 4, because in step 4, the difference is 0 (6-6) and it has no effect on total total *Easting* differences. The divider for *Southing* is 3, because, in both steps 4 and 5, the differences are 0 (5-5).

As a result of the grading formula, 7 tiles are prefetched. Since *Easting* and *Southing* are both positive, tiles on the right-down side of the source tile are retrieved. The tiles retrieved for final view extent (darker cells) are displayed in the figure below:

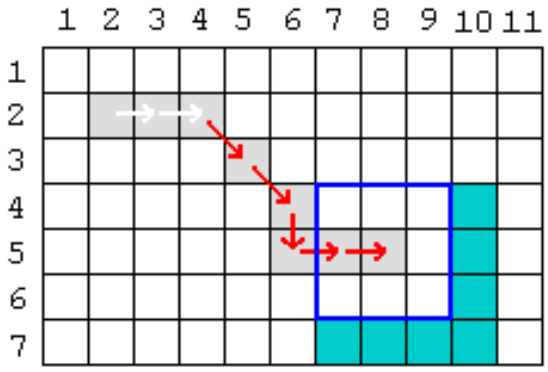

Figure 19: Tiles prefetched when history depth is 5

9 darker tiles are added to prefetching cache and 9 tiles inside the view extent are added to default tile cache. From now on, when there is a tile request on a view extent change, firstly, prefetch cache is checked. If the requested tile is in this cache, no request is sent to the WMS server and tile in the prefetch cache is retrieved. Otherwise, the default protocol is executed. Tile is searched in the default tile cache. If there is no entry for that tile, it is requested from the WMS server.

Formula for cache size calculation according to memory

Cache size is the maximum number of tiles that can be saved in the cache. In the proposed algorithm this value is configurable. If the user does not specify the cache size, it is calculated by a simple calculation. The formula is:

```
CACHE_SIZE = ( Maximum_Memory – Current_Memory ) /
                        Tile_Size
```

Maximum_Memory is the maximum memory that can be used during the execution of the client application. *Current_Memory* is the total memory used in the client application at that moment. So, subtracting *Current_Memory* from *Maximum_Memory* gives the maximum amount of memory can be utilized for the cache. Although this memory is used for the application's other processes, it can be discarded. Because it is relatively small and also when the application is out of memory and cache does not reach to its limit. At that point, 75% of main and prefetch caches are removed. The percentage is chosen 75% as a result of the following simple assumption:

- Clearing both caches will rewind the application to its initial state and all the tiles cached will be lost. So, the performance improvement of the caches will cease.
- Removing less than 3/4 of the tiles (for instance 50% of the tiles) in the caches can be adequate for that moment to prevent out of memory errors, but after a short period of time memory usage will hit the limit and the caches will be cleared again. If this process is executed too frequently, it means most of the application's execution time is spent with intense memory usage. In these conditions, Garbage Collector of Java runs excessively and this will affect the performance of the application negatively.

Because of these reasons, 75% is a feasible ratio. Reaching the memory limits next time is postponed more afterwards and enough amounts of tiles are kept in the caches to keep the applications performance in a decent level. Also, according to the

benchmark results in table below, 75% comes out as the most feasible value to choose.

Table 20: Performance of different ratios for clearing the cache

Ratio	Cache Hit	Cache Miss	Number of GC Executions	Total Time (ms)
%25	94	45	38	10788
%50	92	47	25	7558
%75	88	51	15	5993
%100	55	84	8	8054

Most of the executions for 25% and some of the executions for 50% could not be lasted until the end of the test. Application was frozen because of the excessive number of GCs (Garbage Collections). In those ratios, the time spent for the memory allocation and deallocation processes choked the application and the application could not respond the user actions. So, the values under total time and cache hit/miss columns are obtained from the successful execution. As a result, 25% and 50% ratios are eliminated directly, because in those ratios, memory utilization and application stability cannot be obtained.

On the other hand for the ratios 75% and 100%, the execution of the application lasted until the end of the tests. However, the performance of 100% was worse than 75%, because each time a GC is executed caches are cleared and next requests are redirected to WMS server instead of the prefetching cache. For 75%, GC executions are more than 100%, but the performance is better. Despite the fact that ¾ of the prefetching cache is cleared (since most recently prefetched tiles are still in the cache), user's future requests are met from the cache.

This study does not specifically claim that 75% gives better results than any other closer ratios like 70% or 80%. 75% is a reasonable average value to choose and in some circumstances, slightly lower or higher ratios can give better results as well.

The advantages of this method are:

- Since the possible future cells that will be visited in the next navigation are retrieved during the request of a tile, the tiles needed to load for the next navigation are already present in the cache.
- As in the previous method, each process of prefetching for a tile is handled in a separate thread other than the main thread.
- As in the previous method, the size of the cache is configurable.
- The prefetching method is optimized for not making excessive requests to WMS server by determining the number of tiles to be prefetched by implementing a heuristic approach. By this way, the future tiles are predicted and unnecessary neighbor and zoom level tiles are not requested.

The disadvantages of this method are:

- The memory consumed is more than the first method, because of the separate cache for keeping the prefetched tiles.
- Because of the first disadvantage, just like in the previous method, memory reaches to its maximum capacity sooner than the first method. Since, the memory consumption is less than the second method; this method reaches to its maximum capacity later. Nevertheless, the memory consumption can be considered as reasonable near its contribution to performance enhancement.

CHAPTER 4

TEST BED

This study represents a prefetching algorithm that aims to find the best probable map tiles that forms the next possible moves of the user. In this section, the basic structure of the library used for demonstrating the proposed prefetching algorithm and the alternative caching algorithms are explained.

SwingX-WS is a Java library that introduces several Java beans for interacting with web services within Swing applications. The reason to select this library in the study is very obvious. Because, it is an open source project and its components are very easy to use.

The most popular map server of the today's Web GIS technology is provided by Google. The map server of Google divides the world into equally sized tiles which are referenced as Cartesian coordinates, x and y. Although, this design decision gains a remarkable performance improvements, WMS map service standard defined by OGC® accepts the a map request as a bounding box given in latitude and longitude coordinates. In this context, we can say that Google does not obey the standard implementation proposed by WMS. But, as mentioned earlier, a WMS extension called "Tiled WMS" is also proposed by OGC®. So, we can also say that dividing the maps into tiles is encouraged by OGC too. Herein, JXMapViewer component of SwingX-WS steps in. JXMapViewer provides a build-in map viewer that retrieves map data from any WMS server that is implemented by obeying the rules defined in OGC® WMS implementation specification. But, the major feature of this component is dividing the map into the tiles just like Google does and "Tiled WMS" recommends. This is very important because caching the map requests depends on tiling the map into "equally sized images" (tiles).

In this study, the WMS service provided by Geoserver 1.7.2 (http://geoserver.org) is used as map data provider. This server provides a map layer called "topp:states" which displays the states of USA. In the demo application, States map provided by this WMS service is displayed as below:

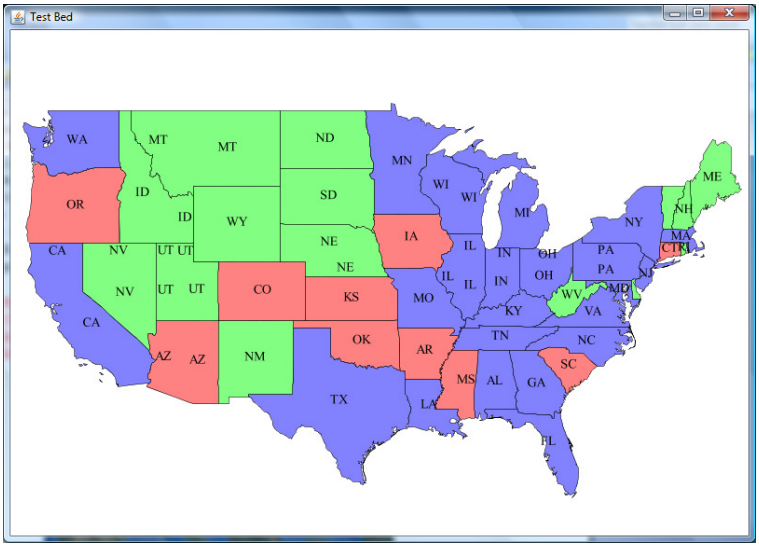

Figure 21: States Map in Geoserver WMS Service

There are many public OGC® WMS servers over the internet [20]. To increase in variety and use a WMS server that operates on internet, another WMS server is used. This WMS server (called *World Mineral Deposits*) is provided by Mineral Resources Division, Geological Survey of Canada and the chosen layer is a patchwork map depicting generalized bedrock domains for era-level age ranges and predominant rock types [21]. The map below shows the age ranges of the rocks in Turkey and its neighbors. Each color represents a different age range, but within this study, their meanings are not essential information. The colors and their meaning can be obtained with a WMS request called *GetLegendGraphic*, which gives the legend information of the layer given in the request [22].

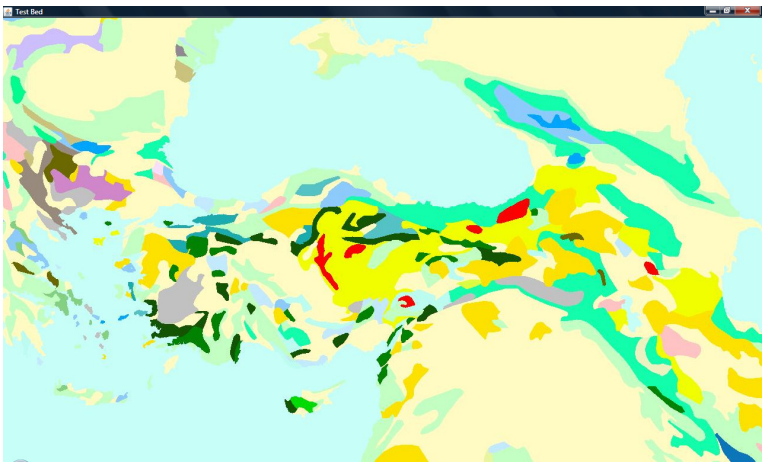

Figure 22: The age ranges of the rocks in Turkey and its neighbors in World Mineral Deposits WMS Service

To demonstrate the effects of the caching and prefetching mechanisms used in this study, a relatively small tile size (300 pixels) is preferred for both WMS services mentioned.

In JXMapViewer, using these two WMS services as map providers is very easy. The WMS web service is represented by the WMSService class. BaseURL for the web service and the layer with properties are set on the WMSService instance. The BaseURL is the URL where the service resides (without any parameters). The layer represents a particular portion of the data available from this map service. In general, these two pieces of information are all needed to connect to a WMS server.

Most WMS servers have several layers; usually a single base layer and several data layers. The base layer usually covers the entire world and the data layers are transparent data sets which can overlaid on top of the base layer. In this study, only the base layer is used, because any other additions to the base layer will not affect the result of the study.

Once the WMSService instance is created, there is only one thing left; wrapping it in a new WMSTileFactory and set it on the map. The WMSTileFactory is the class that will convert WMS requests into the tile based coordinate system that JXMapViewer

understands. EPSG:4326 is used as Spatial Reference System (the method of representing the surface of a sphere or other shape on a plane).

WMSTileFactory uses the build-in caching mechanism defined in TileCache class. The caching mechanism of TileCache is very simple. When a WMS request comes to WMSTileFactory, a new thread for loading the requested tiles is initialized. This thread checks the cache and if there is a hit for the requested tile, it passes to the next tile. Otherwise, the tile is requested from the WMS server and put into the cache. The cache is cleared when an OutOfMemoryError is thrown. This error occurs when the Java Virtual Machine cannot allocate an object because it is out of memory, and no more memory could be made available by the garbage collector. As it can be seen, there is not a complicated logic in the caching mechanism. The size of the cache is limited with the maximum memory allocated for the application. This can be considered as the lowest level of caching without any complicated strategy.

To convert a tile request to a valid WMS request, WMSService class has a method called toWMSURL. This method has four parameters which defines a tile; *x*, *y*, *zoom* and *tile size*. *"x"* and *"y"* indicates the position of the tile cell in the whole map, in other words order of the tile. *"zoom"* gives the zoom level of the tile. In JXMapViewer, valid zoom range is between 0 and 15. *"tile size"* is the size of the tile in pixels. For a WMS request, *"tile size"* corresponds to width/height of the image returned by the WMS service. JXMapViewer has a flag to turn on/off displaying the x, y and zoom properties and the borders of the tiles drawn. When, this flag is opened, the map is displayed as below:

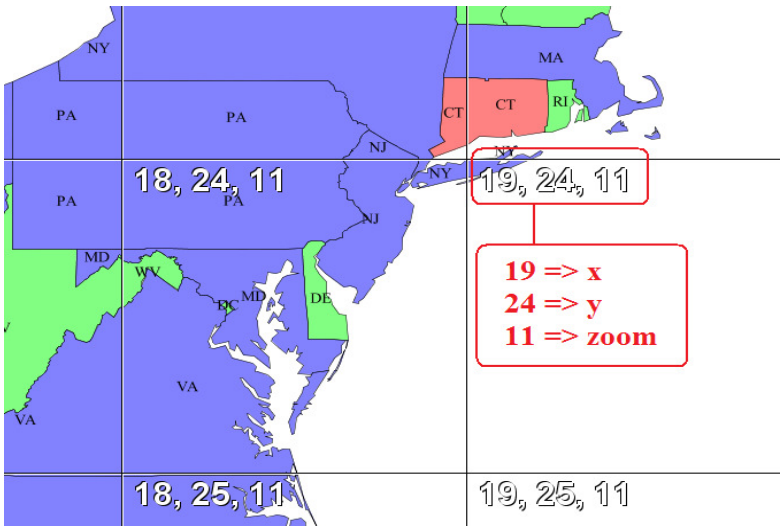

Figure 23: Distribution of map tiles in JXMapViewer component

4.1 Test Simulation

For performance tests, a simple demo application is developed. The main purpose of this application is automating the performance tests and exporting the results in PDF format to keep them permanently. The application is build on JXMapViewer component of SwingX-WS library.

The simulation of the user actions are handled by the “Robot API” of Java. Robot API provides the methods for executing the mouse actions programmatically. So, each map action like navigating different directions and changing zoom levels are automated via calling these methods.

When the application is initialized, the user enters the configurations for the simulation or just selects the predefined data which are obtained from a configuration file. The configuration file contains the following properties:

wms_name_list: name of the WMS service. WMS services that are used in this study are Geoserver's topp:states and World Mineral Deposits service of Mineral Resources Division, Geological Survey of Canada.

wms_URL_list: URL of the each WMS service in *wms_name_list.* All the WMS parameters that construct a valid request are appended after this given URL. For instance, the URL in this list for the WMS request

http://localhost:8080/geoserver/wms?bbox=-130,24,-66,50&styles=population&Format=image/png&request=GetMap&layers=topp:states&width=550&height=250&srs=EPSG:4326

is:

http://localhost:8080/geoserver/wms?

wms_layer_list: Layer name of each WMS service in *wms_name_list.*

wms_SRS_list: SRS name of each WMS service in *wms_name_list.*

wms_center_lat_list: Latitude of the center location where the map is opened for each WMS service. Range is [-90, 90].

wms_center_lon_list: Longitude of the center location where the map is opened for each WMS service. Range is [-180, 180].

wms_zoom_level: The zoom level which the map opens with. Its range is determined by JXMapViewer as 0 to 15.

navigations: Navigation steps to be applied during the simulations for each WMS service in *wms_name_list*. There are 9 basic directions and two zoom options that form the navigation steps. The user can add as many navigations as s(he) desires.

wms_selected: The index of the selected WMS service which will be used for the simulation. The index begins from "0".

cache_selected: Index of the selected cache among the cache list in configuration window (see the next figure below). There are 3 cache strategies:

- TC: Default caching strategy. Only the visited tiles are kept in the cache.
- SP: Simple prefetching implemented over TC method. Only the 8 neighbor tiles of a tile is prefetched and saved in the cache.
- RAP: Proposed algorithm. This is the cache that the proposed algorithm is implemented. When this cache is selected, value in the *cache_depth* configuration parameter is used as explained below.
- HCBP: Tile cache that uses Hilbert Curve Based Prefetching algorithm [13] mentioned in RELATED WORK section.
- PKM: Tile cache that uses previous-k-movements prefetching algorithm mentioned in RELATED WORK section [14].

cache_depth: Number of tiles will be kept in the history for RAP and PKM method. If the *cache_depth* is "5", it means 5 tiles passed to reach to current tile are kept in history list.

number_of_items_in_cache: Maximum number of tiles will be kept in the cache (cache size). If this value is over 0, it is used as cache size. Otherwise, cache size is determined according to the memory reserved for the client application. The formula to determine the cache size according to memory is given in Section 3.4.

A WMS service may limit the bounding box of a layer smaller than the whole earth. In that situation, center latitude, center longitude, zoom level and navigation steps shall be given as staying inside the bounding box of the layer. Otherwise, WMS service may throw a WMS exception and will not return a valid map image.

The configuration window below is opened with the values in the configuration file. However, the user may enter a WMS service with its parameters by pressing "Add WMS" button. User can enter all the parameters except navigations via this screen. Navigation steps are entered in configuration window.

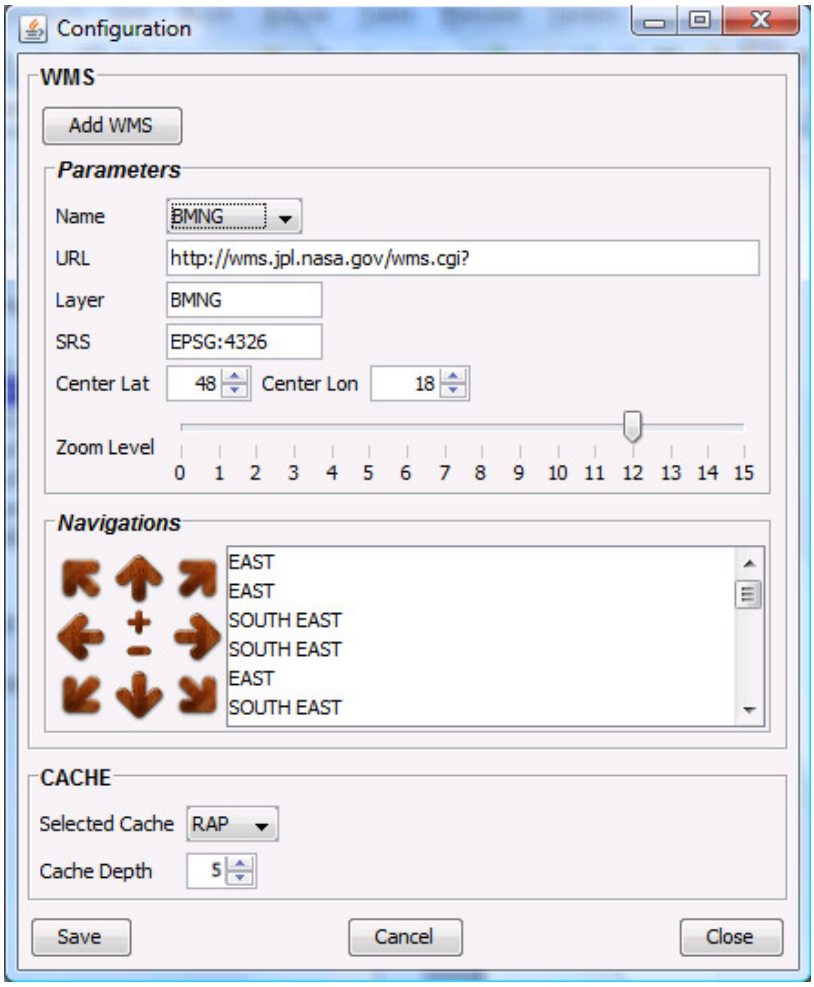

Figure 24: Configuration window

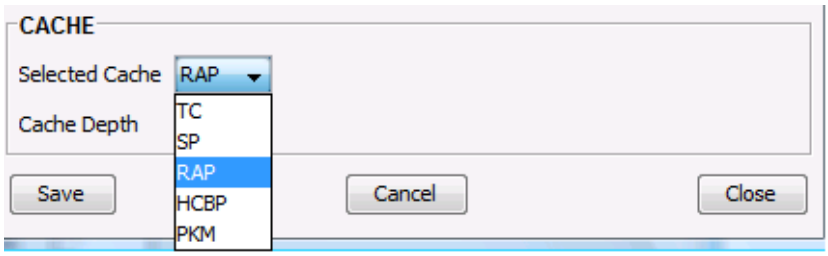

Figure 25: Configuration window: Cache selection

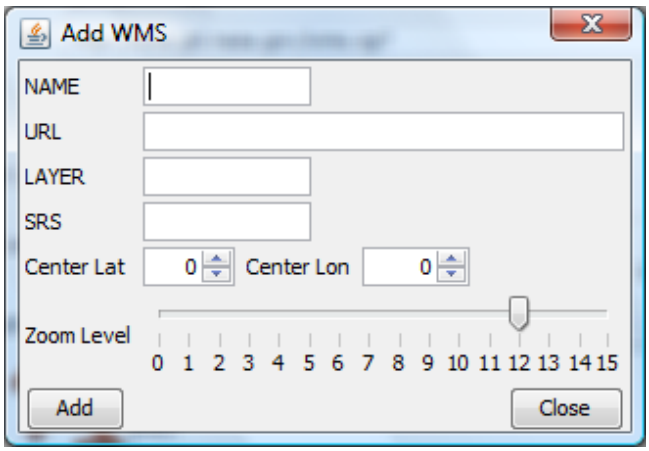

Figure 26: Add WMS window

4.2 Outcome of the Tests

After the configuration is entered, all the performance tests are done automatically without any user interaction. For instance, let's assume that the user selects the Geoserver as the WMS service, enters the initial zoom level as 12 and navigation steps as follows: zoom out, east, southeast, south, east, zoom in, zoom in, northwest, west and north. After the simulation is over, the results are saved in a PDF file. Exporting the results in PDF format is done with the help of another open source library called *iText* (http://www.lowagie.com/iText). The test results contain the following information:

Navigation Summary Table: This is a table that contains the navigation steps that are applied by the application automatically. Zoom levels, latitudes and longitudes for each navigation step are also presented in this table.

Table 6: Navigation summary table

Navigation	Zoom Level	Center Latitude	Center Longitude
Initial	11	40,00000	-100,00000
ZOOM OUT	12	40,00000	-100,00000
EAST	12	40,00000	-88,00000
SOUTH EAST	12	34,01671	-76,00000
SOUTH	12	27,58021	-76,00000
EAST	12	27,58021	-64,00000
ZOOM IN	11	27,58021	-64,00000
ZOOM IN	10	27,58021	-64,00000
NORTH WEST	10	29,22938	-67,00000
WEST	10	29,22938	-70,00000
NORTH	10	30,85242	-70,00000

Charts: Two different charts are generated at the end of a test. The charts are created with the JFreeChart library (an open-source framework for Java, which allows the creation of complex charts in a simple way (http://www.jfree.org/jfreechart)).

The first chart is "JVM Memory Usage" chart which displays the memory used in Java Heap Space and Maximum Heap Space over time.

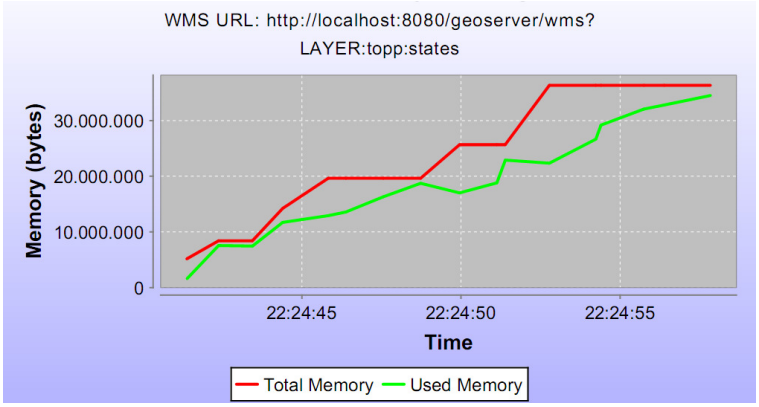

Figure 27: JVM Memory Usage chart

The other chart is called “Map Refresh Time with Tile Numbers” and it is a combination of two charts. The first chart displays the refresh time of the map over time. Each tick on the graph represents a navigation step given in the Navigation Summary table. The graph below this graph gives the number of tiles loaded for each navigation step. When these two tables are evaluated together, the time spent for loading the tiles requested for each navigation steps can be obtained.

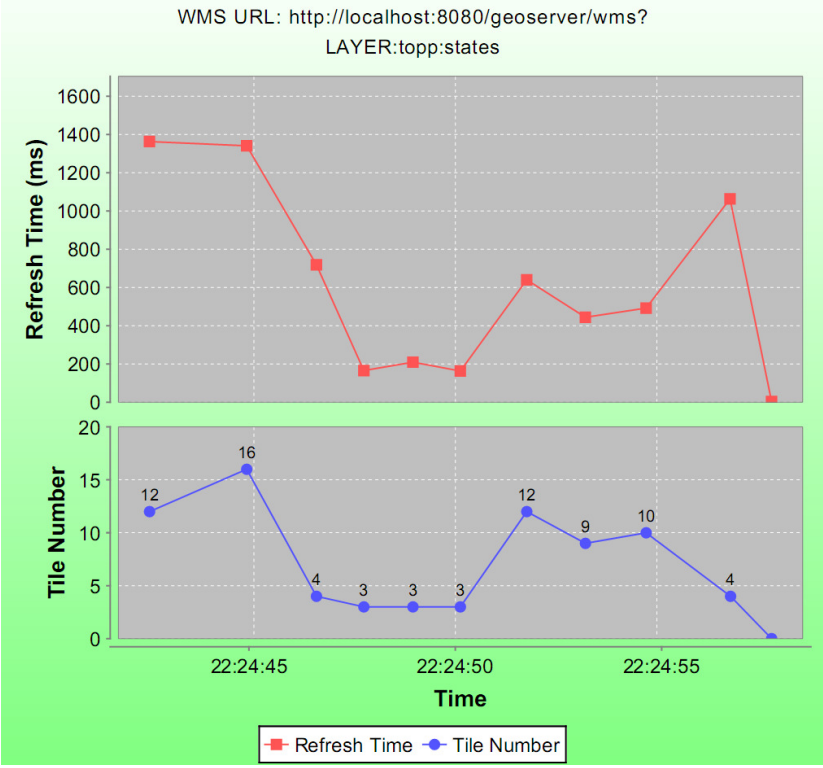

Figure 28: Map Refresh Time with Tile Numbers chart

Performance Results Table: This is the table where the results of the simulation are summarized. The numerical values of the graphs are given in this table. For instance, by looking at this table, it can be concluded that no tiles are loaded in last step (north), because they were retrieved from the cache. Also, in 4th step (southwest), all 3 tiles requested are obtained from the prefetching cache and no requests are sent to WMS server.

Table 7: Performance results table

Navigation	Tile Number	Cache Hits	Cache Misses	Refresh Time (ms)	Used Memory	Total Memory
Initial	12	0	12	1726,419978	6,282,344	10,706,944
ZOOM OUT	16	0	16	1909,965652	17,608,936	26,636,288
EAST	4	0	4	516,628231	28,770,152	36,962,304
SOUTH EAST	3	3	0	66,146954	30,686,416	36,962,304
SOUTH	3	3	0	64,629722	33,590,936	36,962,304
EAST	3	3	0	16,252345	34,216,656	36,962,304
ZOOM IN	12	0	12	1016,552941	45,084,120	55,181,312
ZOOM IN	9	9	0	157,304071	51,369,736	55,181,312
NORTH WEST	10	6	4	364,770383	55,103,176	85,037,056
WEST	4	0	4	362,267551	62,453,816	85,037,056
NORTH	0	0	0	4,242160	68,737,696	85,037,056

CACHE STATS
HITS: 24
MISSES: 52

CHAPTER 5

RESULTS AND DISCUSSION

In this chapter the performance and memory consumption of RAP method is compared with other methods. Tests are executed in 2 separate WMS servers to increase variety:

- Geoserver (runs on local machine)
- World Mineral Deposits Service (runs on internet)

In tests executed in Georserver, RAP method is compared with Tile Caching and Simple Prefetching methods. In second test executed on World Mineral Deposits Service, besides the other two methods, proposed algorithm is compared with Hilbert Curve based prefetching algorithm and previous-k-movement method as well.

Tests are completely automated by simulating the user actions and results are generated when the simulation is over. During the tests no user interaction is required. Also, 1 second delay is put between navigations to reflect the average user behavior.

On each test, “randomly” generated 21 navigations are simulated and all these navigations are performed for all the methods included in the tests. In total, 100 tests are executed and in each test, a unique list of 21 navigations is performed. All the navigation sets executed in each test are completely different than each other. They all contain 21 navigations to reflect the consistency in comparisons.

The statistics of the results acquired at the end of the tests are given in APPENDIX - C. Refresh times and memory usages for each method are evaluated and some statistical values are obtained.

A sample of 100 navigation sets generated randomly for the simulations is given in the table. So, if we count the initial state as well, there are 22 states and 21 transitions in total.

Table 8: Navigation Summary

Navigation	Zoom Level	Center Latitude	Center Longitude
Initial	11	40.00000	-100.00000
EAST	11	40.00000	-94.00000
EAST	11	40.00000	-88.00000
SOUTH EAST	11	37.06732	-82.00000
SOUTH EAST	11	34.01671	-76.00000
EAST	11	34.01671	-70.00000
SOUTH EAST	11	30.85242	-64.00000
SOUTH	11	27.58021	-64.00000
SOUTH	11	24.20741	-64.00000
SOUTH EAST	11	20.74294	-58.00000
EAST	11	20.74294	-52.00000
EAST	11	20.74294	-46.00000
ZOOM IN	10	20.74294	-46.00000
ZOOM IN	9	20.74294	-46.00000
ZOOM IN	8	20.74294	-46.00000
ZOOM IN	7	20.74294	-46.00000
ZOOM IN	6	20.74294	-46.00000
EAST	6	20.74294	-45.81250
EAST	6	20.74294	-45.62500
EAST	6	20.74294	-45.43750
ZOOM IN	5	20.74294	-45.43750
ZOOM IN	4	20.74294	-45.43750

The figure below illustrates these 21 navigations (transitions). For simplifying the illustration, zoom actions are marked as “5x” and “2x” over the figure.

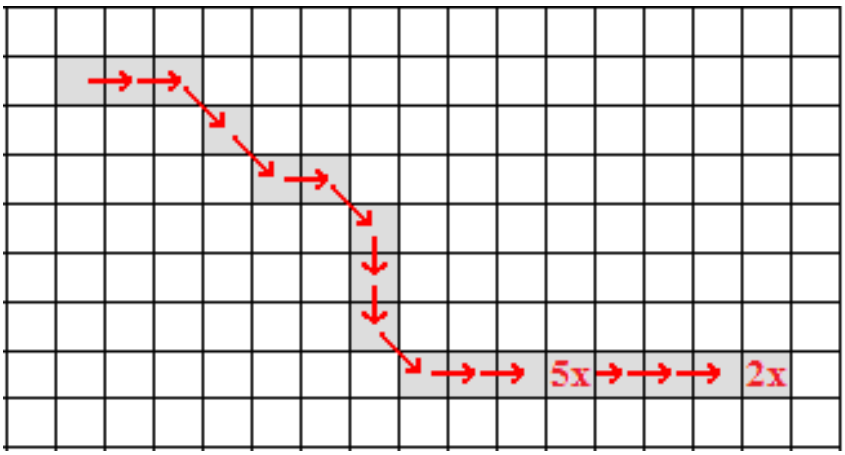

Figure 29: Cells transitions of navigation steps

Each step of the test simulation for this sample scenario is given in APPENDIX –B. In that section, for each navigation, direction of the navigation, top-left tile of the current view extent, *Easting*, *Southing* and *Zooming* values are presented. Also, east, south and zoom differences, hit-ratios and *zoomFactors* for each data in history are shown in a table row-by-row. Also, at the end of each navigation, tiles to be prefetched are written in a table. Now on, throughout the document, the phrase "sample scenario" refers to this navigation set selected among the 100 navigation sets.

Besides the 100 random navigations, 2 navigation sets that give the best and worst results for RAP method are given too. Those navigation sets and their meaning for RAP method are explained at the end of this section (5.3Extreme scenarios).

5.1 Execution of the Tests in Geoserver

The first tests are executed with Geoserver. The initial and final view extents during the tests are shown in the figures below. These view extends are obtained during the execution of the sample scenario.

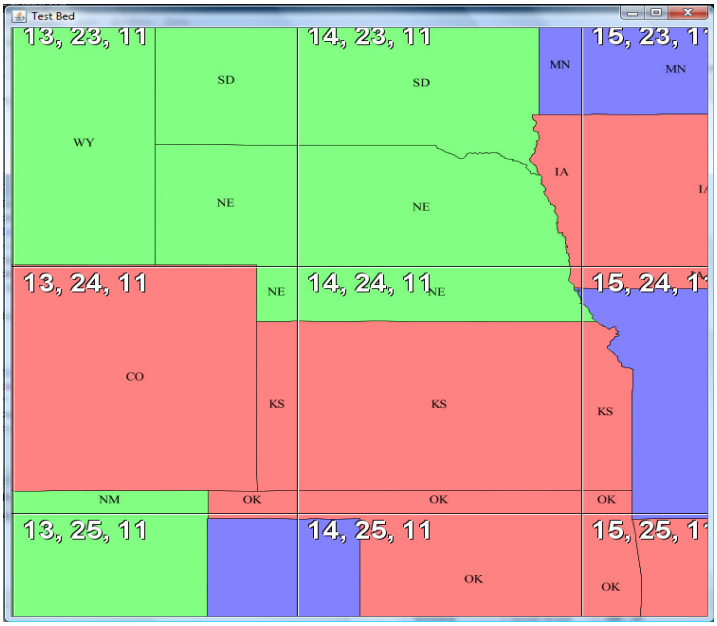

Figure 30: Initial view extent in Geoserver's topp:states (center tile is x = 14, y = 24, zoom = 11)

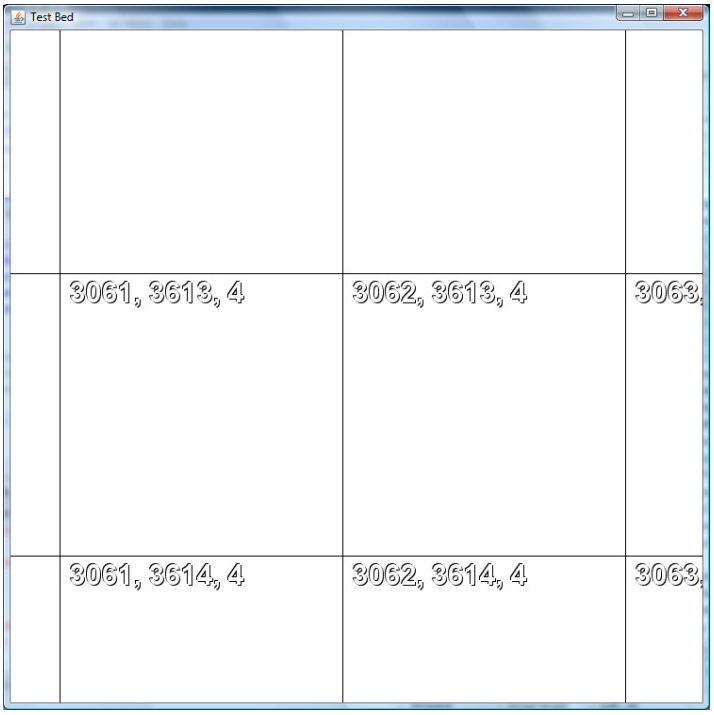

Figure 31: Final view extent in Geoserver's topp:states (center tile is x = 3062, y = 3613, zoom = 4)

The average memory usage during the execution of tests is given in 3 diagrams below. In all 3 tests, the memory usage increases in direct proportion to the time passes as expected.

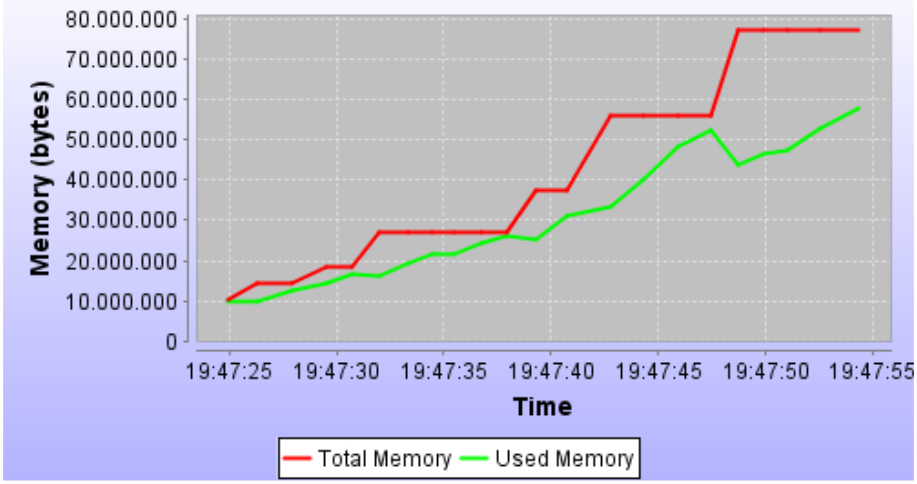

Figure 32: Memory Usage over Time for TC method (Geoserver's topp:states)

The figure above shows the average memory usage when TC method is used during the simulation. The memory consumption in this test is the lowest according to other two tests. The reason is very clear; the TC method does not utilize a prefetching mechanism, so no memory is allocated for a prefetching cache.

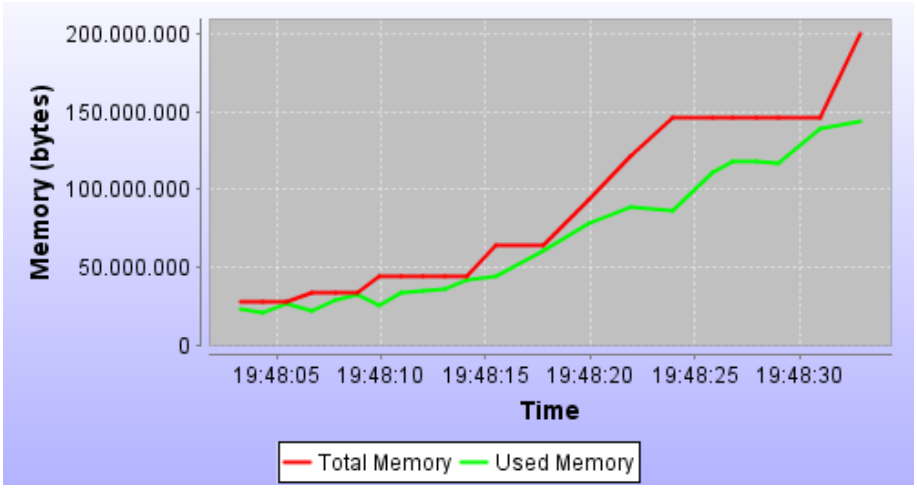

Figure 33: Memory Usage over Time for SP method (Geoserver's topp:states)

The figure above shows the memory usage when the SP method is used during the simulation. The memory consumption in this test is the highest according to other two tests. Since this algorithm has a separate cache for prefetching and the cache is filled with the prefetched tiles, the memory usage of this method is more than the TC method. Also, it is more than the RAP method as well. Because, when this method prefetches the entire neighbor tiles around a tile, in RAP method, only the tiles that are predicted as candidates of next moves. Since, the number of tiles prefetched in this method is more than the RAP method, more memory is allocated.

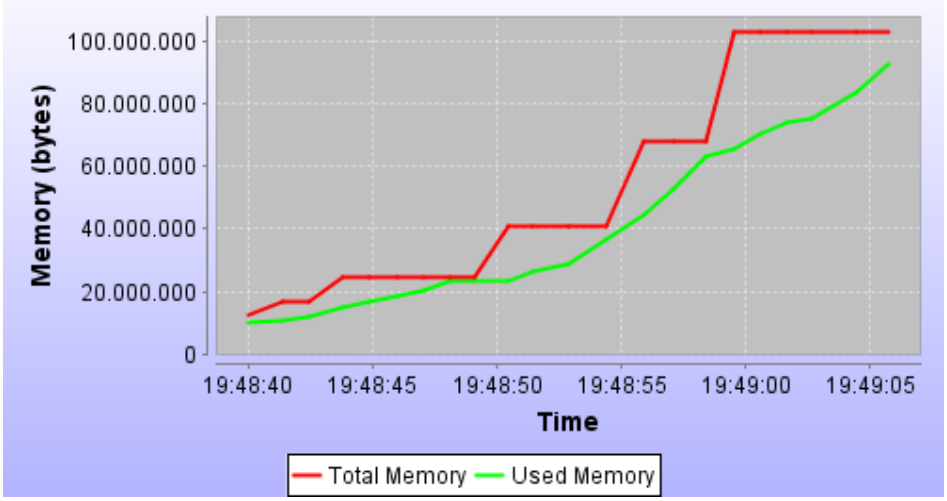

Figure 34: Memory Usage over Time for RAP method (Geoserver's topp:states)

The memory usage in RAP method is given in the figure above. In this situation, memory consumption is between the other two methods. The cache utilized for prefetching causes allocating more memory than the TC.

Two graphics in the figure below shows the number of tiles loaded for each navigation and the refresh time of the map when the sample scenario is executed. As expected, the map refresh time increases when more tiles are requested from map server. It can be clearly observed that the first time 12 tiles are loaded takes more time than the later 5 times 12 tiles are loaded. The reason for that difference is; although 12 tiles are requested to construct the view extent image, not all the requests

are directed to WMS server. Since, some tiles are already in the prefetching cache, they are loaded from the cache and that reduces the refresh time.

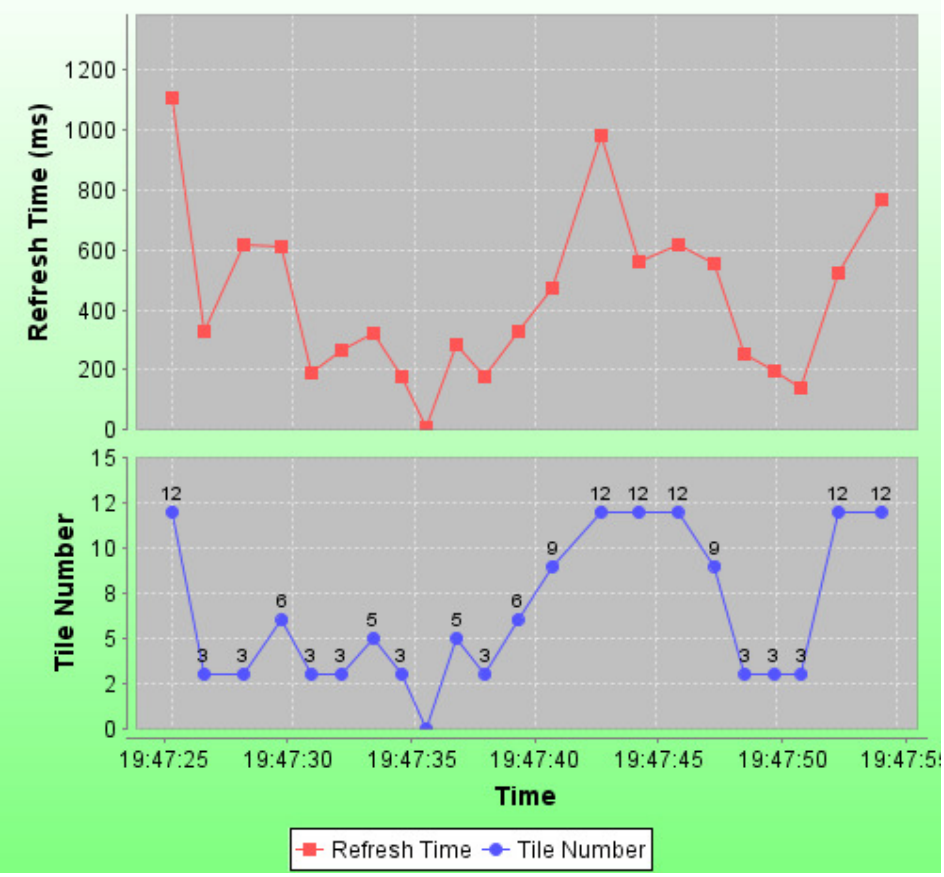

Figure 35: Refresh Time & Requested Tile Number over Time for TC method (Geoserver's topp:states)

The refresh times when SP method is used are given in the figure below. In this graphic, refresh times for the navigations to neighbor tiles (other than zooms) are less than the first method. Because, in SP method, during navigation, the entire neighbor tiles of the current tiles are prefetched. But this advantage is lost when a zooming operation is performed by the user. In that case, the neighbor tiles prefetched are useless. The time and memory spent for this process does not help the user at all. Because of those lost time and memory, total refresh time is more than the first method.

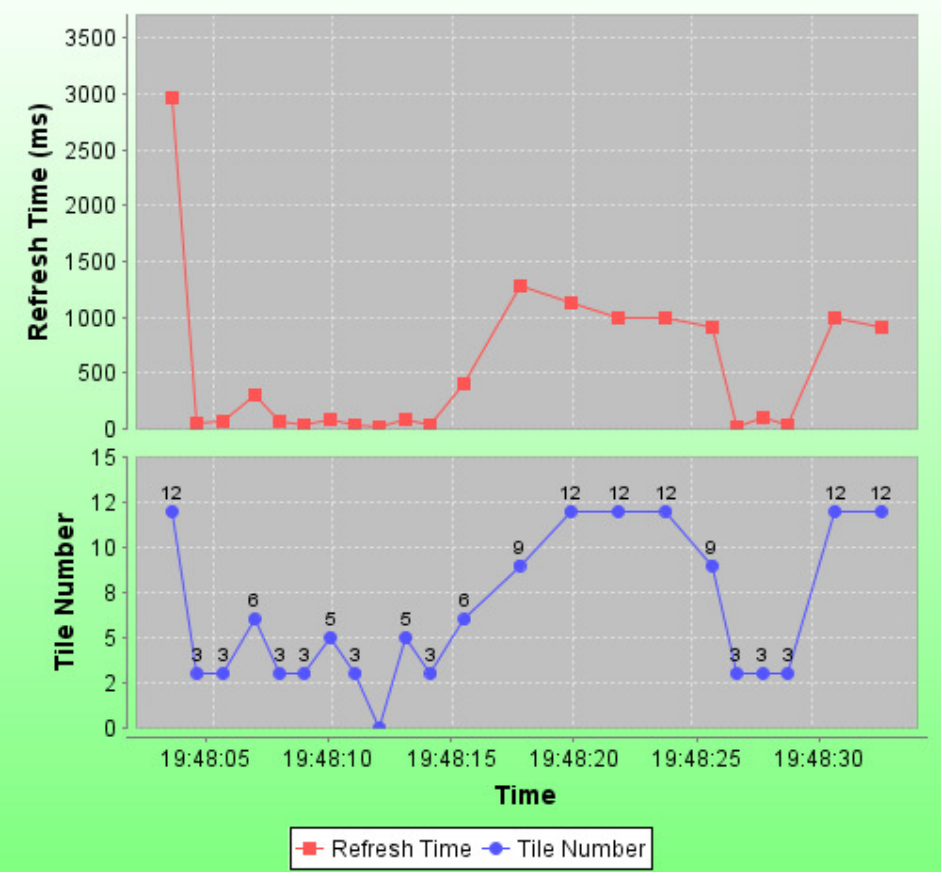

Figure 36: Refresh Time & Requested Tile Number over Time for SP method (Geoserver's topp:states)

On the other hand, the performance of RAP method can be considered better than the other two methods. By examining both the graphic and the *Refresh Times* table

below, it can be observed that tiles needed to construct the view extent for the subsequent navigations are already in the prefetching cache and no requests are made to the WMS server.

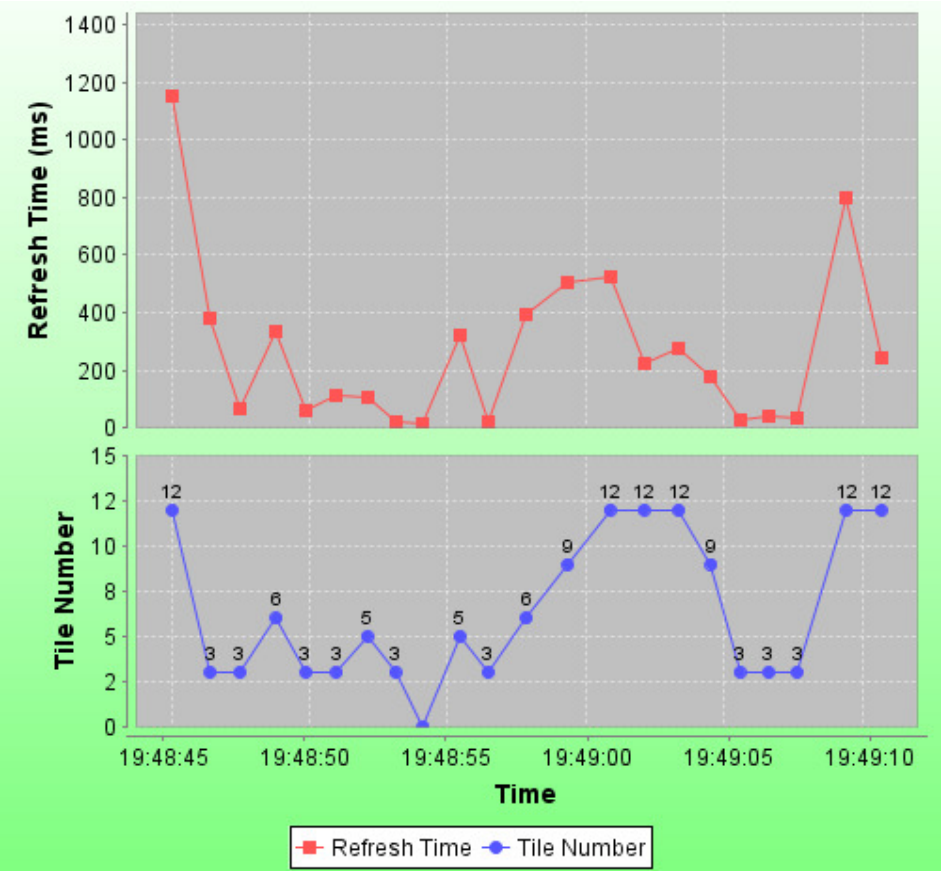

Figure 37: Refresh Time & Requested Tile Number over Time for RAP method (Geoserver's topp:states)

In the table below, the refresh times of each navigation in sample scenario is given for each method.

Table 9: Refresh Times in sample scenario (Geoserver's topp:states)

NAVIGATIONS	REFRESH TIMES (ms)		
	TC	SP	RAP
Initial State	*1107.970706*	*2962.687157*	*1152.543689*
EAST	329.712855	62.060960	382.424582
EAST	620.125742	69.773951	68.800644
SOUTH EAST	611.535823	300.716432	332.493376
SOUTH EAST	189.358424	69.000110	61.958434
EAST	266.433583	30.583216	113.176954
SOUTH EAST	319.458758	82.644023	104.271912
SOUTH	176.638651	31.701516	23.616130
SOUTH	6.534909	16.715532	16.030529
SOUTH EAST	286.408189	87.125040	319.418250
EAST	177.163857	42.674977	18.035812
EAST	330.791204	408.796369	392.735669
ZOOM IN	471.701291	1287.298665	503.781347
ZOOM IN	980.571833	1128.092677	522.979876
ZOOM IN	561.601824	1001.044825	224.464841
ZOOM IN	615.048281	987.587110	274.608924
ZOOM IN	554.017620	909.840496	176.477178
EAST	250.426801	17.701691	27.956905
EAST	196.520228	96.799174	43.251866
EAST	138.332386	40.002291	34.254074
ZOOM IN	521.456777	987.298665	801.292063
ZOOM IN	771.359768	909.840496	243.039549
Total Refresh Time	**8375.198804**	**8657.298216**	**4685.068915**

In the table below, the memory usage during the execution of each navigation in sample scenario is given for each method.

Table 10: Used Memory & Total Memory in sample scenario (Geoserver's topp:states)

NAVIGATIONS	USED MEMORY (bytes)		
	TC	SP	RAP
initial	9,851,672	23,349,040	9,902,496
EAST	9,823,392	21,010,344	10,812,688
EAST	12,487,168	26,281,168	11,902,160
SOUTH EAST	14,440,784	22,013,160	14,754,752
SOUTH EAST	16,710,408	29,267,560	16,997,456
EAST	16,335,512	32,643,624	18,293,696
SOUTH EAST	19,566,432	25,664,248	20,586,552
SOUTH	21,500,152	33,376,456	23,447,936
SOUTH	21,555,816	35,311,664	23,506,592
SOUTH EAST	24,385,656	35,589,296	23,461,360
EAST	26,250,760	42,329,968	26,227,792
EAST	25,430,928	44,040,872	29,080,992
ZOOM IN	31,013,896	60,967,352	36,452,912
ZOOM IN	33,242,808	77,748,184	44,732,560
ZOOM IN	40,265,768	88,549,216	52,785,712
ZOOM IN	48,300,080	86,611,592	62,959,976
ZOOM IN	52,569,240	110,391,416	65,320,248
EAST	43,826,312	117,499,808	70,205,992
EAST	46,384,872	117,508,688	74,263,880
EAST	47,209,160	117,265,408	75,349,232
ZOOM IN	52,787,720	139,229,328	83,414,512
ZOOM IN	58,020,280	143,293,456	92,673,032
Total Memory Allocated in the End	***77,115,392***	***199,340,032***	***102,731,776***

Values in "Total Memory Allocated in the End" row are given only for informative purposes. Those values do not give the memory usage of the algorithms, but only the total memory allocated at that moment by the JVM to prevent out of memory errors. The memory values that matter are the ones written in bold font before that row.

As mentioned earlier, these tests are performed 100 times with randomly generated unique navigation sets. The table below gives the averages of the total refresh times and memory usages for each method at the end of these 100 executions.

Table 11: Average refresh times & memory usage for 100 executions (Geoserver's topp:states)

Averages	TC	SP	RAP
Refresh Time (ms)	8034.383335	8244.343422	4566.377590
Memory Usage (bytes)	60,120,550	149,156,746	90,774,362

The cache statistics of each method obtained at the end of the test simulations is given in the table below:

Table 12: Cache Statistics (Geoserver's topp:states)

METHOD	Cache Hit	Cache Miss
TC	0	139
SP	58	81
RAP	88	51

As a result of the test, using SP method is not a wise choice for optimizing the performance of a Web GIS application, since it does not provide a better refresh time and memory usage. However, in RAP method, the average execution time of the simulations is reduced to 56.8% (4566.377590 / 8034.383335) of TC method. The only disadvantage of this method is using more memory than TC method (almost 1.51 times). But, it is a known fact that, in today's computers, increasing the size of the memory is not hard as upgrading the CPU.

5.2 Execution of the Test in World Mineral Deposits Service

Besides Geoserver, to diversify the tests, a free WMS server that runs on internet is used. The WMS server used in this test is "World Mineral Deposits Service" provided by Mineral Resources Division, Geological Survey of Canada. The initial and final view extents obtained from this service during the tests are shown in the figures below. As in the previous section, these view extends are obtained during the execution of the sample scenario.

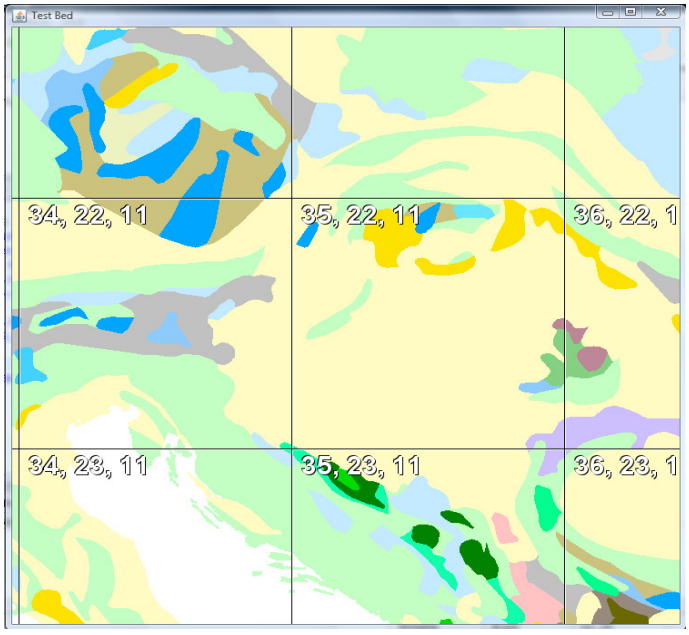

Figure 38: Initial view extent in World Mineral Deposits Service (center tile is x = 35, y = 22, zoom = 11)

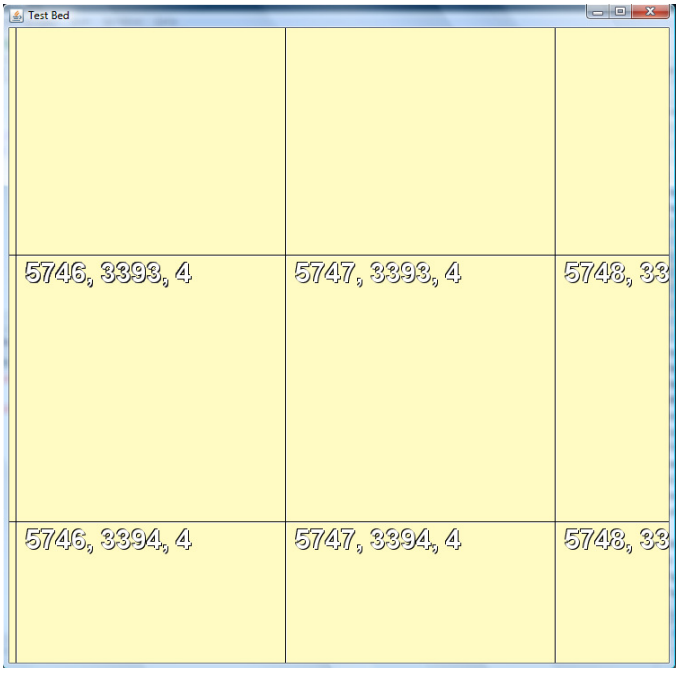

Figure 39: Final view extent in World Mineral Deposits Service (center tile is x = 5447, y = 3393, zoom = 4)

This time, besides the other two methods, RAP method is compared to HCBP and PKM methods as well. The following 5 diagrams show the memory usages of all methods. All observations from the previous tests in Geoserver are valid for these tests too. Whilst, TC method consumes the least memory, SP method uses the most memory among the others. On the other hand, the memory consumed by RAP method is lower than SP method, but higher than TC, HCBP and PKM methods.

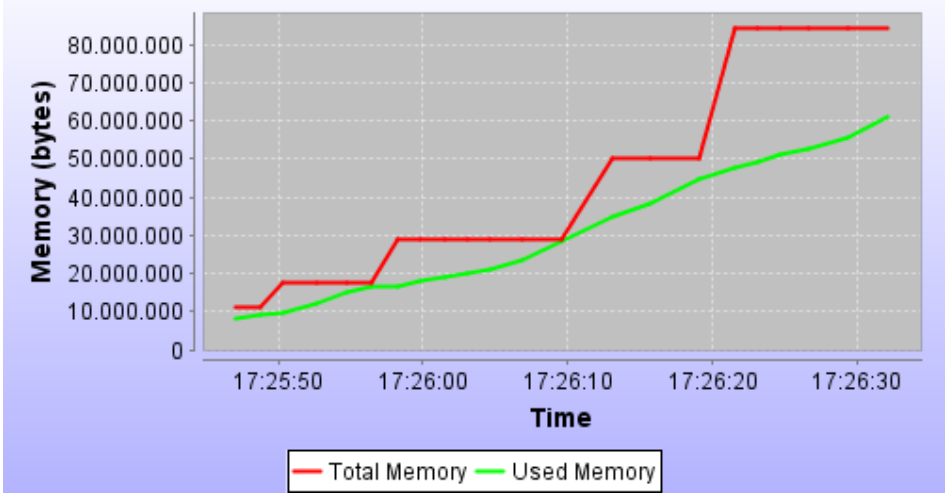

Figure 40: Memory Usage over Time for TC method (World Mineral Deposits Service)

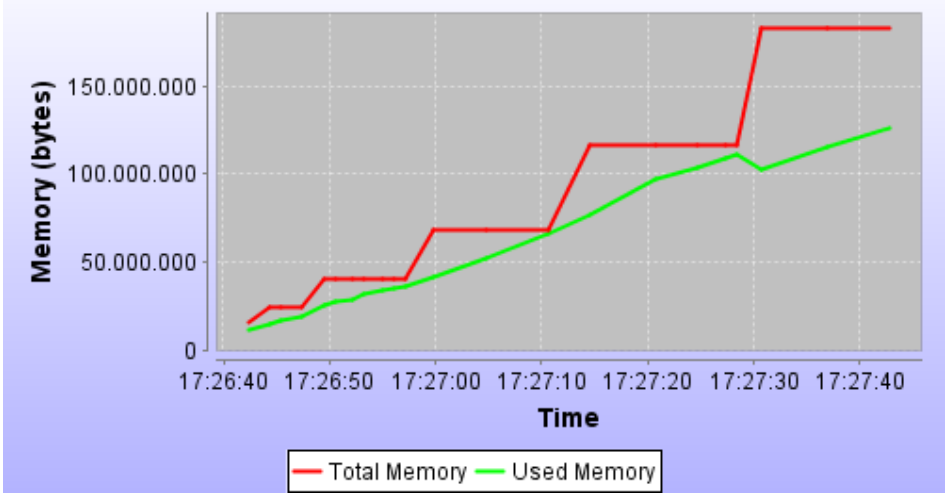

Figure 41: Memory Usage over Time for SP method (World Mineral Deposits Service)

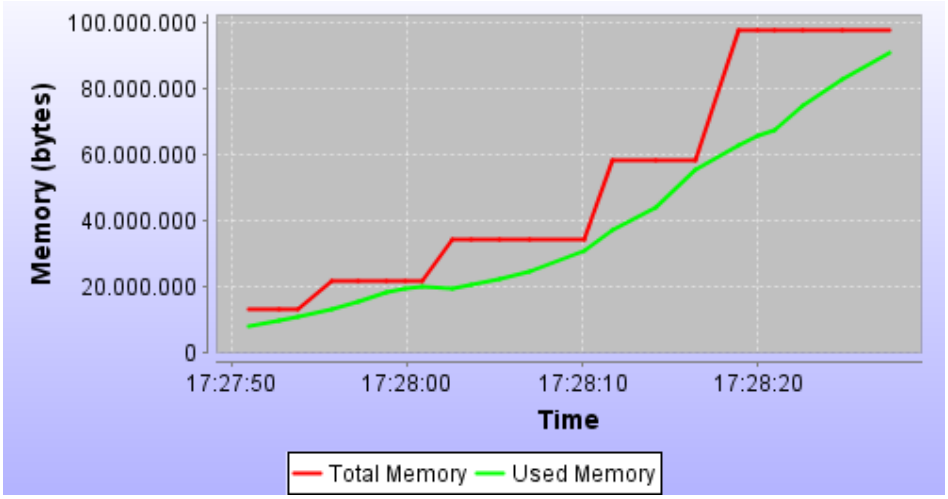

Figure 42: Memory Usage over Time for RAP method (World Mineral Deposits Service)

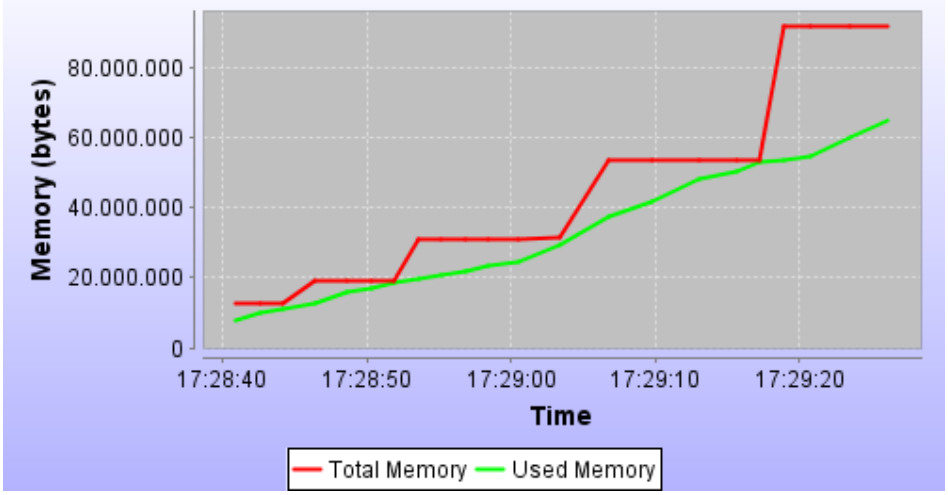

Figure 43: Memory Usage over Time for HCBP method (World Mineral Deposits Service)

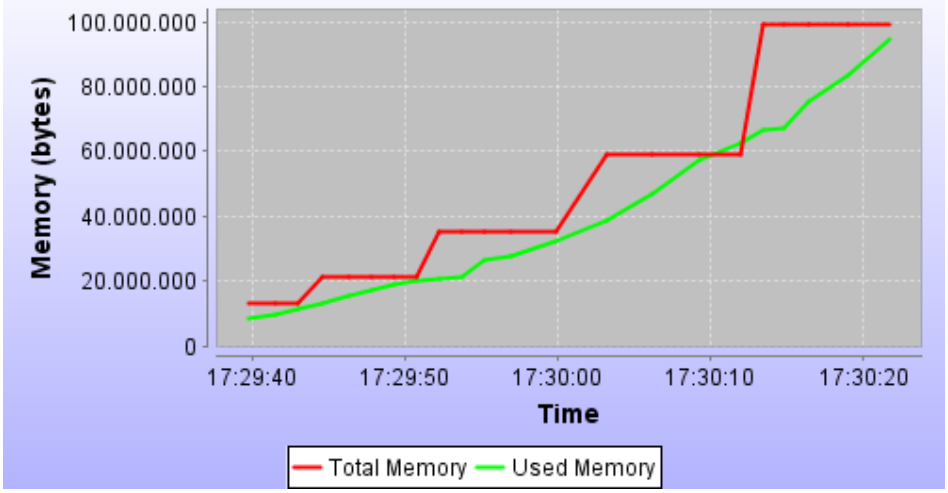

Figure 44: Memory Usage over Time for PKM method (World Mineral Deposits Service)

The diagrams below are basically the same diagrams of the tests executed in Geoserver for 4 methods. The data in these diagrams show that the performance results of the algorithms are consistent with the previous tests.

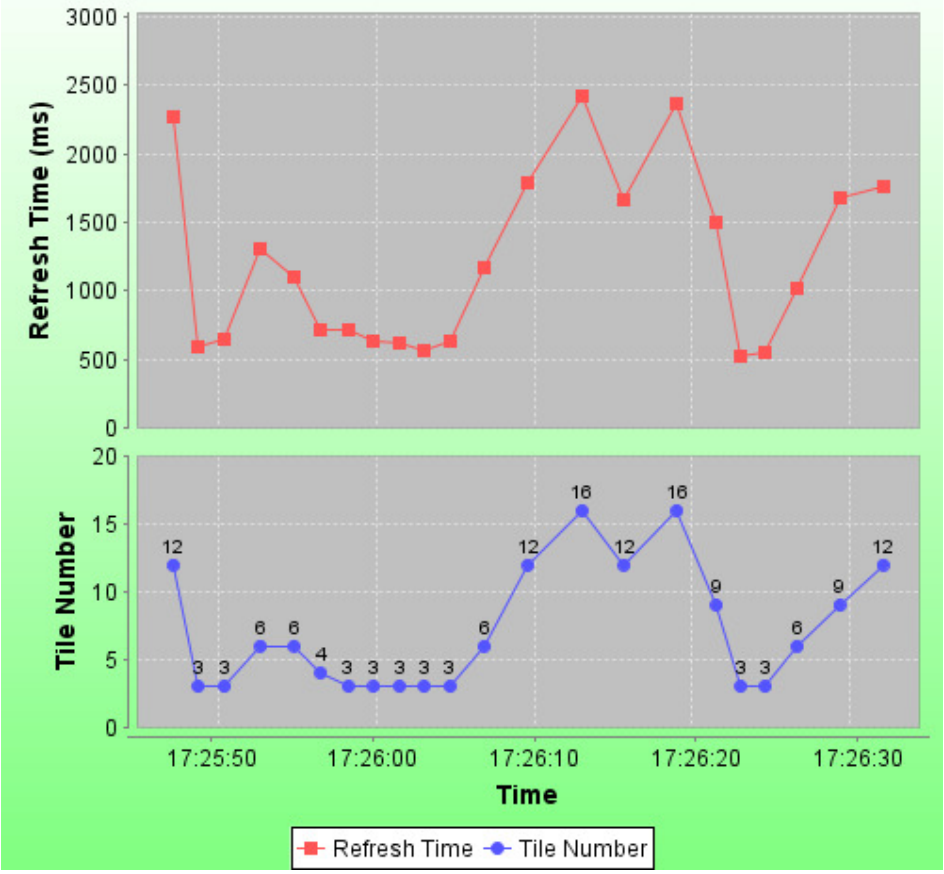

Figure 45: Refresh Time & Requested Tile Number over Time for TC method (World Mineral Deposits Service)

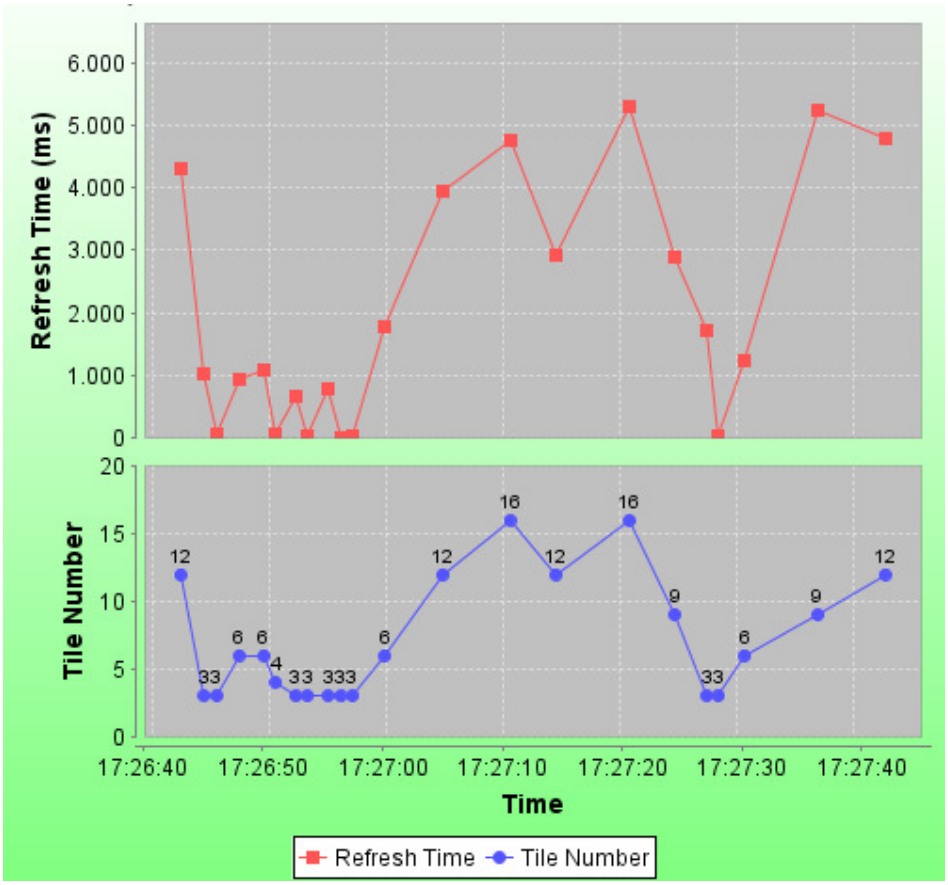

Figure 46: Refresh Time & Requested Tile Number over Time for SP method (World Mineral Deposits Service)

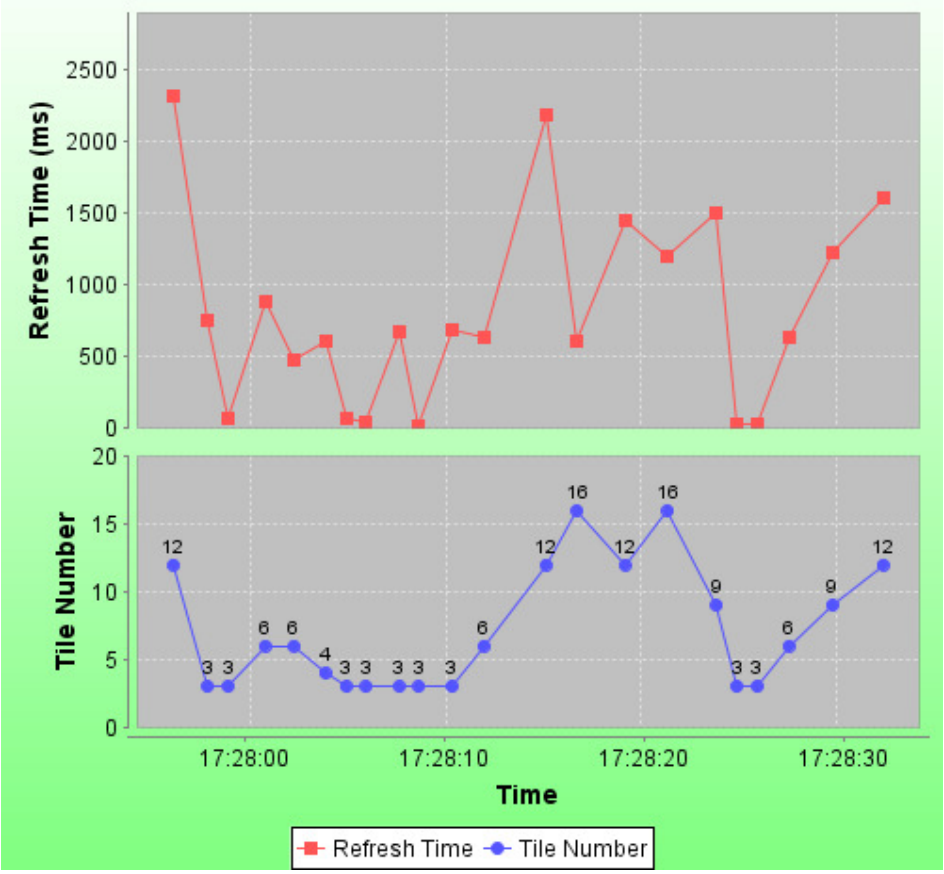

Figure 47: Refresh Time & Requested Tile Number over Time for RAP method (World Mineral Deposits Service)

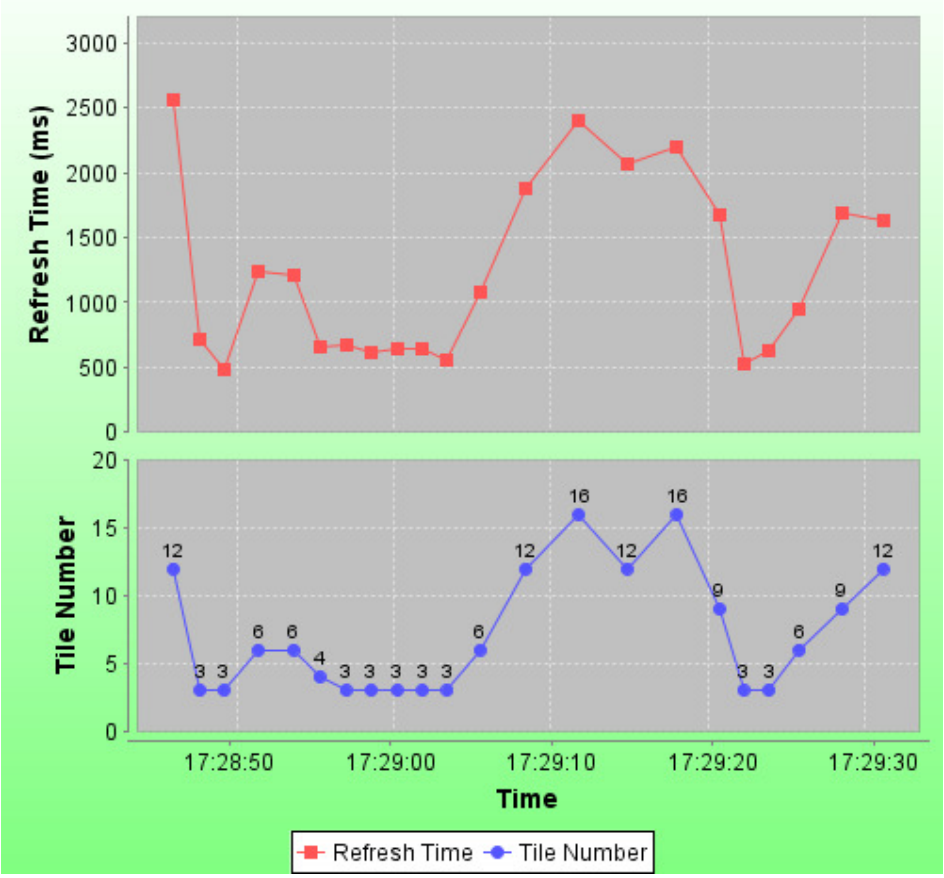

Figure 48: Refresh Time & Requested Tile Number over Time for HCBP method (World Mineral Deposits Service)

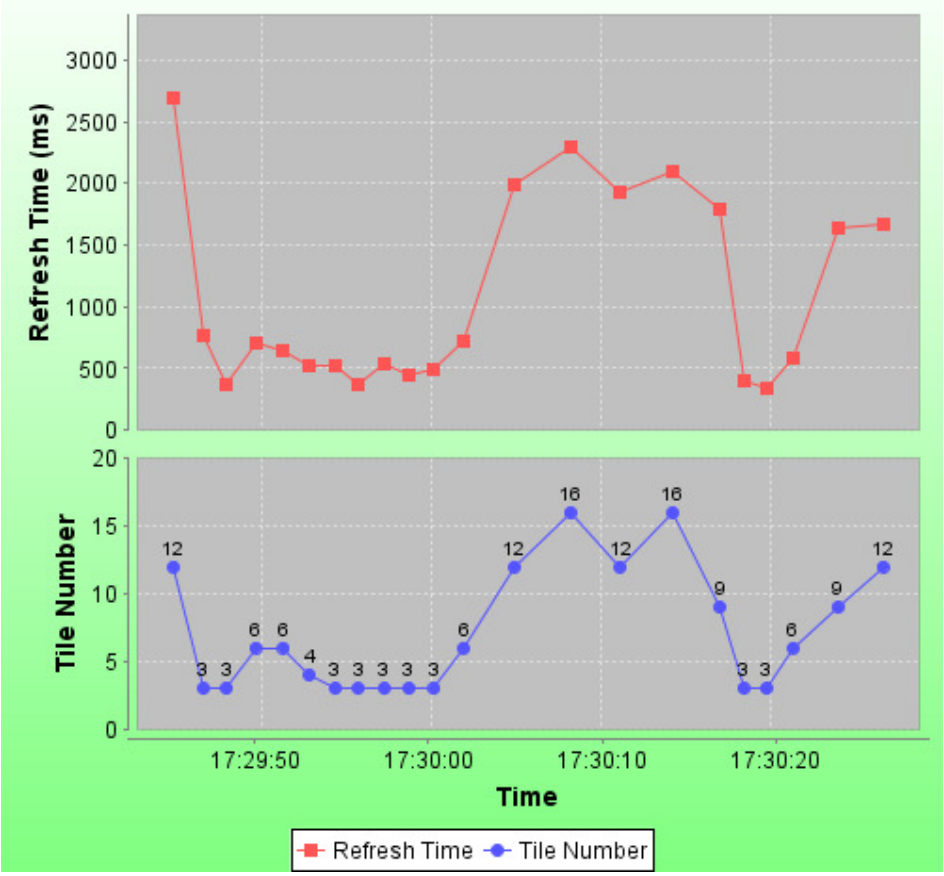

Figure 49: Refresh Time & Requested Tile Number over Time for PKM method (World Mineral Deposits Service)

As in the simulations executed in Geoserver, 2 tables below are generated for the sample scenario. The first columns of the Refresh Time and Used Memory tables are removed from the tables below to fit the content on the page.

Table 13: Refresh Times in sample scenario (World Mineral Deposits Service)

TC	SP	RAP	HCBP	PKM
2269.352390	*4299.678032*	*2319.042784*	*2564.248250*	*2694.162380*
597.391848	1016.520535	757.803017	710.946782	766.293388
646.276552	60.462433	65.507487	487.242373	377.003286
1311.768954	934.605935	881.952595	1234.081147	711.902234
1100.597143	1091.152647	477.113432	1204.818769	643.119233
717.109856	56.850242	604.619302	664.556809	528.334833
723.894797	669.009050	72.311145	678.188149	515.858889
636.629236	27.301235	44.224616	611.543925	364.656548
617.571787	785.227731	674.392696	644.181872	537.122318
572.072708	18.111520	20.342250	647.019942	444.342755
629.978671	30.004093	688.166513	562.514509	499.188220
1174.988721	1772.051755	636.451280	1080.908048	728.496283
1785.750424	3946.307980	2186.442690	1879.763210	1997.523384
2420.872206	4756.840325	604.293004	2401.830680	2291.191420
1668.906802	2924.871457	1454.268629	2065.917316	1933.434216
2368.727589	5300.655251	1195.464786	2199.846350	2101.213129
1502.065207	2900.997194	1509.176725	1678.667261	1788.779855
531.142924	1719.400650	28.222022	525.229883	402.228487
553.846927	23.225578	34.004881	626.989743	332.112412
1023.809730	1229.700422	631.911877	941.669402	587.899543
1678.946067	5245.573948	1228.211406	1686.387516	1633.488844
1762.899754	4781.074131	1604.818490	1634.622557	1667.112194
24025.247900	**39289.944110**	**15399.698840**	**24166.926240**	**20851.301471**

Table 14: Used Memory & Total Memory in sample scenario (World Mineral Deposits Service)

TC	SP	RAP	HCBP	PKM
8,052,264	11,690,552	7,775,432	8,063,672	8,288,345
9,433,992	14,851,640	9,563,424	9,873,752	9,984,433
9,949,704	17,234,680	10,435,352	11,242,416	11,523,232
12,379,128	19,348,672	13,141,648	12,806,624	13,112,579
15,029,744	25,279,528	15,533,688	15,696,592	15,634,452
16,830,200	27,923,368	18,279,776	16,856,760	17,332,456
16,613,472	28,815,864	19,240,128	18,539,248	18,865,386
17,888,088	32,021,232	19,949,024	19,786,736	20,045,666
19,032,224	33,692,128	19,460,248	20,750,600	20,750,600
20,122,088	35,528,248	20,551,504	21,832,432	21,312,552
21,221,936	35,756,992	22,335,216	23,411,216	26,443,216
23,457,336	41,965,832	24,226,264	24,710,728	27,738,839
28,405,400	51,962,016	30,750,304	29,305,456	32,633,285
35,046,200	65,776,784	36,722,504	37,562,240	38,555,438
38,587,768	76,734,704	44,077,072	41,901,064	46,664,438
44,637,800	96,853,968	55,100,688	48,046,592	57,568,832
47,572,104	103,817,752	62,954,672	50,476,696	62,363,317
49,308,112	108,766,048	65,318,616	52,809,184	66,773,522
51,053,504	110,637,040	67,174,824	53,234,688	67,457,433
52,665,992	102,788,160	74,885,160	54,715,024	75,434,567
55,824,240	115,844,760	82,589,664	60,160,144	83,443,216
60,918,576	126,109,360	90,820,512	64,576,608	94,545,433
84,029,440	***182,366,208***	***97,320,960***	***91,283,456***	***98,866,422***

The table below gives the averages of the total refresh times and memory usages for each method at the end of these 100 executions.

Table 15: Average refresh times & memory usage for 100 executions (Geoserver's topp:states)

Averages	TC	SP	RAP	HCBP	PKM
Refresh Time (ms)	23955.125	36449.226	15221.266	22213.554	19975.775
Memory Usage (bytes)	61,443,637	179,655,440	88,543,288	65,403,223	93,716,629

The cache statistics of each method obtained at the end of the test simulations is given in the table below:

Table 16: Cache Statistics (World Mineral Deposits Service)

METHOD	Cache Hit	Cache Miss
TC	0	153
SP	59	94
RAP	77	76
HCBP	14	139
PKM	45	94

The results of the tests give the same conclusions as the tests executed in Geoserver. The memory used in SP method doubles the TC method, because the number of tiles prefetched is exorbitant. In these tests, execution time of SP method is more than the TC method, because of the redundant neighbor tile prefetches during the zoom-in actions of the user. Since both memory usage and refresh times are worse than the first method, this method cannot be chosen as an upgrade to the TC method.

On the other hand RAP method gives better results in performance. Total execution time of the tests is lower than the other methods (63% of TC method). As in the Geoserver test, the memory usage of RAP method is only 1.55 (88,543,288 / 61,443,637) times of TC. This difference is absolutely acceptable according to the

overall performance improvement. So, this method can be chosen as an optimization technique to improve the Web GIS application performance.

As it can be seen from the tables above, HCBP and PKM methods do not provide a better performance improvement than the proposed RAP method. The reason can be concluded from the cache statistics table above. Cache hit ratios of those methods are worse than the proposed method. Namely, for those methods, application made more requests to the WMS server than the proposed method and that caused spending more time to retrieve the tiles of the view extent. On the other hand, memory usage of HCBP method is better than the proposed method. Because, the tiles prefetched and the memory utilized for the execution of the algorithm is less than the proposed method. The average memory consumption of PLM method is more than RAP, but this relatively small difference can be ignored.

Keep in mind that all of these tests explained above are executed on a computer with single core CPU. As mentioned in the previous sections, on a computer with multi-core CPU, the performance of the prefetching methods will increase in direct proportion to the core number. Because, the threads constructed for each prefetch request are distributed to separate cores. 1 thread for neighbor tiles, 1 thread for zoom level and 1 thread for neighbor tiles of the zoom level are created for a prefetching request. For instance, on a quad-core computer, the data obtained from the tests will be 2 to 3 times better for prefetching methods.

5.3 Extreme scenarios

In this section two different scenarios are simulated. In the first scenario, the special navigation set is prepared, so it will give the best performance results for RAP method. On the other hand, the navigation set prepared in the second scenario gives the worst performance results for RAP method. Besides RAP method, both simulations are executed for TC, SP, HCBP and PKM methods as well.

5.3.1 Scenario - 1

In the first scenario, the navigation set prepared as giving the best performance results for RAP method. According to the definition of RAP method, to obtain the best performance the predicted movements shall be correct. Namely, if the user navigates constantly in the same direction, RAP method will prefetch the tiles in that direction too. During this simulation 21 movements to the "east" are executed. The refresh times and memory usages of each method are given in the table below:

Table 17: Total refresh times & memory usages for the first extreme scenario

Total	TC	SP	RAP	HCBP	PKM
Refresh Time (ms)	4925.888	2429.571	1750.873	4853.723	4655.852
Memory Usage (bytes)	36,055,152	53,405,056	35,517,224	36,760,576	36,022,128

As expected, RAP method gives the best performance. The reason can be deduced from the cache statistics. As shown in the table below, cache hit ratio of the RAP method is better than all the methods except SP. Despite the better cache hit numbers, the execution of SP method took longer than RAP method. In SP method, number of tiles prefetched is more than the RAP method and for each tile prefetching, a separate thread is executed. As the number of threads increases the performance of the algorithm begins to decrease. Also, the memory consumption of SP method is more than RAP and that deficiency also reduces the performance.

Table 18: Cache Statistics of the first extreme scenario

METHOD	Cache Hit	Cache Miss
TC	0	78
SP	65	13
RAP	60	18
HCBP	15	63
PKM	25	53

The memory usage and refresh time with tile number graphics over time are given in the figures below:

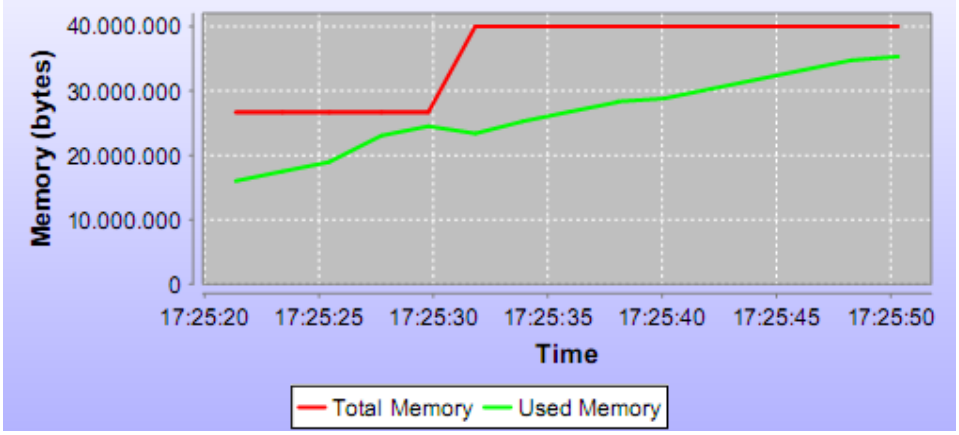

Figure 50: Memory Usage over Time of RAP method for the second extreme scenario

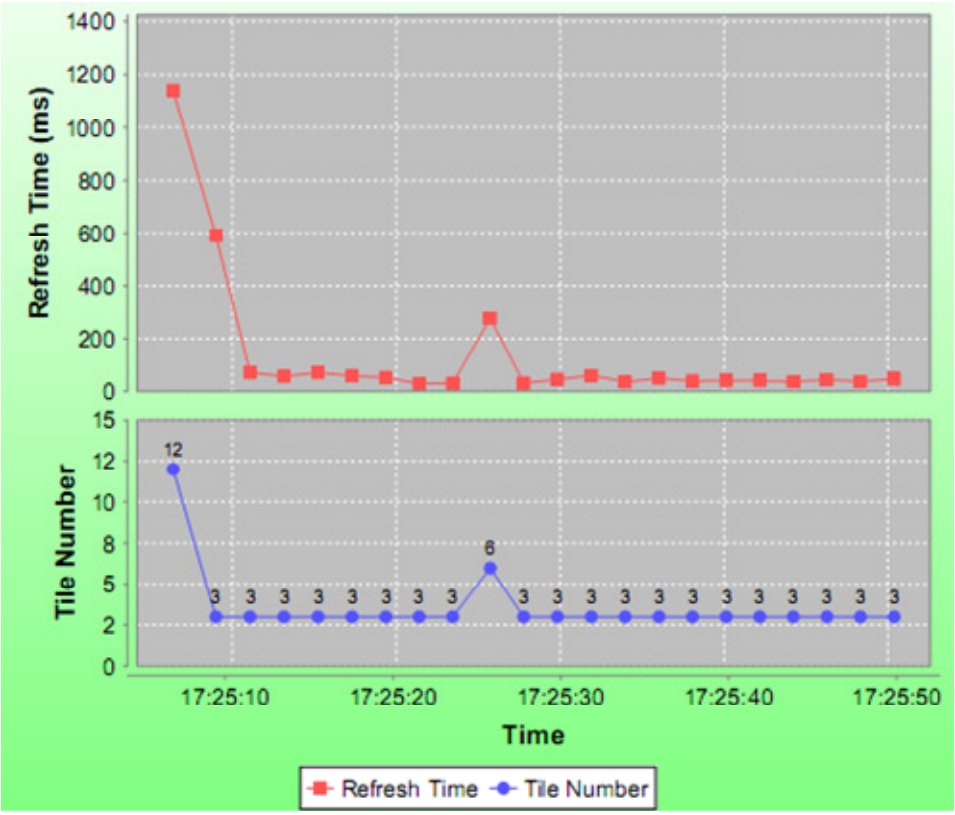

Figure 51: Refresh Time & Requested Tile Number over Time for RAP method for the second extreme scenario

5.3.2 Scenario - 2

In the second scenario, the navigation set prepared as giving the worst performance results for RAP method. To obtain the worst performance, the navigations are selected as not forming a pattern to make the predictions accurate. Namely, if the movements in the history list are all unique during the simulation, the tiles prefetched by RAP method will not reflect the user's next movement entirely. The navigation steps selected for this simulation is given in the table below:

Table 19: Navigation Summary of the second extreme scenario

Navigation	Zoom Level	Center Latitude	Center Longitude
Initial	11	40.00000	-100.00000
EAST	11	40.00000	-94.00000
NORTH	11	42.81194	-94.00000
WEST	11	42.81194	-100.00000
SOUTH	11	40.00000	-100.00000
ZOOM IN	10	40.00000	-100.00000
WEST	10	40.00000	-103.00000
SOUTH	10	38.54861	-103.00000
EAST	10	38.54861	-100.00000
NORTH	10	40.00000	-100.00000
ZOOM OUT	11	40.00000	-100.00000
EAST	11	40.00000	-94.00000
NORTH	11	42.81194	-94.00000
WEST	11	42.81194	-100.00000
SOUTH	11	40.00000	-100.00000
ZOOM IN	10	40.00000	-100.00000
WEST	10	40.00000	-103.00000
SOUTH	10	38.54861	-103.00000
EAST	10	38.54861	-100.00000
NORTH	10	40.00000	-100.00000
ZOOM OUT	11	40.00000	-100.00000

Table 20: Total refresh times & memory usages for the second extreme scenario

Total	TC	SP	RAP	HCBP	PKM
Refresh Time (ms)	1867.835	2490.779	2683.314	2622.497	2646.642
Memory Usage (bytes)	23,212,032	51,601,408	36,753,408	30,896,128	39,154,903

As expected, RAP method gave worse performance than TC, because no cache hit occurred in the prefetching cache. Hence, the tile prefetching processes and the memory consumption caused reduction in the performance. However, for the same scenario and for the same reason, HCBP and PKM did not give a much better results either.

Table 21: Cache Statistics of the second extreme scenario

METHOD	Cache Hit	Cache Miss
TC	0	36
SP	20	16
RAP	0	36
HCBP	1	35
PKM	1	35

The memory usage and refresh time with tile number graphics over time are given in the figures below:

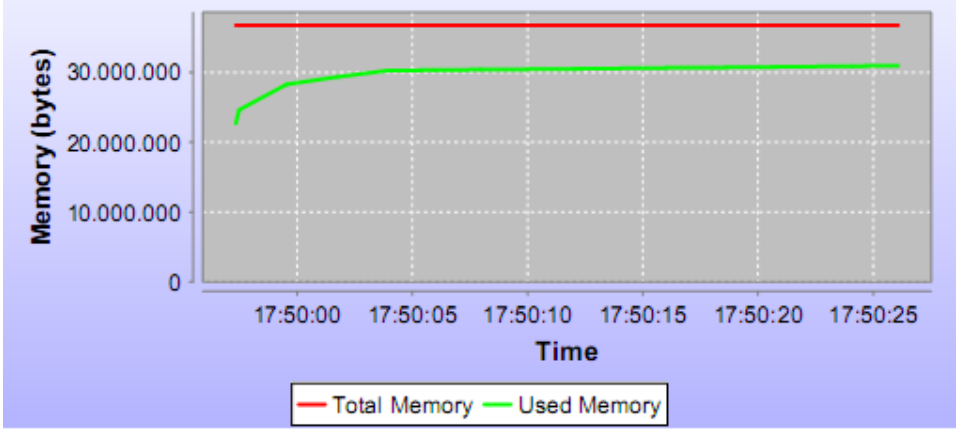

Figure 52: Memory Usage over Time of RAP method for the second extreme scenario

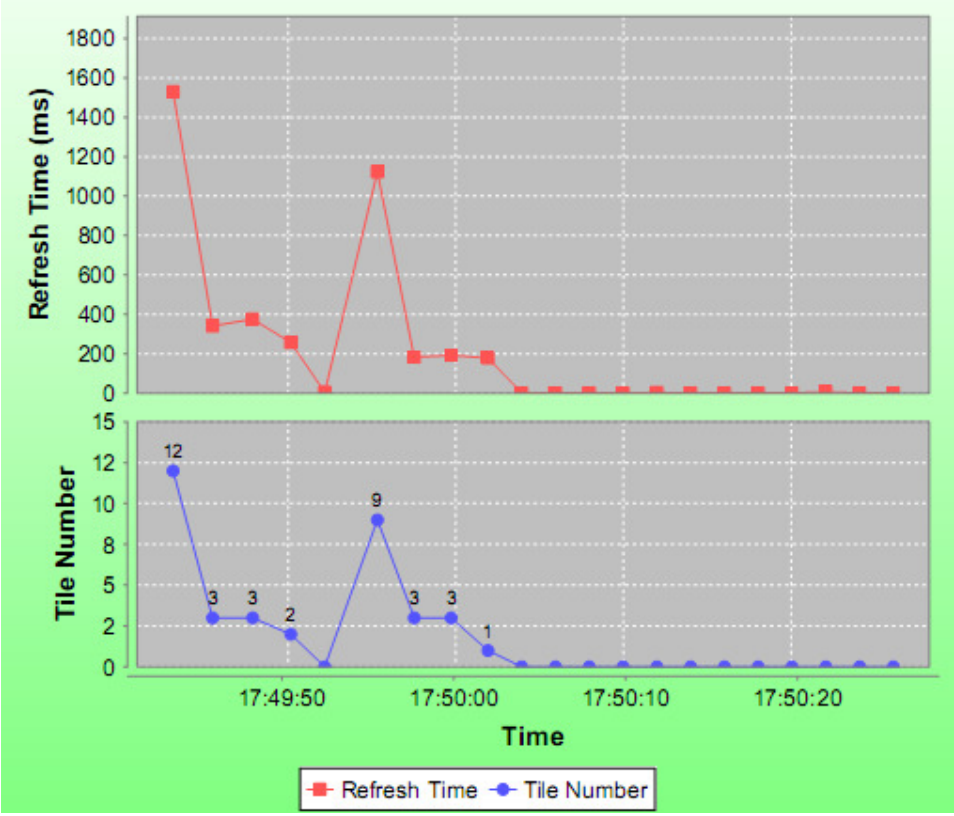

Figure 53: Refresh Time & Requested Tile Number over Time for RAP method for the second extreme scenario

CHAPTER 6

CONCLUSION AND FUTURE WORK

In this study, a heuristic prefetching algorithm is proposed. 3 alternative optimization methods are analyzed and compared with this proposed algorithm by stating the advantages and disadvantages of each method. The proposed algorithm utilizes a heuristic method which tries to predict the next moves of the user and finds out the locations of the candidate tiles to be prefetched. Then these tiles are requested from the WMS server and kept ready for the future navigations.

The accuracy of the prediction formula of the proposed algorithm is determined with the number of the moves (depth) that will be kept in history. When depth is 5, it means that 5 moves before the current navigation action are taken into consideration while predicting the tiles to be prefetched. Since more previous data provides more information about the navigational behavior of the user, to obtain an accurate prefetching result, keeping the depth of the history a higher value will be a wise choice. However, more depth also means more calculations and more memory for keeping the historical data. So, it is always advised to constrain the depth value to an optimum value like 5 to 10.

The next possible moves of the client are computed by giving weights to each previous move in the history and calculating the average optimized value for the tiles predicted. The weights of the previous moves are generated by using the cache statistics. The contribution of each former move to the accurate prefetches is reflected to the weight of that move. By this way, the effect of a move in the history changes dynamically through the execution of the application.

The memory allocated for the prefetching cache is a configurable parameter. User can either give the capacity of the cache as number of tiles or leave the decision to the application. The memory capacity decision algorithm simply subtracts the

currently used memory from the maximum memory reserved for the application and determines the number of tiles can be kept in the remaining amount of memory. The result is the capacity of the cache. When the application is out of memory, %75 of the cache is cleared. %75 of the data to be removed from the cache contains the oldest tiles in the cache.

To execute the implementation of the methods, a test application is developed. This test application automates the tests that generate the performance results of each method. During the tests, no user interaction is required, because the entire process of the simulation is fully automated. The parameters like selected WMS server, navigation steps, selected cache and the parameters of the selected WMS servers (such as layer, initial coordinate and zoom level) are all configurable and can be entered via a separate application.

After the simulations, the performance and memory usage of each method are generated for 2 different WMS servers; one local (same computer client application operates on) and one remote (a free WMS service on internet). Each method is compared with each other and according to the results; the proposed heuristic prefetching method is appeared to give the best overall performance among the other three compared methods.

As the next step of this method, the time interval between each navigation may be considered as a factor for evaluating the value of navigation in the history list. From assumption of “frequent consecutive movements are likely to happen again”, it may be conclude that as the frequency of a movement increases, the weight of that movement in the history may be updated as an improvement.

Also more heuristic approaches can be adapted to the prefetching algorithm proposed in this study to gain more accurate solutions. There are many heuristic optimization algorithms such as Simulated Annealing, Markov Chain, and Genetic Algorithms which can be adapted to this problem to predict the next candidate moves of the user while navigating on a map.

REFERENCES

[1] OGC® Web Feature Service Implementation Specification 1.1.0 (OGC 04-094), Ed. Panagiotis A. Vretanos, (May 3, 2005). Retrieved from http://portal.opengeospatial.org/files/?artifact_id=8339.

[2] OGC® Web Coverage Service Implementation Standard 1.1.2 (OGC 07-067r5), Ed. Arliss Whiteside, John D. Evans, (March 19, 2008). Retrieved from http://portal.opengeospatial.org/files/?artifact_id=27297.

[3] OGC® Web Map Service Interface 1.3.0 (OGC 03-109r1), Ed. Jeff de la Beaujardiere, (January 20, 2004). Retrieved from http://portal.opengeospatial.org/files/?artifact_id=4756.

[4] OpenGIS® Tiled WMS Discussion Paper 0.3.0 (OGC 07-057r2), Ed. Keith Pomakis, (August 14, 2007). Retrieved from http://portal.opengeospatial.org/files/?artifact_id=23206.

[5] OpenGIS® Geography Markup Language (GML) Encoding Standard 3.2.1 (OGC 07-036), Ed. Clemens Portele, (August 27, 2007). Retrieved from http://portal.opengeospatial.org/files/?artifact_id=20509.

[6] OGC® Binary Extensible Markup Language (BXML) Encoding Specification 0.0.8 (OGC 03.002r9), Ed. Craig Bruce, (January 13, 2006). Retrieved from http://portal.opengeospatial.org/files/?artifact_id=13636.

[7] Jay Ratcliff, Kevin Shaw, Shengru Tu, Maik Flanagin, Ying Wu, Mahdi Abdelguerfi, Eric Normand, Venkata Mahadevan: "*Design Strategies to Improve Performance of GIS Web Services"*, Proceedings of the International Conference on Information Technology, Coding and Computing (ITCC'04) vol. 2, pp. 444-448, ISBN: 0-7695-2108-8, 5-7 April 2004

[8] C.S. Baptista, C.P. Nunes, A.G. de Sousa, E.R. da Silva, F.L. Leite Jr., and A.C. de Paiva: "*On Performance Evaluation of Web GIS Applications"*, Proceedings of the IEEE Sixteenth International Workshop on Database and Expert Systems Applications, pp. 497-501, ISBN: 0-7695-2424-9, 2005

[9] Arron Walker, Binh Pham, Anthony Maeder, "*A Bayesian framework for automated dataset retrieval in Geographic Information Systems*", Proceedings of the

10th International Multimedia Modeling Conference, pp. 138-144, ISBN: 0-7695-2084-7, 5-7 January 2004.

[10] Haixia Zhao, Ben Schneiderman, "*Image-based highly interactive Web mapping for geo-referenced data publishing*", (Report No. HCIL-2002-26, CS-TR-4431, UMIACS-TR-2003-02), December 8, 2002

[11] P.D. Coddington, K.A. Hawick, and H.A. James, "*Web-Based Access to Distributed High Performance Geographic Information Systems for Decision Support*", Proceedings of the 32nd Hawaii International Conference on System Sciences, vol. 6, pp. 6015. ISBN: 0-7695-0001-3, 1999

[12] Neville Churcer, "*Applications of Distortion-Oriented Presentation Techniques in GIS*",Presented at AURISA/SIRC '95-The 7th Colloquim of the Spatial Information Research Centre, University of Otago, in association with AURISA New Zealand and Massey University, 26-28 April, 1995

[13] Dong-Joo Park, Hyoung-Joo Kim, "*Prefetch Policies for Large Objects in a Web-enabled GIS Application*", Data & Knowledge Engineering, vol. 37, pp. 65-84, ISSN: 0169-023X, April 2001

[14] Dong Ho Lee, Jung Sup Kim, Soo Duk Kim, Ki Chang Kim, Yoo-Sung Kim, and Jaehyun Park, "*Adaptation of a Neighbor Selection Markov Chain for Prefetching Tiled Web GIS Data*", Proceedings of the Second International Conference on Advances in Information Systems, vol. 2457, pp. 213-222. ISBN: 3-540-00009-7, 2002

[15] Wook-Shin Han, Woong-Kee Loh, and Kyu-YoungWhang, "*Type-Level Access Pattern View: A Technique for Enhancing Prefetching Performance*", Proceedings of Database Systems for Advanced Applications, 11th International Conference, DASFAA 2006, Singapore, vol. 3882, pp. 389-403, April 12-15, 2006

[16] Han, W.-S., Whang, K.-Y., and Moon, Y.-S., "PrefetchGuide: Capturing Navigational Access Patterns for Prefetching in Client/Server Object-Oriented/Object-Relational DBMSs", Information Sciences, vol. 152, pp. 47-61, June 2003

[17] Myung-Hee Jo, Yun-Won Jo, Jeong-Soo Oh, Si-Young Lee, "*The Design and Implementation of Dynamic Load Balancing for Web-Based GIS Services*", Proceedings of the 22th Asian Conference on Remote Sensing, 5-9 November 2001

[18] Şafak B. Çevikbaş, "*Visibility Based Prefetching with Simulated Annealing*", Ms. Thesis, Middle East Technical University, Turkey, January 2008

[19] Eric Poupaert and Yves Deville, "*Simulated Annealing with estimated temperature*", Special issue on AI research in the Benelux, vol. 13, pp. 19-26, ISSN: 0921-7126, October 2000

[20] Public OGC® WMS server list. Last accessed January 2010, from http://www.skylab-mobilesystems.com/en/wms_serverlist.html

[21] WMS Capabilities document of the WMS Server maintained by Mineral Resources Division, Geological Survey of Canada. Last accessed January 2010, from http://apps1.gdr.nrcan.gc.ca/cgi-bin/worldmin_en-ca_ows?request=GetCapabilities

[22] Legend information of the map provided by Mineral Resources Division, Geological Survey of Canada. Last accessed January 2008, from http://apps1.gdr.nrcan.gc.ca/cgi-bin/worldmin_en-ca_ows?version=1.1.1&service=WMS&request=GetLegendGraphic&layer=GSC:WORLD_AgeRockDomain&format=image/png

APPENDIX - A

Pseudo Code of the Proposed Algorithm

This study proposes a prefetching algorithm that optimizes the performance of a web GIS application. The proposed algorithm predicts the navigation behavior of the user and tries to predict the locations of the tiles that form the next probable view of the application via heuristic calculations. In this section pseudo-code of this heuristic prefetching algorithm is presented.

```
CACHE PREFETCHING_CACHE, MAIN_CACHE
MAP<STRING, MAP<NAVIGATION, ARRAY<Integer>>> PRIORITIES_MAP
LIST<TileLocation> TILES_TO_PREFETCH, TILES_TO_PREFETCH_FOR_ZOOM
LIST<TileLocation> HISTORY
INTEGER DEPTH, VIEW_WIDTH, VIEW_HEIGHT
BOOLEAN PREFETCHER_READY
TileLocation TOP_LEFT_TILE
```

centerTileChanged method is called when the center tile of the map changes as a result of a user action. This method resets the top-left tile of the view extent, and the number of horizontal and vertical tiles visible to user. Then the locations of the tiles that will be prefetched are calculated. If there is at least one candidate tile to prefetch, prefetcher's status is set to ready.

```
FUNCTION centerTileChanged( TileLocation topLeftTile, INTEGER
viewWidth, INTEGER viewHeight) {
TOP_LEFT_TILE := topLeftTile
VIEW_WIDTH := viewWidth
VIEW_HEIGHT := viewHeight
findTilesToPrefetch()
PREFETCHER_READY := TILES_TO_PREFETCH.isNotEmpty() ||
TILES_TO_PREFETCH_FOR_ZOOM.isNotEmpty()
}
```

findTilesToPrefetch is the method where the locations of the tiles to be prefetched are calculated. The former moves in the history are compared to each other in order. The easting and southing differences are graded according to the cache hit ratio of each move in history. Zooming value is calculated according to the zooming differences of each former move by ranking each of them by their zooming factor. Hit ratios and zooming factor of each former move in history are calculated in *navigationChanged* method. Once easting, southing and zooming values are calculated, they are normalized to obtain the number of tiles to prefetch. Normalized values are used for determining the direction of the prefetching and the locations of each candidate tiles.

```
FUNCTION findTilesToPrefetch () {
TILES_TO_PREFETCH.clear()
TILES_TO_PREFETCH_FOR_ZOOM.clear()
IF HISTORY.size <= 1 THEN
      RETURN
ENDIF

DOUBLE east := south := zoom := 0
INTEGER totalEast := totalSouth := 0
HistoryData lastHistory := HISTORY.getLastElement()
TileLocation lastNavigation := lastHistory.tileLocation
FOR i FROM historySize - 2 TO 0 DECREASE BY 1 DO
      HistoryData exHistory := HISTORY.get(i)
      TileLocation exTileLocation = exHistory.tileLocation
      DOUBLE hitRatio := exHistory.hitRatio
      INTEGER diffEast := lastNavigation.x - exTileLocation.x
      INTEGER diffSouth := lastNavigation.y - exTileLocation.y
      east += diffEast * hitRatio
      south += diffSouth * hitRatio
      totalEast += diffEast
      totalSouth += diffSouth
      zoom += (lastNavigation.zoom - exTileLocation.zoom) *
exHistory.zoomFactor
      lastNavigation = exTileLocation
      lastHistory = exHistory
ENDFOR
```

```
INTEGER easting :=
      IF totalEast == 0 THEN 0 ELSE ROUND(east / totalEast) ENDIF
INTEGER southing :=
      IF totalSouth == 0 THEN 0 ELSE ROUND(south / totalSouth) ENDIF
INTEGER zooming :=
      IF ROUND(zoom) == 0 THEN 0 ELSE (IF ROUND(zoom) < 0 THEN -1
ELSE 1 ENDIF) ENDIF

LIST<TileLocation> tilesOnEdges := getTilesOnEdges()
NAVIGATION prefetchDirection := getDirectionType(easting, southing)
IF prefetchDirection != NAVIGATION.NONE THEN
      ARRAY<NAVIGATION> navigations :=
getPrefetchNavigations(easting, southing)
      ARRAY<INTEGER> tileIndicies :=
PRIORITIES_MAP.get(VIEW_WIDTH+"X"+VIEW_WIDTH).get(prefetchDirection)
      FOREACH tileIndex IN tileIndicies DO
            TileLocation pivotTile := tilesOnEdges.get(tileIndex)
            FOREACH navigation IN prefetchNavigations DO
                  IF navigation == NAVIGATION.NONE THEN
                        EXIT_FOREACH_LOOP
                  ENDIF
                  TileLocation tileToAdd := NEW TileLocation WITH
            x:= pivotTile.x + navigation.x , y:= pivotTile.y +
            navigation.y , zoom:= pivotTile.zoom
                  IF tileToAdd NOT_IN (TILES_TO_PREFETCH AND
tilesOnEdges)     THEN
                        TILES_TO_PREFETCH.add(tileToAdd)
                  ENDIF
            ENDFOREACH
      ENDFOREACH
ENDIF

IF zooming != 0 THEN
      INTEGER absZoom := absolute(zooming)
      INTEGER stepZoom := absZoom / zooming
      DOUBLE sign := IF zooming > 0 THEN 0.5 ELSE 2 ENDIF
      TileLocation centerTile := NEW TileLocation
            WITH x:= TOP_LEFT_TILE.x + VIEW_WIDTH / 2, y:=
      TOP_LEFT_TILE.y + VIEW_HEIGHT / 2, zoom:= TOP_LEFT_TILE.zoom
```

```
      FOR i FROM 0 TO absZoom BY 1 DO
            centerTile := zoomTile(centerTile, sign, stepZoom)
            TILES_TO_PREFETCH_FOR_ZOOM.add(centerTile)
      ENDFOR
ENDIF
}
```

centerTileChanged method finds the location of the tile which is "*zoom*" number of levels in/out of the *centerTile*. *sign* determines whether zoom in or out.

```
FUNCTION zoomTile ( TileLocation centerTile, DOUBLE sign, INTEGER
zoom ) RETURNS TileLocation{

TileLocation tile := NEW TileLocation WITH x:= centerTile.x * sign,
y:= centerTile.y * sign, zoom:= centerTile.zoom + stepZoom
RETURN tile
}
```

The method for finding out the tiles on the edge of the view extent is *getTilesOnEdges*. Via this method, tiles in the center of the view extent are ignored during prefetching process, because the neighbor tiles of those tiles are the tiles on the edges of the view extent. Namely, the prefetching candidates for the tiles on center are already present on view extent. (For prefetching zoom levels, center tile of the view extent is calculated separately.)

```
FUNCTION getTilesOnEdges( ) RETURNS LIST<TileLocation> {

INTEGER tpx := TOP_LEFT_TILE.x
INTEGER tpy := TOP_LEFT_TILE.y
INTEGER zoom := TOP_LEFT_TILE.zoom
LIST<TileLocation> tiles
INTEGER x := tpx
INTEGER y := tpy
WHILE  x < tpx + VIEW_WIDTH DO
      tiles.add(NEW TileLocation WITH x:=x++, y:=y, zoom:= zoom)
ENDWHILE
x--
y++
```

```
WHILE  y < tpy + VIEW_HEIGHT DO
      tiles.add(NEW TileLocation WITH x:=x, y:=y++, zoom:= zoom)
ENDWHILE
y--
x--
WHILE  x >= tpx DO
      tiles.add(NEW TileLocation WITH x:=x--, y:=y, zoom:= zoom)
ENDWHILE
x++
y--
WHILE  y > tpy DO
      tiles.add(NEW TileLocation WITH x:=x, y:=y--, zoom:= zoom)
ENDWHILE
RETURN tiles
}
```

When *navigationChanged* method is called, it means user stop that navigation action. At this point, prefetching threads are terminated, because a new navigation is on the way for restarting another prefetching process. Also, in this method, hits and misses in prefetching cache for the latest navigation are saved in the history with the navigation direction.

```
FUNCTION navigationChanged (NavigationEvent event) {
IF  event.type == NavigationEvent.NAVIGATION_CHANGED THEN
      STOP_PREFETCHING_THREADS
      INTEGER hits = cacheStats.hits
      INTEGER misses = cacheStats.misses
      addNavigationToHistory(event.navigation, hits , misses)
ENDIF
}
```

addNavigationToHistory method saves the latest navigation in the history. The hit ratios and zooming factor of the remaining data in history are updated with the updated cache statistics and navigation information.

```
FUNCTION addNavigationToHistory(NAVIGATION navigation, INTEGER hits,
INTEGER misses) {
IF  HISTORY.size == 0 THEN
```

```
      HISTORY.addAtEnd(NEW HistoryData WITH (tileLocation:=NEW
TileLocation WITH x:=0, y:=0, zoom:=0))
ELSE IF  historySize >= DEPTH THEN
      HISTORY.removeFirst()
ENDIF
INTEGER zoomFactor := 0
FOREACH historyData IN HISTORY DO
      historyData.hits += hits
      historyData.misses += misses
      IF  HISTORY.size >= DEPTH THEN
            INTEGER total := historyData.hits + historyData.misses
            historyData.hitRatio :=  IF  total == 0 THEN 1 ELSE hits
/ total ENDIF
      ENDIF
      historyData.zoomFactor := zoomFactor++ / HISTORY.size
ENDFOREACH
HistoryData last = HISTORY.getLastElement()
HistoryData historyData = NEW HistoryData WITH (tileLocation:=NEW
TileLocation x:= navigation.x + last.x, y:=navigation.y + last.y,
zoom:=navigation. zoom + last.zoom)
HISTORY.addAtEnd(historyData)
}
```

getTileImage is the method for retrieving the image data of the tile requested to construct the view extent. According to this pivot tile, prefetching threads are started for predicting the next move of the user. On the other hand, the image of the pivot tile is looked up in the prefetching and main cache respectively. If it is found in any of the caches, it is returned to the calling method.

```
FUNCTION getTileImage(TileLocation pivotTile) RETURNS BufferedImage
{
IF  isPrefetcherReady() THEN
      START_THREAD_FUNCTION : prefetch(pivotTile)
      START_THREAD_FUNCTION : prefetchForZoom(pivotTile)
      START_THREAD_FUNCTION : prefetchForZoom2(pivotTile)
ENDIF
BufferedImage prefetch = PREFETCHING_CACHE.get(pivotTile);
IF prefetch != null THEN
```

```
      RETURN prefetch;
ENDIF
RETURN MAIN_CACHE.get(pivotTile);
}
```

To start a prefetching thread, locations of the tiles to prefetch shall be calculated. *isReady* method check whether the prefetcher is ready or not.

```
FUNCTION isReady() RETURNS BOOLEAN {
      BOOLEAN ready := PREFETCHER_READY
      PREFETCHER_READY := FALSE
      RETURN ready
}
```

prefetch, prefetchForZoom and *prefetchForZoom2* methods add each predicted tile calculated in *findTilesToPrefetch* method to the prefetching cache.

```
FUNCTION prefetch() {
FOREACH tileToPrefetch IN TILES_TO_PREFETCH) DO
      addTileToCache(tileToPrefetch.x, tileToPrefetch.y, zoom);
ENDFOREACH
}

FUNCTION prefetchForZoom() {
FOREACH tileToPrefetch IN TILES_TO_PREFETCH_FOR_ZOOM DO
      addZoomLevels(tileToPrefetch.x, tileToPrefetch.y, zoom);
ENDFOREACH
}
```

prefetchForZoom2 method adds the neighbor tiles of the zooming tiles to be prefetched. If those tiles will not be visible in the view extent, there is no need to make tile requests to WMS server. By this way, bandwidth is used more efficiently by eliminating redundant requests. If the horizontal tile number or vertical tile number is greater than 3, it means neighbor tiles of the zooming tiles stay in the view extent. The first “if” statement in the method is added to check this condition.

```
FUNCTION prefetchForZoom2() {
```

```
IF ((VIEW_WIDTH > 3 || VIEW_HEIGHT > 3)) THEN
      FOREACH tileToPrefetch IN TILES_TO_PREFETCH_FOR_ZOOM DO
            addZoomLevels2(tileToPrefetch.x, tileToPrefetch.y,
zoom);
      ENDFOREACH
ENDIF
}
```

APPENDIX - B

Steps of the Test Simulation

NAVIGATION – 1

Direction -> EAST

Top-Left Tile -> x=14, y=23, zoom=11

HISTORY		EAST	SOUTH	ZOOM	HIT RATIO	ZOOM FACTOR
FROM	TO					
x=0, y=0, zoom=0	x=1, y=0, zoom=0	1.0	0.0	0.0	1.0	1.0
x=0, y=0, zoom=0	x=0, y=0, zoom=0	0.0	0.0	0.0	1.0	1.0

Easting = 1 , Southing = 0 , Zooming = 0

TILES TO PREFETCH	
TYPE	TILE LOCATION
NEIGHBOR	x=17, y=24, zoom=11
NEIGHBOR	x=17, y=23, zoom=11
NEIGHBOR	x=17, y=25, zoom=11

NAVIGATION – 2

Direction -> EAST

Top-Left Tile -> x=15, y=23, zoom=11

HISTORY		EAST	SOUTH	ZOOM	HIT RATIO	ZOOM FACTOR
FROM	TO					
x=1, y=0, zoom=0	x=2, y=0, zoom=0	1.0	0.0	0.0	1.0	1.0
x=1, y=0, zoom=0	x=0, y=0, zoom=0	1.0	0.0	0.0	1.0	1.0
x=0, y=0, zoom=0	x=0, y=0, zoom=0	0.0	0.0	0.0	1.0	0.666

Easting = 1 , Southing = 0 , Zooming = 0

TILES TO PREFETCH	
TYPE	TILE LOCATION
NEIGHBOR	x=18, y=24, zoom=11
NEIGHBOR	x=18, y=23, zoom=11
NEIGHBOR	x=18, y=25, zoom=11

NAVIGATION – 3

Direction -> SOUTH EAST

Top-Left Tile -> x=16, y=23, zoom=11

HISTORY		EAST	SOUTH	ZOOM	HIT RATIO	ZOOM FACTOR
FROM	TO					
x=2, y=0, zoom=0	x=3, y=1, zoom=0	1.0	1.0	0.0	1.0	1.0
x=1, y=0, zoom=0	x=2, y=0, zoom=0	1.0	0.0	0.0	1.0	1.0
x=1, y=0, zoom=0	x=0, y=0, zoom=0	1.0	0.0	0.0	1.0	0.75
x=0, y=0, zoom=0	x=0, y=0, zoom=0	0.0	0.0	0.0	1.0	0.5

Easting = 1 , Southing = 1 , Zooming = 0

TILES TO PREFETCH	
TYPE	TILE LOCATION
NEIGHBOR	x=19, y=27, zoom=11
NEIGHBOR	x=19, y=26, zoom=11
NEIGHBOR	x=18, y=27, zoom=11
NEIGHBOR	x=17, y=27, zoom=11
NEIGHBOR	x=19, y=25, zoom=11
NEIGHBOR	x=16, y=27, zoom=11
NEIGHBOR	x=19, y=24, zoom=11
NEIGHBOR	x=19, y=23, zoom=11

NAVIGATION – 4

Direction -> SOUTH EAST

Top-Left Tile -> x=17, y=24, zoom=11

HISTORY		EAST	SOUTH	ZOOM	HIT RATIO	ZOOM FACTOR
FROM	TO					
x=3, y=1, zoom=0	x=4, y=2, zoom=0	1.0	1.0	0.0	1.0	1.0
x=2, y=0, zoom=0	x=3, y=1, zoom=0	1.0	1.0	0.0	1.0	1.0
x=1, y=0, zoom=0	x=2, y=0, zoom=0	1.0	0.0	0.0	1.0	0.8
x=1, y=0, zoom=0	x=0, y=0, zoom=0	1.0	0.0	0.0	1.0	0.6
x=0, y=0, zoom=0	x=0, y=0, zoom=0	0.0	0.0	0.0	1.0	0.4

Easting = 1 , Southing = 1 , Zooming = 0

TILES TO PREFETCH	
TYPE	TILE LOCATION
NEIGHBOR	x=20, y=27, zoom=11
NEIGHBOR	x=20, y=26, zoom=11
NEIGHBOR	x=19, y=27, zoom=11
NEIGHBOR	x=20, y=25, zoom=11
NEIGHBOR	x=18, y=27, zoom=11
NEIGHBOR	x=20, y=24, zoom=11
NEIGHBOR	x=17, y=27, zoom=11

NAVIGATION – 5

Direction -> EAST

Top-Left Tile -> x=18, y=24, zoom=11

HISTORY		EAST	SOUTH	ZOOM	HIT RATIO	ZOOM FACTOR
FROM	TO					
x=4, y=2, zoom=0	x=5, y=2, zoom=0	1.0	0.0	0.0	1.0	1.0
x=3, y=1, zoom=0	x=4, y=2, zoom=0	0.6	0.6	0.0	0.6	0.833
x=2, y=0, zoom=0	x=3, y=1, zoom=0	0.777	0.777	0.0	0.777	0.666
x=1, y=0, zoom=0	x=2, y=0, zoom=0	0.412	0.0	0.0	0.412	0.5
x=1, y=0, zoom=0	x=0, y=0, zoom=0	0.412	0.0	0.0	0.412	0.333

Easting = 1 , Southing = 1 , Zooming = 0

TILES TO PREFETCH	
TYPE	TILE LOCATION
NEIGHBOR	x=21, y=27, zoom=11
NEIGHBOR	x=21, y=26, zoom=11
NEIGHBOR	x=20, y=27, zoom=11
NEIGHBOR	x=21, y=25, zoom=11
NEIGHBOR	x=19, y=27, zoom=11
NEIGHBOR	x=21, y=24, zoom=11
NEIGHBOR	x=18, y=27, zoom=11

NAVIGATION – 6

Direction -> SOUTH EAST

Top-Left Tile -> x=19, y=25, zoom=11

HISTORY		EAST	SOUTH	ZOOM	HIT RATIO	ZOOM FACTOR
FROM	TO					
x=5, y=2, zoom=0	x=6, y=3, zoom=0	1.0	1.0	0.0	1.0	1.0
x=4, y=2, zoom=0	x=5, y=2, zoom=0	1.0	0.0	0.0	1.0	0.833
x=3, y=1, zoom=0	x=4, y=2, zoom=0	0.636	0.636	0.0	0.636	0.666
x=2, y=0, zoom=0	x=3, y=1, zoom=0	0.789	0.789	0.0	0.789	0.5
x=1, y=0, zoom=0	x=2, y=0, zoom=0	0.428	0.0	0.0	0.428	0.333

Easting = 1 , Southing = 1 , Zooming = 0

TILES TO PREFETCH	
TYPE	TILE LOCATION
NEIGHBOR	x=22, y=28, zoom=11
NEIGHBOR	x=22, y=27, zoom=11
NEIGHBOR	x=21, y=28, zoom=11
NEIGHBOR	x=22, y=26, zoom=11
NEIGHBOR	x=20, y=28, zoom=11
NEIGHBOR	x=22, y=25, zoom=11
NEIGHBOR	x=19, y=28, zoom=11

NAVIGATION – 7 & 8

Direction -> SOUTH & SOUTH

No prefetching is done in these steps, because center tile, in another words view extent, did not change according to this action. The tiles needed to set the view extent are obtained from the prefetch cache filled in previous steps.

NAVIGATION – 9

Direction -> SOUTH EAST

Top-Left Tile -> x=20, y=27, zoom=11

HISTORY		EAST	SOUTH	ZOOM	HIT RATIO	ZOOM FACTOR
FROM	TO					
x=6, y=5, zoom=0	x=7, y=6, zoom=0	1.0	1.0	0.0	1.0	0.0
x=6, y=4, zoom=0	x=6, y=5, zoom=0	0.0	1.0	0.0	1.0	0.833
x=6, y=3, zoom=0	x=6, y=4, zoom=0	0.0	1.0	0.0	1.0	0.666
x=5, y=2, zoom=0	x=6, y=3, zoom=0	1.0	1.0	0.0	1.0	0.5
x=4, y=2, zoom=0	x=5, y=2, zoom=0	1.0	0.0	0.0	1.0	0.333

Easting = 1 , Southing = 1 , Zooming = 0

TILES TO PREFETCH	
TYPE	TILE LOCATION
NEIGHBOR	x=23, y=30, zoom=11
NEIGHBOR	x=23, y=30, zoom=11
NEIGHBOR	x=22, y=30, zoom=11
NEIGHBOR	x=23, y=28, zoom=11
NEIGHBOR	x=21, y=30, zoom=11
NEIGHBOR	x=23, y=27, zoom=11
NEIGHBOR	x=20, y=30, zoom=11

NAVIGATION – 10

Direction -> EAST

Top-Left Tile -> x=21, y=27, zoom=11

HISTORY		EAST	SOUTH	ZOOM	HIT RATIO	ZOOM FACTOR
FROM	TO					
x=7, y=6, zoom=0	x=8, y=6, zoom=0	0.4	0.0	0.0	0.4	1.0
x=6, y=5, zoom=0	x=7, y=6, zoom=0	0.4	0.4	0.0	0.4	0.833
x=6, y=4, zoom=0	x=6, y=5, zoom=0	0.0	0.823	0.0	0.823	0.666
x=6, y=3, zoom=0	x=6, y=4, zoom=0	0.0	0.947	0.0	0.947	0.5
x=5, y=2, zoom=0	x=6, y=3, zoom=0	0.971	0.971	0.0	0.971	0.333

Easting = 1 , Southing = 1 , Zooming = 0

TILES TO PREFETCH	
TYPE	TILE LOCATION
NEIGHBOR	x=24, y=30, zoom=11
NEIGHBOR	x=24, y=29, zoom=11
NEIGHBOR	x=23, y=30, zoom=11
NEIGHBOR	x=24, y=28, zoom=11
NEIGHBOR	x=22, y=30, zoom=11
NEIGHBOR	x=24, y=27, zoom=11
NEIGHBOR	x=21, y=30, zoom=11

NAVIGATION – 11

Direction -> EAST

Top-Left Tile -> x=22, y=27, zoom=11

HISTORY		EAST	SOUTH	ZOOM	HIT RATIO	ZOOM FACTOR
FROM	TO					
x=8, y=6, zoom=0	x=9, y=6, zoom=0	1.0	0.0	0.0	1.0	1.0
x=7, y=6, zoom=0	x=8, y=6, zoom=0	0.538	0.0	0.0	0.538	0.833
x=6, y=5, zoom=0	x=7, y=6, zoom=0	0.538	0.538	0.0	0.538	0.666
x=6, y=4, zoom=0	x=6, y=5, zoom=0	0.0	0.837	0.0	0.837	0.5
x=6, y=3, zoom=0	x=6, y=4, zoom=0	0.0	0.949	0.0	0.949	0.333

Easting = 1 , Southing = 1 , Zooming = 0

TILES TO PREFETCH	
TYPE	TILE LOCATION
NEIGHBOR	x=22, y=27, zoom=11
NEIGHBOR	x=26, y=30, zoom=11
NEIGHBOR	x=26, y=29, zoom=11
NEIGHBOR	x=25, y=30, zoom=11
NEIGHBOR	x=24, y=30, zoom=11
NEIGHBOR	x=26, y=28, zoom=11
NEIGHBOR	x=23, y=30, zoom=11
NEIGHBOR	x=26, y=27, zoom=11
NEIGHBOR	x=24, y=28, zoom=11

NAVIGATION – 12

Direction -> ZOOM IN

Top-Left Tile -> x=46, y=55, zoom=10

HISTORY		EAST	SOUTH	ZOOM	HIT RATIO	ZOOM FACTOR
FROM	TO					
x=9, y=6, zoom=0	x=9, y=6, zoom=-1	0.0	0.0	-1.0	0.5	1.0
x=8, y=6, zoom=0	x=9, y=6, zoom=0	0.75	0.0	0.0	0.75	0.833
x=7, y=6, zoom=0	x=8, y=6, zoom=0	0.531	0.0	0.0	0.531	0.666
x=6, y=5, zoom=0	x=7, y=6, zoom=0	0.531	0.531	0.0	0.531	0.5
x=6, y=4, zoom=0	x=6, y=5, zoom=0	0.0	0.812	0.0	0.812	0.333

Easting = 1 , Southing = 1 , Zooming = -1

TILES TO PREFETCH	
TYPE	TILE LOCATION
NEIGHBOR	x=49, y=58, zoom=10
NEIGHBOR	x=49, y=57, zoom=10
NEIGHBOR	x=48, y=58, zoom=10
NEIGHBOR	x=49, y=56, zoom=10
NEIGHBOR	x=47, y=58, zoom=10
NEIGHBOR	x=49, y=55, zoom=10
NEIGHBOR	x=46, y=58, zoom=10
ZOOM TILE (& 8 TILES AROUND)	x=94, y=112, zoom=9

NAVIGATION – 13

Direction -> ZOOM IN

Top-Left Tile -> x=94, y=111, zoom=9

HISTORY		EAST	SOUTH	ZOOM	HIT RATIO	ZOOM FACTOR
FROM	TO					
x=9, y=6, zoom=-1	x=9, y=6, zoom=-2	0.0	0.0	-1.0	0.0	1.0
x=9, y=6, zoom=0	x=9, y=6, zoom=-1	0.0	0.0	-0.833	0.286	0.833
x=8, y=6, zoom=0	x=9, y=6, zoom=0	0.545	0.0	0.0	0.545	0.666
x=7, y=6, zoom=0	x=8, y=6, zoom=0	0.466	0.0	0.0	0.466	0.5
x=6, y=5, zoom=0	x=7, y=6, zoom=0	0.466	0.466	0.0	0.466	0.333

Easting = 0 , Southing = 0 , Zooming = -1

TILES TO PREFETCH	
TYPE	TILE LOCATION
ZOOM TILE (& 24 TILES AROUND)	x=190, y=226, zoom=8

NAVIGATION – 14

Direction -> ZOOM IN

Top-Left Tile -> x=189, y=224, zoom=8

HISTORY		EAST	SOUTH	ZOOM	HIT RATIO	ZOOM FACTOR
FROM	TO					
x=9, y=6, zoom=-2	x=9, y=6, zoom=-3	0.0	0.0	-1.0	0.5	1.0
x=9, y=6, zoom=-1	x=9, y=6, zoom=-2	0.0	0.0	-0.833	0.2	0.833
x=9, y=6, zoom=0	x=9, y=6, zoom=-1	0.0	0.0	-0.666	0.333	0.666
x=8, y=6, zoom=0	x=9, y=6, zoom=0	0.538	0.0	0.0	0.538	0.5
x=7, y=6, zoom=0	x=8, y=6, zoom=0	0.468	0.0	0.0	0.468	0.333

Easting = 1 , Southing = 0 , Zooming = -1

TILES TO PREFETCH	
TYPE	TILE LOCATION
ZOOM TILE (& 24 TILES AROUND)	x=380, y=452, zoom=7
NEIGHBOR	x=192, y=225, zoom=8
NEIGHBOR	x=192, y=226, zoom=8
NEIGHBOR	x=192, y=224, zoom=8
NEIGHBOR	x=192, y=227, zoom=8

NAVIGATION – 15

Direction -> ZOOM IN

Top-Left Tile -> x=379, y=450, zoom=7

HISTORY		EAST	SOUTH	ZOOM	HIT RATIO	ZOOM FACTOR
FROM	TO					
x=9, y=6, zoom=-3	x=9, y=6, zoom=-4	0.0	0.0	-1.0	0.0	1.0
x=9, y=6, zoom=-2	x=9, y=6, zoom=-3	0.0	0.0	-0.833	0.666	0.833
x=9, y=6, zoom=-1	x=9, y=6, zoom=-2	0.0	0.0	-0.666	0.333	0.666
x=9, y=6, zoom=0	x=9, y=6, zoom=-1	0.0	0.0	-0.5	0.4	0.5
x=8, y=6, zoom=0	x=9, y=6, zoom=0	0.571	0.0	0.0	0.571	0.333

Easting = 1 , Southing = 0 , Zooming = -1

TILES TO PREFETCH	
TYPE	TILE LOCATION
ZOOM TILE (& 24 TILES AROUND)	x=762, y=902, zoom=6
NEIGHBOR	x=383, y=450, zoom=7
NEIGHBOR	x=383, y=451, zoom=7
NEIGHBOR	x=383, y=452, zoom=7

NAVIGATION – 16

Direction -> ZOOM IN

Top-Left Tile -> x=761, y=902, zoom=6

HISTORY		EAST	SOUTH	ZOOM	HIT RATIO	ZOOM FACTOR
FROM	TO					
x=9, y=6, zoom=-4	x=9, y=6, zoom=-5	0.0	0.0	-1.0	1.0	1.0
x=9, y=6, zoom=-3	x=9, y=6, zoom=-4	0.0	0.0	-0.833	1.0	0.833
x=9, y=6, zoom=-2	x=9, y=6, zoom=-3	0.0	0.0	-0.666	0.714	0.666
x=9, y=6, zoom=-1	x=9, y=6, zoom=-2	0.0	0.0	-0.5	0.385	0.5
x=9, y=6, zoom=0	x=9, y=6, zoom=-1	0.0	0.0	-0.333	0.429	0.333

Easting = 0 , Southing = 0 , Zooming = -1

TILES TO PREFETCH	
TYPE	TILE LOCATION
ZOOM TILE (& 8 TILES AROUND)	x=1524, y=1806, zoom=5

NAVIGATION – 17

Direction -> EAST

Top-Left Tile -> x=762, y=902, zoom=6

HISTORY		EAST	SOUTH	ZOOM	HIT RATIO	ZOOM FACTOR
FROM	TO					
x=9, y=6, zoom=-5	x=10, y=6, zoom=-5	0.555	0.0	0.0	0.555	1.0
x=9, y=6, zoom=-4	x=9, y=6, zoom=-5	0.0	0.0	-0.833	0.878	0.833
x=9, y=6, zoom=-3	x=9, y=6, zoom=-4	0.0	0.0	-0.666	0.951	0.666
x=9, y=6, zoom=-2	x=9, y=6, zoom=-3	0.0	0.0	-0.5	0.706	0.5
x=9, y=6, zoom=-1	x=9, y=6, zoom=-2	0.0	0.0	-0.333	0.389	0.333

Easting = 1 , Southing = 0 , Zooming = -1

TILES TO PREFETCH	
TYPE	TILE LOCATION
ZOOM TILE (& 8 TILES AROUND)	x=1526, y=1806, zoom=5
NEIGHBOR	x=765, y=903, zoom=6
NEIGHBOR	x=765, y=902, zoom=6
NEIGHBOR	x=765, y=904, zoom=6

NAVIGATION – 18

Direction -> EAST

Top-Left Tile -> x=763, y=902, zoom=6

HISTORY		EAST	SOUTH	ZOOM	HIT RATIO	ZOOM FACTOR
FROM	TO					
x=10, y=6, zoom=-5	x=11, y=6, zoom=-5	1.0	0.0	0.0	1.0	1.0
x=9, y=6, zoom=-5	x=10, y=6, zoom=-5	0.619	0.0	0.0	0.619	0.833
x=9, y=6, zoom=-4	x=9, y=6, zoom=-5	0.0	0.0	-0.666	0.884	0.666
x=9, y=6, zoom=-3	x=9, y=6, zoom=-4	0.0	0.0	-0.5	0.951	0.5
x=9, y=6, zoom=-2	x=9, y=6, zoom=-3	0.0	0.0	-0.333	0.709	0.333

Easting = 1 , Southing = 0 , Zooming = -1

TILES TO PREFETCH	
TYPE	TILE LOCATION
ZOOM TILE (& 8 TILES AROUND)	x=1528, y=1806, zoom=5
NEIGHBOR	x=766, y=903, zoom=6
NEIGHBOR	x=766, y=902, zoom=6
NEIGHBOR	x=766, y=904, zoom=6

NAVIGATION – 19

Direction -> EAST

Top-Left Tile -> x=764, y=902, zoom=6

HISTORY		EAST	SOUTH	ZOOM	HIT RATIO	ZOOM FACTOR
FROM	TO					
x=11, y=6, zoom=-5	x=12, y=6, zoom=-5	1.0	0.0	0.0	1.0	1.0
x=10, y=6, zoom=-5	x=11, y=6, zoom=-5	1.0	0.0	0.0	1.0	0.833
x=9, y=6, zoom=-5	x=10, y=6, zoom=-5	0.644	0.0	0.0	0.644	0.666
x=9, y=6, zoom=-4	x=9, y=6, zoom=-5	0.0	0.0	-0.5	0.886	0.5
x=9, y=6, zoom=-3	x=9, y=6, zoom=-4	0.0	0.0	-0.333	0.952	0.333

Easting = 1 , Southing = 0 , Zooming = -1

TILES TO PREFETCH	
TYPE	TILE LOCATION
ZOOM TILE (& 8 TILES AROUND)	x=1530, y=1806, zoom=5
NEIGHBOR	x=767, y=903, zoom=6
NEIGHBOR	x=767, y=902, zoom=6
NEIGHBOR	x=767, y=904, zoom=6

NAVIGATION – 20

Direction -> ZOOM IN

Top-Left Tile -> x=1529, y=1805, zoom=5

HISTORY		EAST	SOUTH	ZOOM	HIT RATIO	ZOOM FACTOR
FROM	TO					
x=12, y=6, zoom=-5	x=12, y=6, zoom=-6	0.0	0.0	-1.0	1.0	1.0
x=11, y=6, zoom=-5	x=12, y=6, zoom=-5	1.0	0.0	0.0	1.0	0.833
x=10, y=6, zoom=-5	x=11, y=6, zoom=-5	1.0	0.0	0.0	1.0	0.666
x=9, y=6, zoom=-5	x=10, y=6, zoom=-5	0.656	0.0	0.0	0.656	0.5
x=9, y=6, zoom=-4	x=9, y=6, zoom=-5	0.0	0.0	-0.333	0.888	0.333

Easting = 1 , Southing = 0 , Zooming = -1

TILES TO PREFETCH	
TYPE	TILE LOCATION
ZOOM TILE (& 24 TILES AROUND)	x=3062, y=3612, zoom=4
NEIGHBOR	x=1533, y=1805, zoom=5
NEIGHBOR	x=1533, y=1806, zoom=5
NEIGHBOR	x=1533, y=1807, zoom=5

NAVIGATION – 21

Direction -> ZOOM IN

Top-Left Tile -> x=1529, y=1805, zoom=5

HISTORY		EAST	SOUTH	ZOOM	HIT RATIO	ZOOM FACTOR
FROM	TO					
x=12, y=6, zoom=-6	x=12, y=6, zoom=-7	0.0	0.0	-1.0	0.75	1.0
x=12, y=6, zoom=-5	x=12, y=6, zoom=-6	0.0	0.0	-0.833	0.833	0.833
x=11, y=6, zoom=-5	x=12, y=6, zoom=-5	0.9	0.0	0.0	0.9	0.666
x=10, y=6, zoom=-5	x=11, y=6, zoom=-5	0.944	0.0	0.0	0.944	0.5
x=9, y=6, zoom=-5	x=10, y=6, zoom=-5	0.662	0.0	-0.333	0.662	0.333

Easting = 1 , Southing = 0 , Zooming = -1

TILES TO PREFETCH	
TYPE	TILE LOCATION
ZOOM TILE (& 24 TILES AROUND)	x=6124, y=7226, zoom=3
NEIGHBOR	x=3064, y=3612, zoom=4
NEIGHBOR	x=3064, y=3613, zoom=4
NEIGHBOR	x=3064, y=3614, zoom=4

APPENDIX - C

Statistical Evaluation of Test Simulations

In this section, the statistical data obtained after test simulations executed for each 5 methods using Geoserver are given. A simulation is executed 100 times and the navigation set used for each simulation are generated to be unique.

The statistical data obtained for refresh times of each method is given in the table below. For each method, “arithmetic average, arithmetic mean, standard deviation, variance, maximum and minimum” values are obtained after 100 simulations.

Table 22: Statistical data of refresh times obtained for each method

	TC	SP	RAP	HCBP	PKM
AVG	25218.348	35770.577	15897.494	22580.091	20137.396
MEAN	25042.808	35506.903	15790.082	22387.008	19959.855
STD DEV	3017.040	4316.686	1862.905	3000.542	2681.424
VAR	9102532.336	18633786.365	3470416.799	9003255.176	7190034.825
MAX	30939.678	44277.744	20404.340	28569.686	25470.076
MIN	20675.868	25765.962	12262.615	17949.206	14733.480

Table 23: Statistical data of refresh times obtained for each method

	TC	SP	RAP	HCBP	PKM
AVG	59.507915	172.387895	84.741267	63.228430	89.2094023
MEAN	59.436274	172.341856	84.692585	63.108015	89.0863594
STD DEV	2.982983	4.038517	2.916693	3.980492	4.7899252
VAR	8.898188	16.309624	8.507098	15.844321	22.9433835
MAX	66.103461	184.582627	92.935675	71.823728	101.3645039
MIN	54.531893	164.200887	79.419992	57.300433	82.9995689

The figures below give the normal distribution of the refresh times and memory usage.

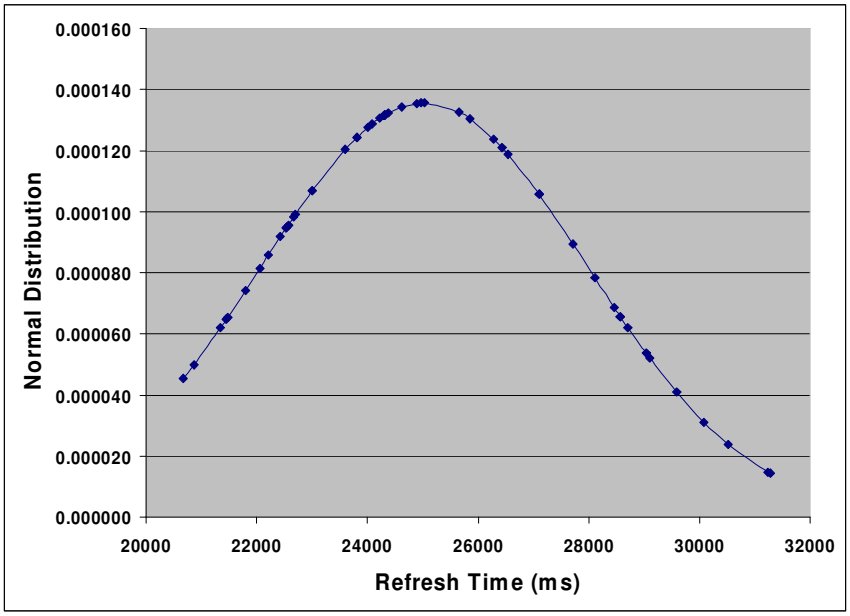

Figure 54: Normal distribution of refresh times for TC method

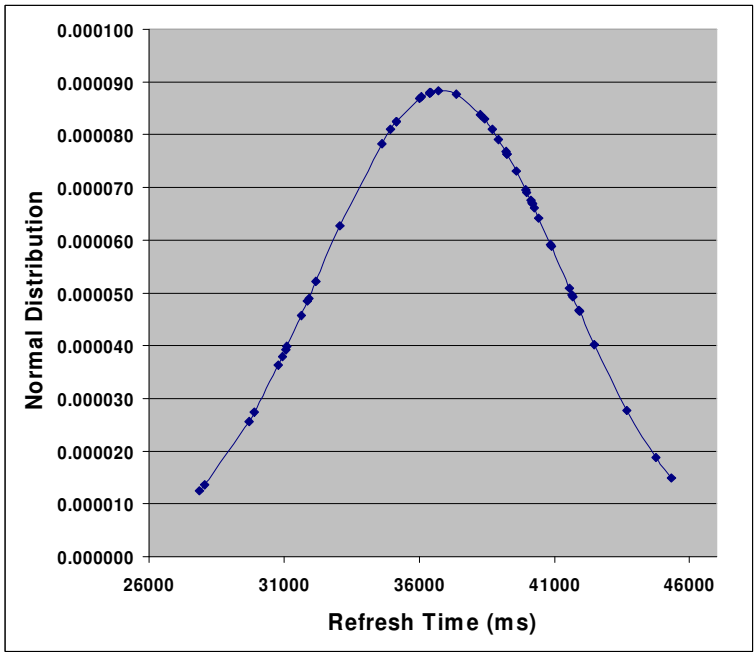

Figure 55: Normal distribution of refresh times for SP method

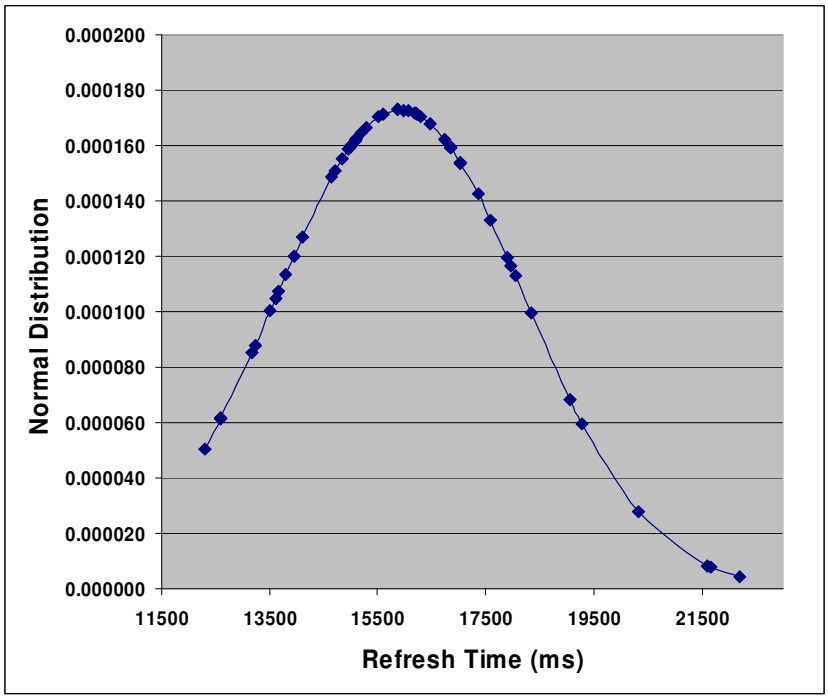

Figure 56: Normal distribution of refresh times for RAP method

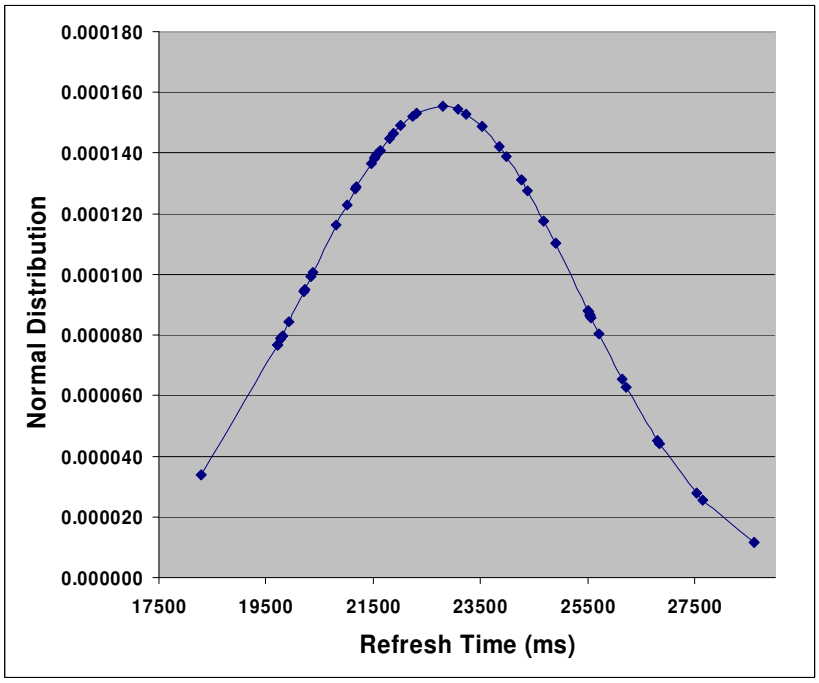

Figure 57: Normal distribution of refresh times for HCBP method

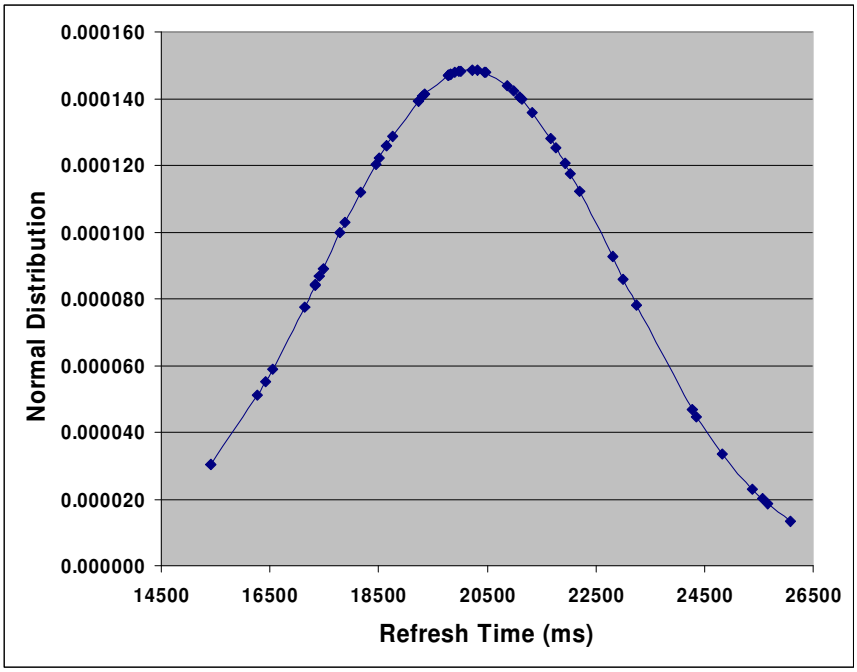

Figure 58: Normal distribution of refresh times for PKM method

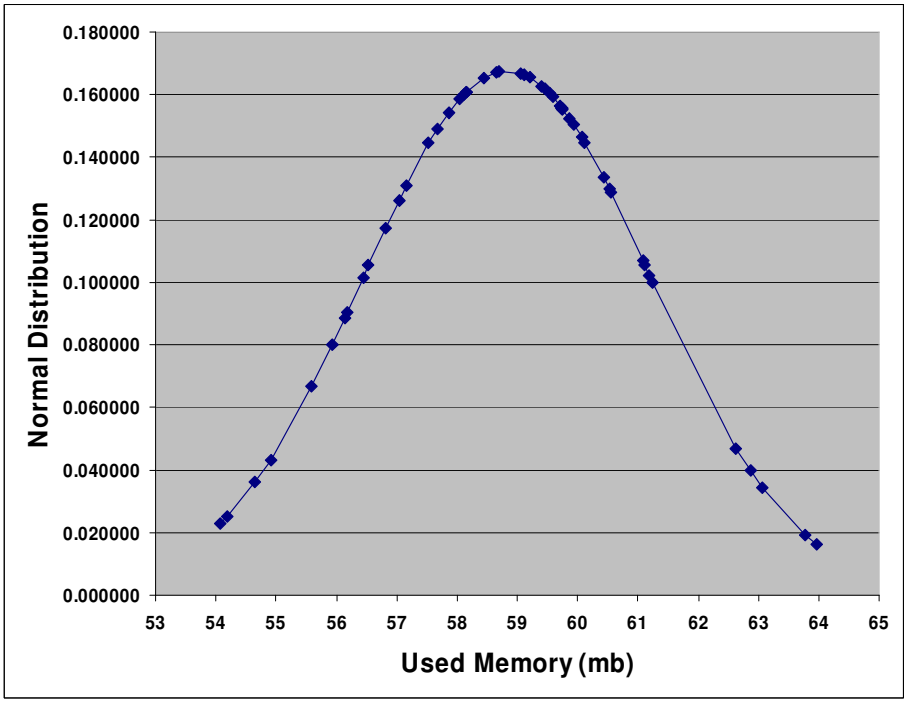

Figure 59: Normal distribution of memory usages for TC method

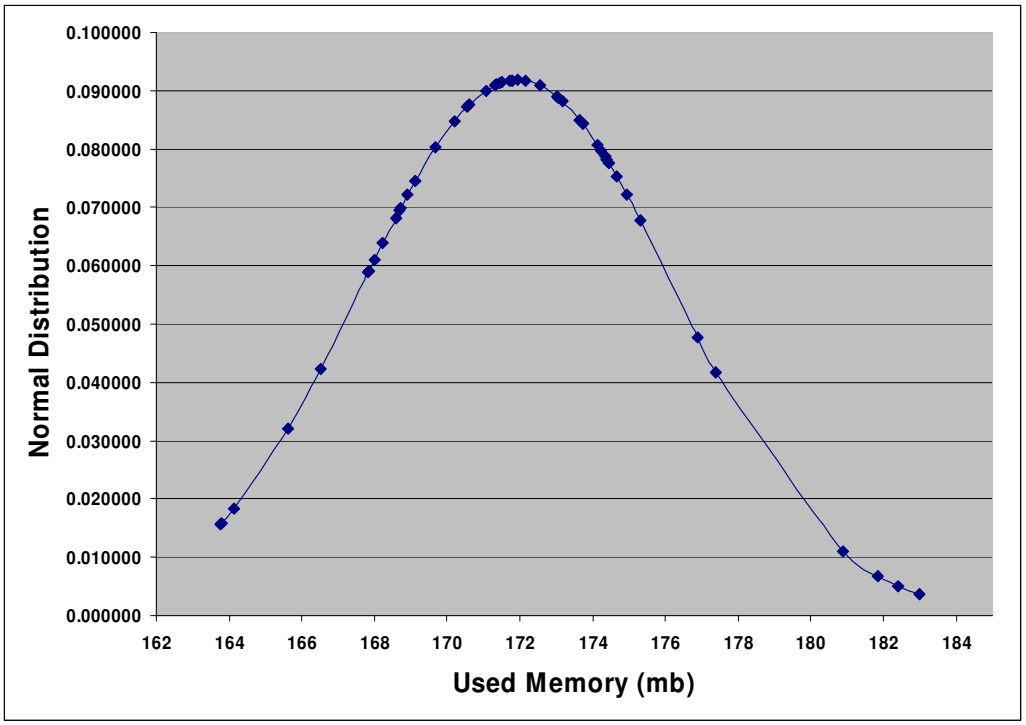

Figure 60: Normal distribution of memory usages for SP method

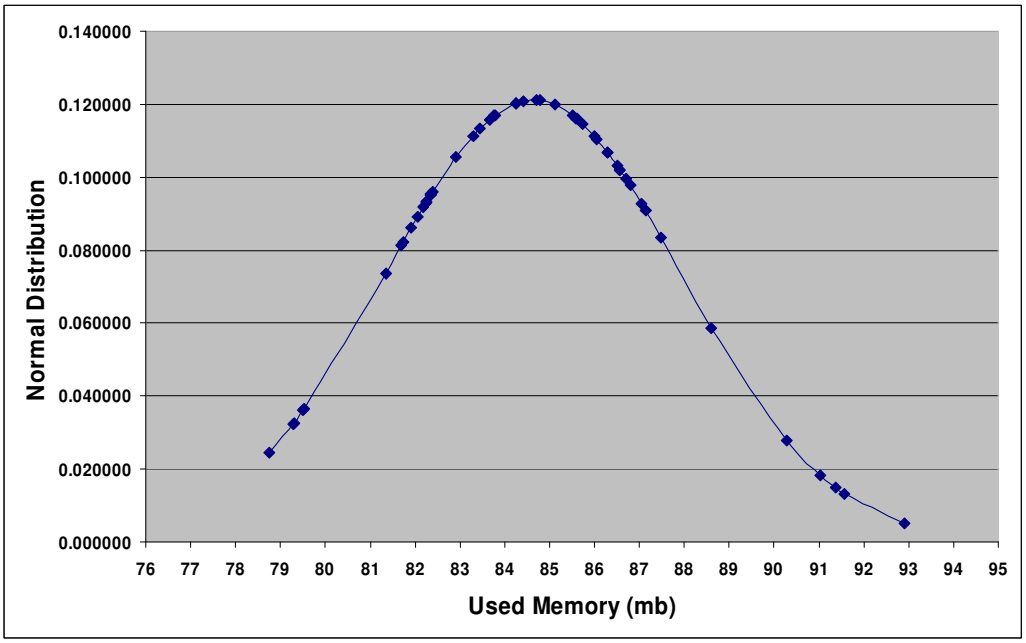

Figure 61: Normal distribution of memory usages for RAP method

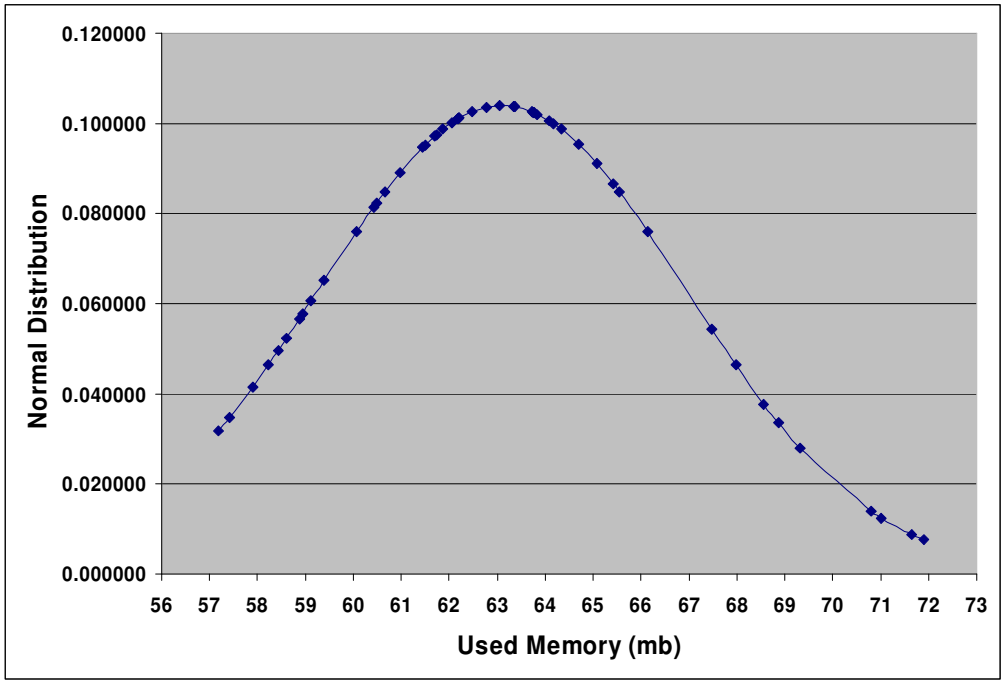

Figure 62: Normal distribution of memory usages for HCBP method

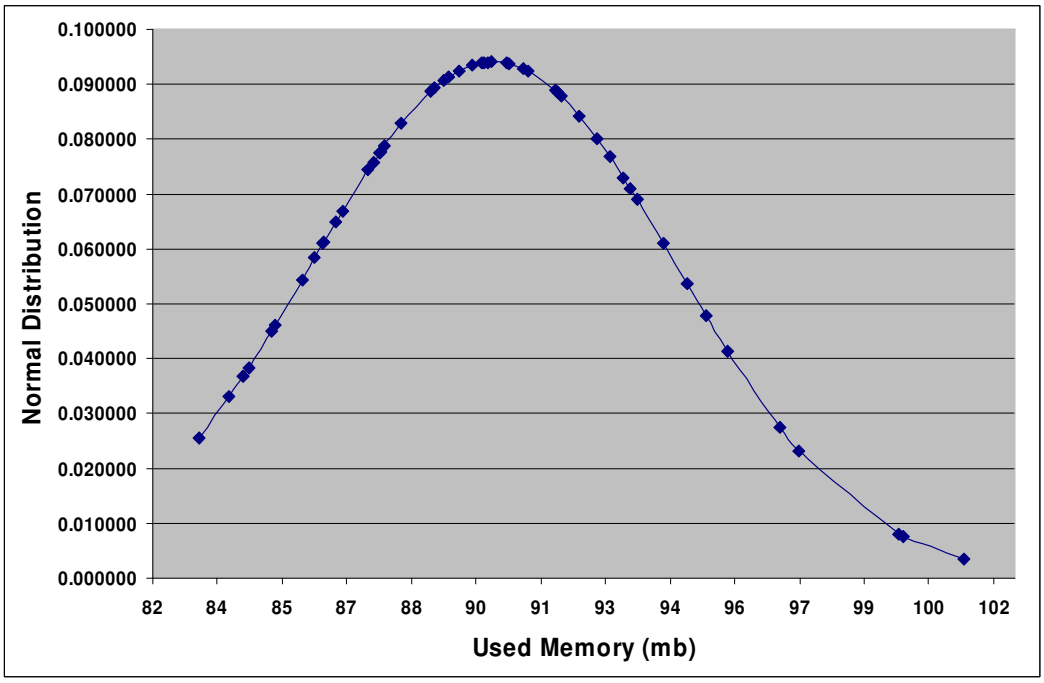

Figure 63: Normal distribution of memory usages for PKM method