Table of Contents

1. Performance Issues, and measurements and analysis

This chapter presents the common performance issues in service-oriented, federated and interoperable GIS systems built based on the common structured data model. As the common data model, OGC [ogc] defined Geographic Markup Language (GML) [GML] is used. Developing a federated information system inspired us enhancing the whole system performance by applying novel parallel processing and caching techniques applied together in large scale interoperable information systems (see Chapter 1.3.2). In addition to this, we proposed some other innovative performance enhancement techniques (see Chapter 1.3.1) such as streaming data transfers, and enhanced parsing and rendering of semi-structured geo-data sets (GML). At the end of each chapter explaining these techniques, performance tests and analysis are provided.

The organization of the rest of the chapter is as follows. Chapter 1.1 summarizes and reviews the general performance issues of interoperable service-oriented GIS systems in which interoperability is granted by using XML-structured common data model and Web Services. Chapter 1.2 presents the limits of the ordinary GIS systems without having any performance enhancements which will be our comparison base for our proposed techniques. Throughout the document, with the term "ordinary" we mean on-demand, single-threaded and no-caching systems. The last chapter (Chapter 1.3) explains our approaches to developing high performance GIS systems, and performance evaluations by comparing with the ordinary systems. We approach the performance issues from the two aspects. One is data-oriented and the other is federator-oriented. The data-oriented approaches deal with transferring large-sized XML structured data in common model, and high performance parsing and rendering algorithms. The federator-oriented approaches deal with the performance enhancement techniques based on data characteristics. For the un-frequently changing archived data handling we propose pre-fetching technique. On the other hand, for the frequently changing archived data, we propose a hybrid technique composed of caching and parallel processing applied together. This hybrid system is actually proposes a novel locality based workload forecasting for variable sized and un-evenly distributed data, and locality information is obtained from the session based one time caching (Chapter 1.3.2.2).

1.1. General Performance Issues in Interoperable Service-oriented GIS

Performance issues in interoperable service-oriented GIS can be generalized into two groups:

- Issues regarding semi-structured data model (GML).
- Issues regarding domain specific data characteristics. In GIS, the data is described with location attribute defined in (x, y) coordinates. Based on the location value, the data is characterized as un-evenly distributed and variable sized.

1.1.1. Using Semi-structured Data Model

Using semi-structured data model enables interoperability and inter-service communication. XML's emergence as the de facto standard for encoding tree-oriented, semi-structured data has brought significant interoperability and standardization benefits to distributed computing. On the other hand, performance has been still a persistent concern for large scale applications, because of the size issues and processing overheads [Lu2006]. The processing is detailed as parsing and differentiating (separating) the core-data from the attributes and other tags to create required application specific data formats.

Structured data representations enable adding some attributes and additional information to the data. These attributes and additions are mostly due to the interoperability and security reasons.

XML representations of data tend to be significantly larger than binary representations of the same data. The larger document size means that the greater bandwidth is required to transfer of data, as compared to the equivalent binary representations. The larger size often implies greater processing costs as well, since much of the overhead involved in communication processing is going to be based on the data volume.

There are two well-known and commonly-used paradigms for processing XML data, the Document Object Model (DOM) and the Simple API for XML (SAX). DOM builds a complete object representation of the XML document in memory. This can be memory intensive for large documents, and entails making at least two passes through the data. SAX operates at one level lower. Rather than actually constructing a model in memory, it informs the application of elements through callbacks. This also requires at least two passes through the data. These are all expensive and resource (such as CPU and memory) consuming processes and they don't provide enough performance for the large scale applications.

In the document these issues are called data-oriented performance issues, and the proposed solution approaches are presented in Chapter 1.3.1

1.1.2. Data Characteristics

The different domains have different data types having different characteristic to be handled. As an example, in GIS domain, which is our motivating domain, science applications need to manipulate geo-data. Geo-data is described with its location ((x, y) coordinates) on the earth. Based on the location attribute, geo-data is un-evenly distributed (such as human-population and temperature distributions) and variable sized. Because of these characteristics, it is not easy to implement some well-known performance enhancing techniques as applied in other science domains. Since it is not possible to know the work-load earlier, the classic load balancing algorithms do not work for the variable sized and unevenly distributed data. The work is decomposed into independent work pieces, and the work pieces are of highly variable sized. This issue is illustrated in Figure 1 for the case of using one-step-binary query partitioning based on the location attribute of the data. As it is illustrated in the figure, there are four worker nodes, and the worker node assigned to R2 gets the heaviest part of the total work, and therefore the expected performance gain from usinf classic load balancing will not be obtained.

The geo-data is queried based on their attributes. Since all the data is described by their locations, in order to get the data sets falling in a specific region, the bounding box (bbox) values are used. The regions are defined in bboxes. A bbox defines a rectangular shape in a two-sim coordinate plane, and it is dormulated as (minx, miny, maxx, maxy). For example, Figure 1 shows a region formulated in bbox value (a, b, c, d).

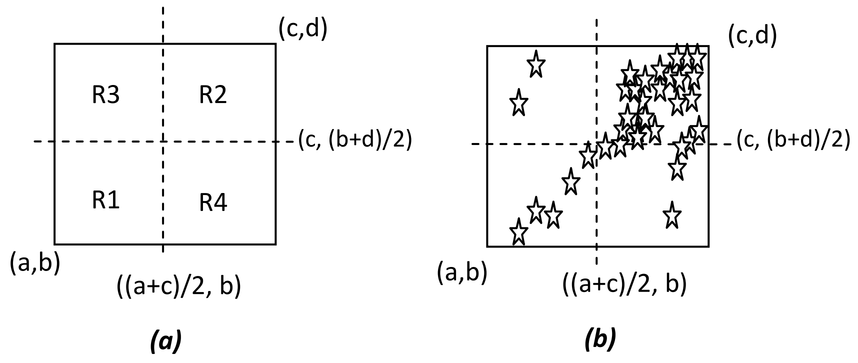


Figure 1: Un-balanced load sharing. Server assigned R2:"((a+b)/2, (b+d)/2), (c, d)" gets the most of the work.

These performance issues are dealt with in Chapter 1.3.2 which is called federator-oriented performance issues.

Before giving our solution approaches to these issues generalized in last two chapters, we give the ordinary OGC compatible GIS systems' performance results and present general processes involved in it. Interoperability is granted by using structured common data model for the representation of any data.

The performance results presented in the following chapter will be our comparison base for the proposed techniques' measurement of successes.

1.2. Ordinary GIS System Performance Using Common Data Model for the Interoperability (To compare with)

In order to solve data and service heterogeneities for the GIS computation and data services OGC and ISO/TC-211 standards are used. These standards recommend using structured common data model called GML for the representation of location based geo-data. The standard bodies aim is to make the geographic information and services neutral and available across any network, application, or platform. Currently the two major geospatial standards organizations are the Open Geospatial Consortium (OGC) and the Technical Committee tasked by the International Standards Organization (ISO/TC211).

With the ordinary system we mean a GIS developed with widely used technique without using any novel advanced techniques to handle the data. Most of the implementations are based on single-threaded and on-demand processing. Deegree project [deegree] and Minnesota Map Server [minmapserv] can be given as sample projects. In order to compare and contrast our novel approaches to the ordinary systems approaches, we tested and presented their performance results at Table 1 and *Figure 2*.

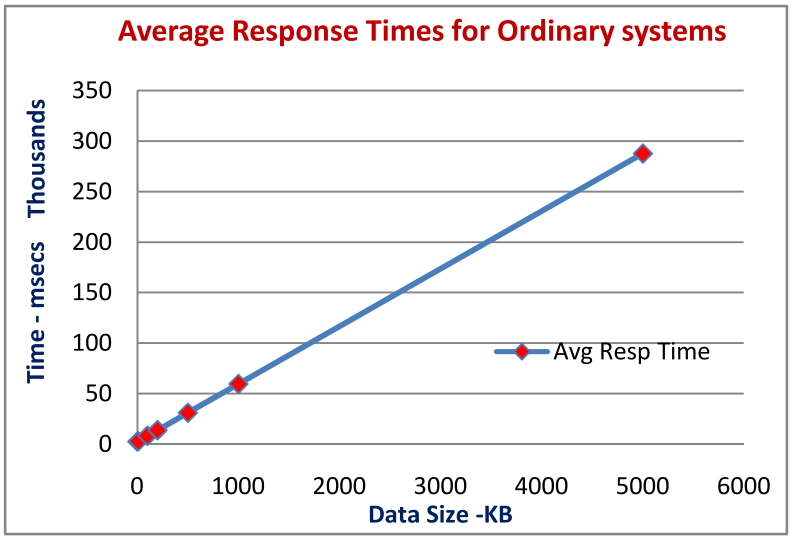
This performance results teach us valuable lessons in terms of the capabilities and limits of the general distributed and interoperable GIS systems. From the figure we draw following conclusions. First, for the small data payloads (less than 500KB) the response time is acceptable.

However for larger data payloads the performance gets worse and the response time gets relatively longer. On the other hand, scientific applications require handling (transferring, parsing, rendering and displaying) large scale data.

Table 1: The round-trip times (or response times) of the ordinary system.

Data Size KB	ART- Average Response Times (msec)	Log(ART) msec	Standard Deviation
1	2,375.24	3.38	152.40
10	2,578.69	3.41	252.49
100	7,973.16	3.90	374.12
200	13,612.78	4.13	417.19
500	30,868.52	4.49	482.83
1000	59,635.69	4.78	343.76
5000	288,594.12	5.46	333.07

(a)



(b)

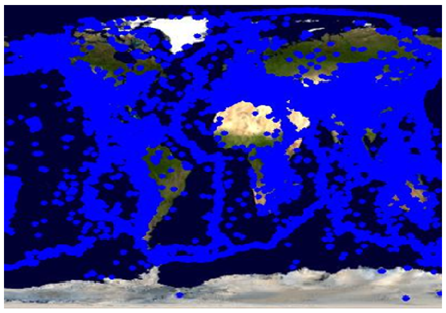


Figure 2: (a) Performance result of the ordinary system. (b) Sample output consisting of two layers.

In order to be able to make more reasonable comparisons, we adjusted the timing values given in Table 1 by taking their logarithmic values and plotted them in Figure 3.

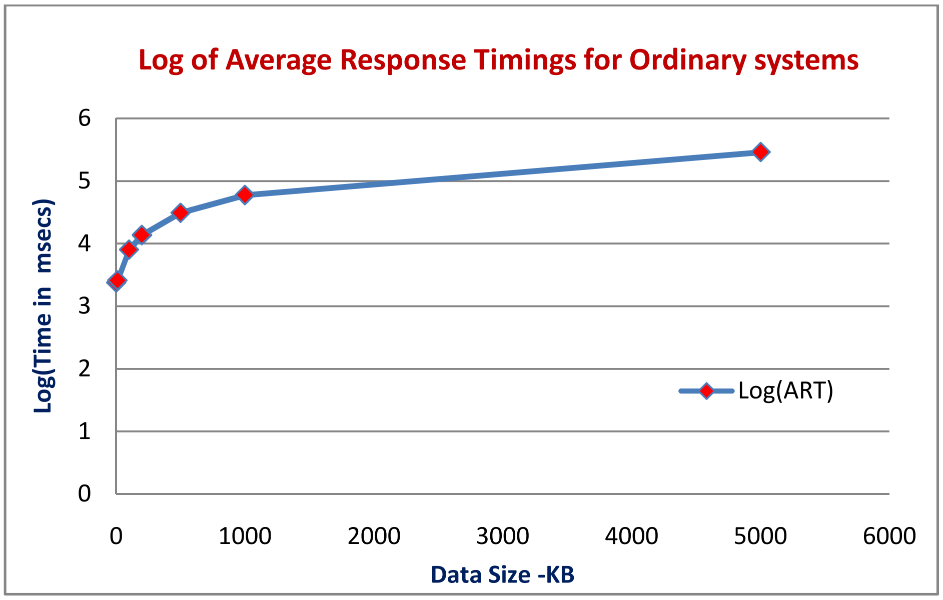


Figure 3: Adjusted performance values over *Figure 2* for the ordinary systems.

The test above shows that the performance is not enough in order to meet Geo-Science Grids' performance requirements. As you see, if the spatial data is over 500KB, the ordinary system framework is not feasible to use in large scale science applications. Time column (y) in the *Figure 2* (a) and Figure 3 represent response times including querying, transforming, rendering and displaying spatial data. In other words the figure illustrates the response time of the ordinary GIS systems as formulated below:

$$time_{(measured)} = time_{(result\ is\ displayed)} - time_{(client\ makes\ request)}.$$

Measured time in the figure ($\boldsymbol{time_{(measured)}}$) can be detailed as below:

- **[*time*$_{(client\ makes\ request)}$.]** Client makes requests by interactive smart map tools to Web Map Server (WMS).
- WMS parse and render requests and define set of actions required based on the requests and its capabilities file.
- WMS Creates map images (from the returned datasets) and returns them to the clients:
    - Defines the set of WFSs [WFS] and other WMSs [WMS] to communicate with to build the response by in accordance with its capability file.
    - Creates requests for WFSs and other WSMs
    - Invokes WFSs *getFeature* Web Services for vector data encoded in Geographic Markup language (GML) [GML].
    - Invokes other WMSs *getMap* Web Services for raster data rendered in map images
    - Transferring GML data (feature collections) from WFS and WMS
    - Parsing and rendering returned GML data sets
    - Aggregating and overlaying layers according to the request and capability file.
    - Sending the map images to the WMS Client.
- **[*time*$_{(map\ is\ displayed)}$]** Client shows the returned maps on his browser

From our experience we saw that depending on the total data size, over %90 of the *time*$_{(measured)}$ comes from the step called "transferring GML data (feature collections) from WFS and WMS". Because of that, even if we use the most efficient and fast parsing and rendering algorithms (such as using pull parsing or application specific XPath querying), it won't improve performance very much if the data transfer time still stays that much high as shown in the *Figure 2*.

1.3. High Performance Design and Evaluation of the Proposed System

Our approaches to the performance issues are grouped into two. The first group of approaches deals with the general performance issues result from using semi-structured data encodings (such as GML), and large size data exchange, parsing and rendering (Chapter 1.3.1). The second group of approaches is regarding the federator oriented design and techniques to enhance the overall system performance (Chapter 1.3.2).

1.3.1. Data-oriented Approaches

Distributed GIS systems typically handle a large volume of datasets. Therefore the transmission, processing and visualization/rendering techniques used need to be responsive to provide quick, interactive feedback. There are some characteristics of GIS services and data that make it difficult to design distributed GIS with satisfactory performance. One of them is that GIS services often transmit large resulting datasets such as structured data, images, or large files in tabular-matrix formats.

In order to provide interoperability and extensibility we use common data format represented and formulated in XML. This degrades the performance even worse for large scale applications. The major hurdle of the proposed federated GIS framework is encoding, transferring and rendering the data in common data model. In the following two sub-sections we present our approaches to these issues. One is regarding large scale structured data transfer (Chapter 1.3.1.1) and other is regarding the large scale data parsing (Chapter 1.3.1.2).

1.3.1.1. Streaming Data Transfer

Our experience shows that although we can easily integrate several GIS services into complex tasks by using Web Services, providing high-rate transportation capabilities for large amounts of data remains a problem because the pure Web Services implementations rely on SOAP [Donbox] messages exchanged over HTTP. This conclusion has led us to an investigation of topic-based publish-subscribe messaging systems for exchanging SOAP messages and data payload between Web Services. We have used NaradaBrokering [Pallickara2003] which provides several useful features besides streaming data transport such as reliable delivery, ability to choose alternate transport protocols, security and recovery from network failures.

Naradabrokering is a message oriented middleware (MoM) [Tran] system which facilitates communications between entities through the exchange of messages. This also allows us to receive individual results and publish them to the messaging substrate instead of waiting for whole result set to be returned.

In case of transferring the GML result set in the form of string causes some problems when the GML is larger than some amount of size (500KB see *Figure 2*-a). Since the WFS returns the resulting XML document as an <xsd:string>, this has to be constructed in memory and the size will depend on several parameters such as the system configuration and memory allocated to the Java Virtual Machine etc. Consequently there will be a limit on the size of the returned XML documents. For these reasons we have investigated alternative ways for data transport and, researched the use of topic based publish-subscribe messaging systems for streaming the data. Our research on NaradaBrokering shows that it can be used to stream large amount of data between nodes without significant overhead. Additional capabilities such as reliable messaging and support for different transport protocols already inherent in NaradaBrokering show that it is a powerful yet easy to integrate messaging infrastructure. For these reasons we have developed a novel Web Map Service and Web Feature Service that integrate OGC specifications with Web Service-SOAP [Donbox] calls and NaradaBrokering messaging system. Architecture is shown in Figure 4.

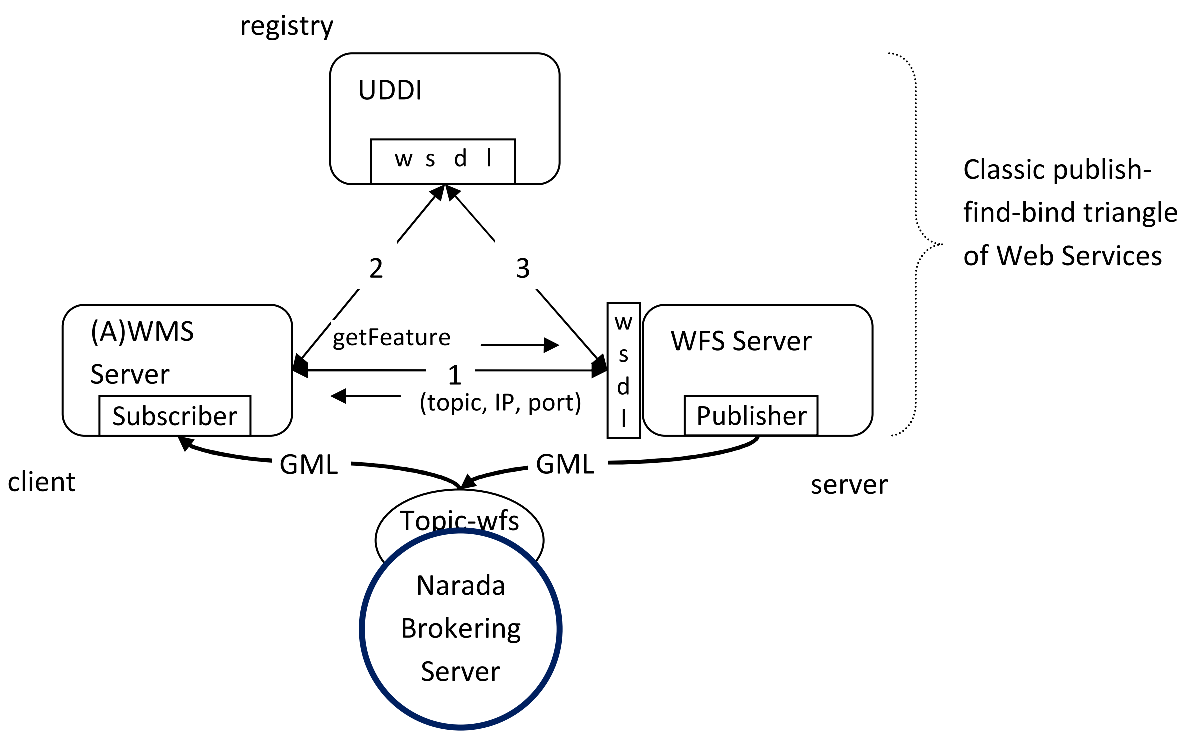


Figure 4: Streaming data transfer using Naradabrokering publish-subscribe topic based messaging middleware.

Connection lines 2 and 3, and UDDI (Universal Description, Discovery and Integration) [uddi] service are displayed in the figure for showing classic publish-find-bind triangle of the Web Service based Service Oriented architecture. We don't go into details of these interactions and UDDI registry service in this document but these can be summarized as following. WFS services publish their existence and service providing wit their WSDL service description files (line-3). Clients (such as WMS) find appropriate WFS by searching UDDI registries (line-2). After finding appropriate service, clients are bind to that service by creating their client stubs. Instead of using lines 2 and 3, clients can also directly communicate with the services if they know the service's WSDL file earlier.

In case of streaming through Naradabrokering, the clients make the requests with standard SOAP messages (line-1) but for retrieving the results a NaradaBrokering subscriber class is used.

Through first request to Web Service (called *getFeature)*, WMS gets the topic (publish-subscribe for a specific data), IP and port to which WFS streams requested data. Second request is done by *NaradaBrokering* Subscriber. Even in the case of that the whole data is not received by WMS; WMS can draw the map image with the returned data. This depends on the WMS's internal implementation.

Table 2 gives a comparison of the streaming and non-streaming data access approaches for the different data sizes. These values are obtained by running Pattern Informatics (PI) [patterninfo] geo-science application over the earthquake seismic data records. These are GML data access times including query conversion at WFS, result set conversion from database to GML and transfer times from WFS to AWMS.

Table 2: Data access times (from source to AWMS) while using (1) streaming and (2)non-streaming data transfer techniques.

	Streaming			Non-Streaming	
Data Size (KB)	Average Time for Streaming	Average Response Time	Standard deviation	Average Response Time	Standard deviation
10	31.3	2425	38	3912.5	77
30	100	2661	27	3917.1	38
100	320.1	2945	50	4098.7	71
300	826.7	3405	48	4414	39
1000	2414.2	4570	360	5662.6	31

Figure 5 below explains that streaming data transfer enhancement is still not enough for providing satisfactory large scale application performance. See Chapter 1.3.2 for the other proposed overall performance enhancement techniques.

We can deduce from the table that for the larger data sets when using streaming our gain is about 25%. But for the smaller data sets this gain becomes about 40% which is mainly because in the traditional Web Services the SOAP message has to be created, transported and decoded the same way for all message sizes which introduces significant overhead.

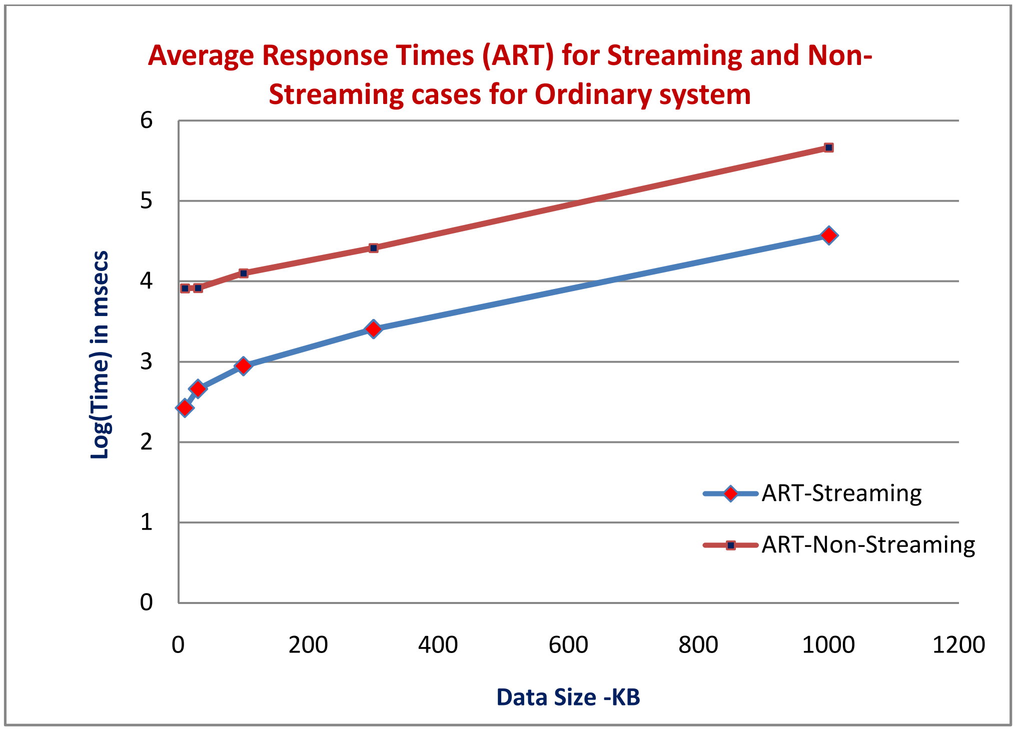


Figure 5: Comparisons of Streaming vs. Non-Streaming data response timings from source to AWMS.

1.3.1.2. Pull Parsing and Application Specific Rendering

Proposed system includes data rendering/filtering tasks assigned to Web-based Map Services to create comprehensible data representations derived from the semi-structured common data

(GML). These comprehensible representations are called maps. Regarding the rendering of large GML data and creating map images we use parsers.

There are three general parsing techniques proposed for processing XML structured data. These are document model, push model and pull model. There are also other hybrid alternatives built on these main approaches. In order to process data in XML structured common data model we use pull parsing technique.

Pull parsing, as exemplified by the XML Pull Parser [Alexander], is an efficient paradigm similar to SAX in that it does not build a complete object model in memory. It differs in that the tags and content are returned directly to the application from calls to the parser, rather than indirectly in the form of callbacks. The pull approach of this parsing model results in a very small memory footprint (no document state maintenance required – compared to DOM), and very fast processing (fewer unnecessary event callbacks - compared to SAX).

Pull parser only parses what is asked for by the application rather than passing all events up to the client application as SAX parsing does. You can see the article where pull parsing is compared with other leading Java based XML parsing implementations [Sosnoski].

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

For application specific comparison of Pull parsing and DOM see Table 3 and Figure 6. The performance values are measured in milliseconds and data sizes are in MBs. Performance test is done with 1GB allocated JAVA Virtual Machine. Dashed-line values in the table represent memory exceptions thrown. The figure illustrates the timing values for the data size till 100MB. Above this threshold value for the Virtual Machine allocated 1GB memory, DOM become useless.

Test case: For the XML data we use earthquake seismic data records encoded in GML. Each earthquake seismic record has some attributes and some geometry elements. In our tests we will parse the GML data in XML documents and extract the geometry elements. In case of DOM, parsing and extraction are done separate as it is shown in two columns in Table 3. In case of pull parsing, geometry data is extracted from GML with parsing and extraction applied all together.

Table 3: The performance values of DOM and Pull parsing (Xpp) over GML data. Dashed-line values imply memory exception.

	DOM (dom4j)					Pull Parsing	
Data Size (MB)	Average Parsing Time(msec)	StdDev	Average Rendering Time(msec)	Average Total Time Pars+Rend	StdDev	Average Total Time	StdDev
0.001	394.29	18.68	75	469	21.32	15.59	0.87
0.01	429.32	36.46	65	494	20.87	72.81	7.41
0.1	484.41	18.18	141	625	23.04	183.06	23.25
1	663.94	18.09	96	760	31.58	270.47	40.09

5	1,247.00	36.74	175	1,422	47.66	671.74	76.05
10	2,126.63	20.73	1,430	3,557	61.51	1,025.67	51.49
100	1,159,613.67	13,122.61	----	----	----	7,059.72	93.16
150	----	----	----	----	----	11,047.89	107.80
200	----	----	----	----	----	14,949.12	253.15

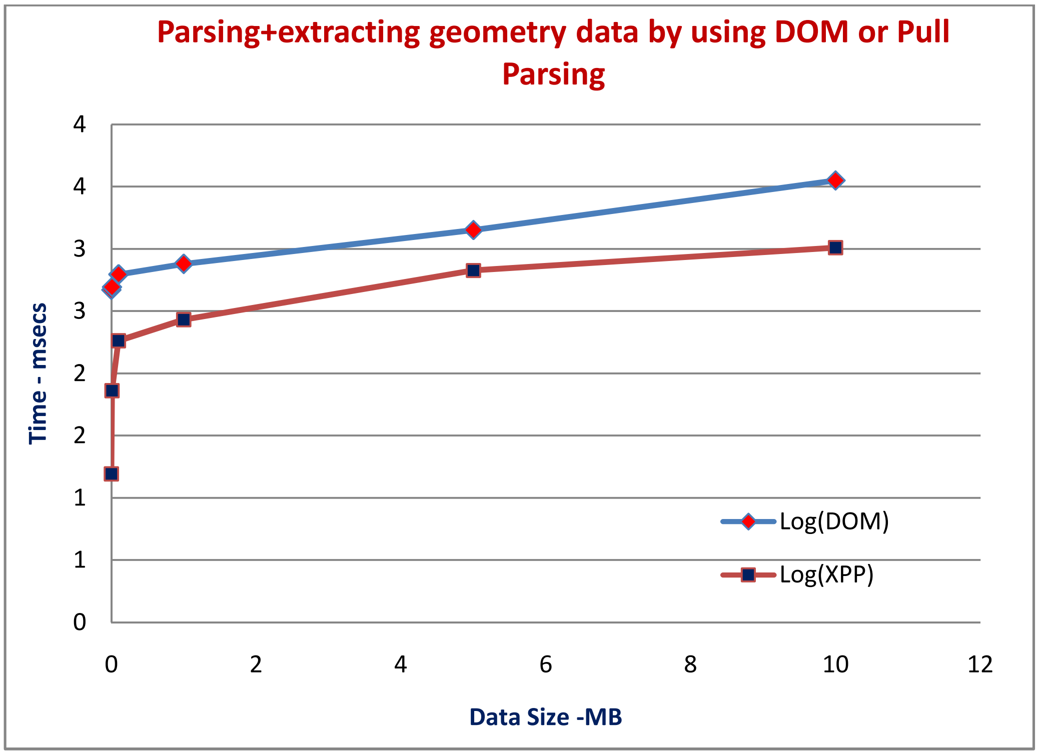


Figure 6: Performance comparison of two XML data processors, pull parsing and Document Object Model by using dom4j.

As it is mentioned dashed lines in Table 3 represent memory exceptions. It means system does not have enough memory for completing its work. Since there is extreme performance difference between using DOM and pull parsing techniques, we plot their logarithmic values to illustrate the performance gains of using pull parsing more clearly.

1.3.2. Federator (AWMS)-oriented Approaches

The federator (AWMS) in the proposed federated GIS system inherently enables load balancing and parallel processing and this helps with enhancing the overall system performance. This chapter presents the techniques and system design to develop high performance federated GIS system through the federator.

The proposed system design and applied techniques are based on WMS Aggregator (AWMS). AWMS is actually a WMS [WMS] with some extensions providing enhanced map rendering services by using innovative pre-fetching, parallel processing with caching techniques. AWMS aggregates, composes and orchestrates WMS and WFS services and, express the layer level compositions in its capabilities file by federating other services' capabilities metadata, and present it to the users through "GetCapabilities" Web Service interface.

The system design changes depending on the characteristics of the data application use. For the un-frequently changing data we propose pre-fetching (Chapter 1.3.2.1) technique. For the frequently changing data (similar to real-time data) we propose hybrid approach composed of caching and parallel processing techniques applied together (Chapter 1.3.2.2).

In summary, pre-fetching is purely for overcoming the natural bandwidth problem, caching helps the system with preventing to redo the jobs of querying and rendering, and parallel processing helps with workload sharing and parallel job run. Depending on the data characteristics, AWMS uses only one or the combination of these techniques. These techniques will be explained in the following sections.

1.3.2.1. Pre-Fetching

In the proposed integration framework we deal with the archived data in GML format. Archived data does not change often. Therefore, it is not reasonable transferring and rendering the same data again and again for every request coming from the different or even the same users. In order to solve this problem we use pre-fetching. Pre-fetching is used to overcome the performance degradation of transferring large sized data from source (database) to destination (WMS). It also indirectly enables getting rid of the data transformation overhead at WFS. As it is mentioned before, WFS transform any-data kept in databases into GML format every time it gets a request.

Pre-fetching is briefly defined as getting the data before it is needed. We accomplish the pre-fetching by the data transfer technique explained in Section 1.3.1.1. The general architecture for the pre-fetching is shown at Figure 7. A performance result of the pre-fetching and comparisons to the on-demand fetching techniques are displayed in Figure 8 and Figure 9 respectively. Since pre-fetching is independent of the real-time application and run in an asynchronous manner, it does not degrade the proposed framework's overall performance. It's running times defined by the periodicity parameter of the Pre-fetching module (PM) (see Figure 7).

The OGC's standard WMS and WFS specifications are based on HTTP Get/Post methods, but this type of services have several limitations such as the amount of data that can be transported, the rate of the data transportation, and the difficulty of orchestrating multiple services for more complex tasks. Web Services help us overcome some of these problems by providing standard interfaces to the tools and applications we develop.

As in the proposed data exchange framework defined in Section 1.3.1.1, the pre-fetching module make the requests with standard SOAP messages but for retrieving the results a NaradaBrokering subscriber class is used. Through the "*getFeature*" interface of WFS Web Services, pre-fetching module gets the topic name (publish-subscribe for a specific data), IP and port on which WFS streams the requested data. Second request is done by NaradaBrokering Subscriber using the returned parameters. GML data is provided by streaming WFS (implemented by G. Aydin) [Vretanos]. It uses standard SOAP messages for receiving queries from the clients; however, the query results are published (streamed) to a NaradaBrokering topic as they become available. In order to do that, we define the "task" and "timer". Task defines pre-fetching job, and timer defines the running periodicity of the task. Different data might have different periodicities set. Pre-fetching is done over the critical data. The critical data is the GML data affects the performance because of their sizes.

There will be two separate locations for the pre-fetched data. One is temporary into which pre-fetched data is stored. Another is stable which will be used for serving the clients' requests. Even if the system is busy with the pre-fetching job, it keeps itself up and running for the clients by using the stable storage. When the data transfer is done to the temporary location, all the data at that location will be moved to stable location. Reading and writing the data files at the stable locations will be synchronized to keep the data files consistent. This cycle is repeated at some time intervals pre-defined by periodicity parameter of Pre-fetching Module (PM).

In order for the pre-fetching algorithm to work properly, pre-fetching module fetches the data as a whole; no constraint should be defined in the query. On the other hand, the requests from clients contain some query constraints. These queries and their constraints are handled at the A WMS side. Queries are processed by using parser techniques and XPATH queries over the pre-fetched data.

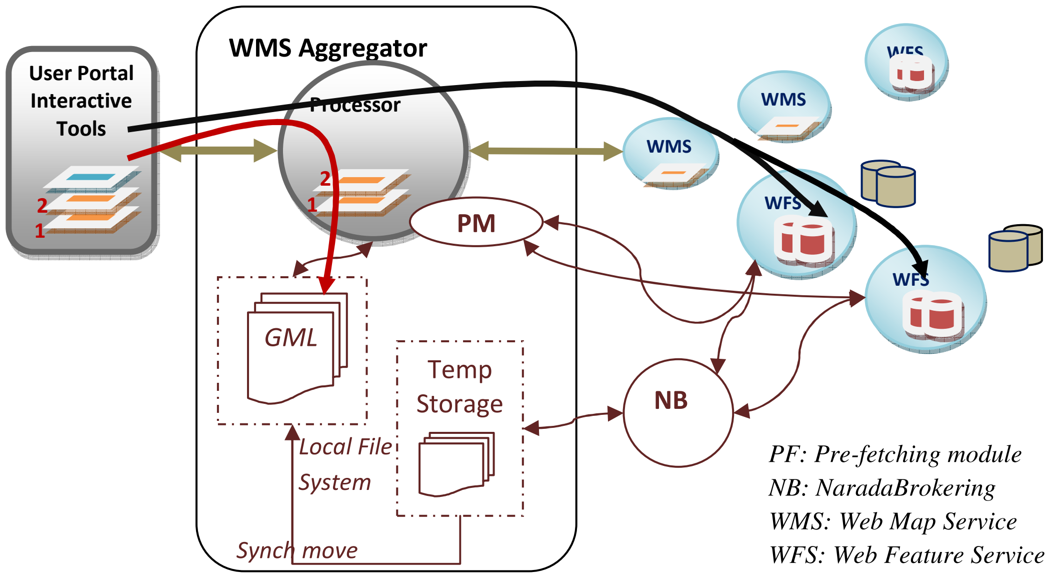


Figure 7: Pre-fetching architecture embedded to the federated GIS system

1.3.2.1.1. Fetching module (PM)

The pre-fetching module (PM) is composed of two components. One is "timer" defining the periodicity that PF will be running, and other is "task" defining what to do. The periodicity should not be less than the time to transfer one set of critical data. Assigning a periodicity at PM is the most critical task. This is defined under the considerations of data characteristics and developer's experience on the domain specific application.

Since the system is developed in JAVA, we use Timer and TaskTimer JAVA class libraries to implement the routinely running pre-fetching module.

Here is the "task" defined in a pseudo code:

```
public void pseudo_TASK() {
    Vector CDMdataList = new Vector();
    CDMdataList = getListPerformanceCritical_GMLDataNames();
    String tempDatastore = applpath + "/prefetchedData";
```

```
        String stableDatastore = applpath + "/prefetchedDataUsed";

        //Fetching all the data in CDM format (GML) - with NB

        fd.FetchDataWithStreaming( NBip,NBport,NBtopic,

                        wfs_address,tempDatastore,CDMdataList );

        //After pre-fetching is done move the data to stable storage

        fd.moveData(tempDatastore, stableDatastore);

    }
```

We also define timer determining the periodicity of task to run. The below sample code sets the periodicity of "task" defined above to three days. It means PF will be running once every three days.

```
        Timer timer = new Timer();

        timer.schedule(task, 0, 40000);
```

Timer class schedules the specified task for repeated fixed-delay execution, beginning after the specified delay. Subsequent executions take place at approximately regular intervals separated by the specified period.

There are two concerns in developing an efficient pre-fetching architecture. First one is limited storage capacity for a node. The size of the pre-fetched data is constrained by local node's storage capacity. Second one is regarding the pre-fetched data characteristics. Some archived data is updated so often that they look like real-time data. In that case, pre-fetching becomes un-feasible and cannot be benefited. For this type of data (archived but updated frequently), we propose a novel parallel processing approach applied together with the caching (see Chapter 1.3.2.2).

1.3.2.1.2. Performance Evaluation

We test the proposed pre-fetching technique over the proposed federated GIS system by using real-world Pattern Informatics (PI) geo-science applications (see Figure 7). PI is an earthquake forecasting application and uses archived earthquake seismic records stored at WFS as feature collections encoded in GML (XML encoded structured data model for geo-data).

We basically test the system as illustrated in Figure 7. Red-curve (short) illustrates the round-trip path for the pre-fetching and black-curve (long) illustrates the round-trip path for the on-demand fetching. For the simplicity we will be using only one critical data to apply pre-fetching.

In summary, we give the performance results for the proposed pre-fetching approach and compare it with the ordinary on-demand fetching approach in Figure 9 and Table 5. In case of on-demand fetching approach, one end is database and other end is user browser (see the black (dark)-curve in Figure 7). Performance results show the response times.

Table 4: Performance results for the response times when the pre-fetched data is used.

GML Data - MB	Average Processing	StdDev	Average Transfer	StdDev	Average Response	StdDev
0.001	961.35	179.92	26.21	6.56	1,006.47	176.84
0.01	1,011.67	233.28	39.13	9.21	1,040.33	233.24
0.1	1,110.00	233.83	38.44	9.57	1,148.44	233.11
1	1,655.56	421.22	31.89	8.22	1,687.44	421.92
10	2,754.58	281.07	30.79	7.64	2,785.37	282.39
20	4,097.89	228.86	28.68	9.25	4,126.58	227.85
40	7,002.61	219.75	31.22	11.90	7,039.11	220.47

60	10,096.32	148.61	32.00	11.60	10,128.32	146.68
80	12,900.94	361.10	26.50	14.15	12,927.44	380.99
100	16,019.50	373.06	31.30	12.54	16,050.80	373.72

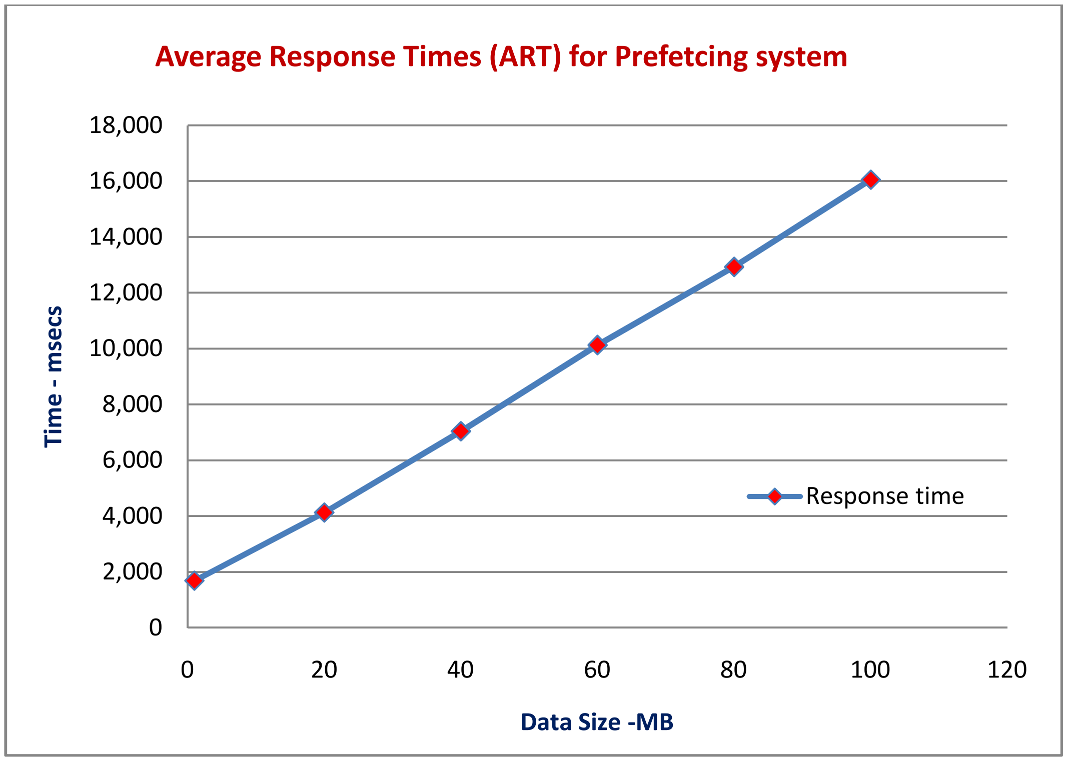


Figure 8: Performance of the pre-fetching technique

Table 5: Comparison of the pre-fetching (Figure 8) and ordinary (on-demand fetching -Figure-2) techniques

Data Size KB	Average Response Pre-fetching	StdDev	Average Response On-demand	StdDev
1	1,006.47	176.84	2,375.24	152.40
10	1,040.33	233.24	2,578.69	252.49
100	1,148.44	233.11	7,973.16	374.12
1000	1,687.44	421.92	59,335.69	343.76
10,000	2,785.37	282.39	573,324.66	836.46

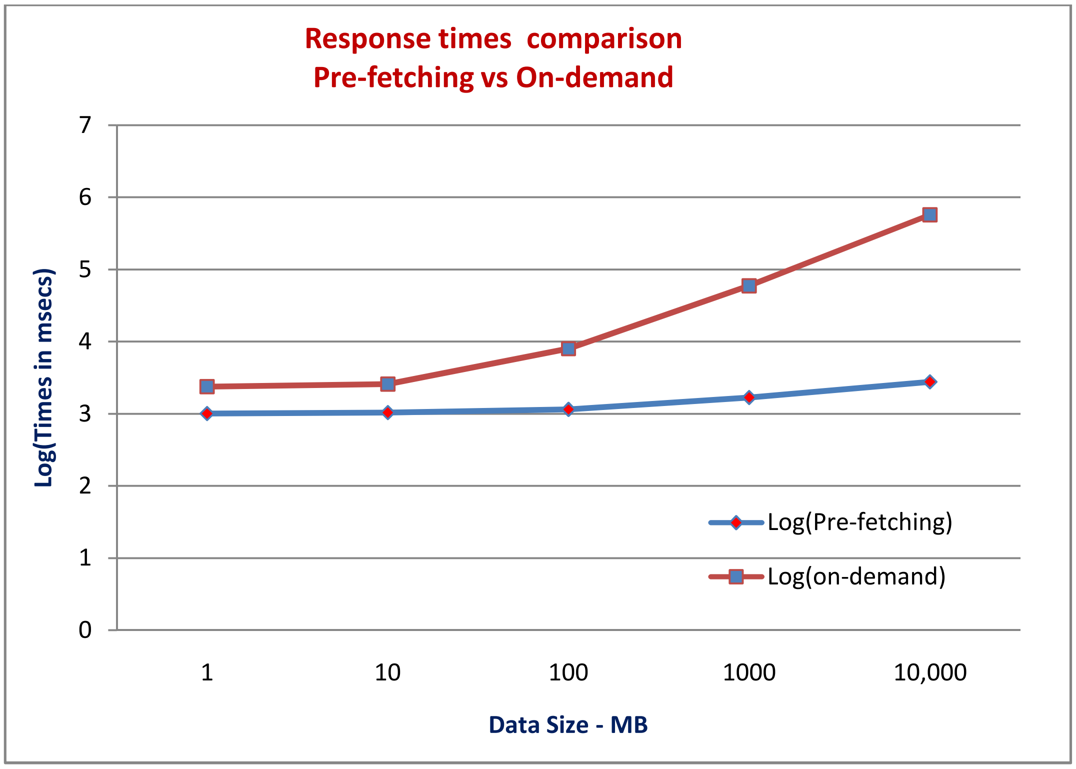


Figure 9: Performance comparison of the map rendering in the proposed GIS system with pre-fetching and ordinary ways.

As it is expected the pre-fetching increased the performance and responsiveness of the system for accessing, querying and rendering archived data. Compared to on-demand fetching (ordinary), pre-fetching removes the times spent on conversion (from database to GML at WFS side) and transferring GML data. In case of cascaded data (going through multiple chained services to access the original data source), performance gains even becomes much larger. Furthermore, the higher the data size, the higher the performance gains.

As mentioned before, this technique is good for only un-frequently changing data. For the frequently changing data we propose another technique explained in the following chapter.

Our criterion for selecting the technique to apply depends on two measurements. One is the minimum time required to fetch a whole critical data from the source and another is the time periodicity in which data is updated in its storage. If the data changes less than a time periods in which whole critical data is fetched, then the data is called frequently changing.

1.3.2.2. Locality-based Query Decomposition and Parallel Processing

This chapter presents another federator (AWMS) oriented high performance design for accessing and rendering of XML encoded large size data (GML).

The parallel processing is implemented based on the main query partitioning. Each partition is assigned to separate thread of work. The number of partitions and their sizes are defined by using locality principles. Locality information is obtained from the cached data kept for the same session and user. In order to achieve this, browser-based sessions are mapped to service-based sessions by updating the headers of the SOAP messages. All the services in the system are Web Services and they communicate through SOAP messages.

AWMS apply the parallel processing after the cached data extraction and rectangulation. Since all the data in the system is geo-referenced and queried in ranges defined by bounding boxes (defining coordinates of rectangles in the form of (minx, miny, maxx, maxy)), we do range query partitioning to implement parallel processing.

Parallel processing with caching algorithm has three parts closely related to each other:

1. *Caching*
2. *Cached-data extraction and Rectangulation*
3. *Parallel-processing*

In order to make these concepts more clear let's illustrate these in a picture (See Figure 10): Figure shows a map image composed of two layers. One is NASA satellite base map layer, and other is a layer showing earthquake seismic records (in blue dots). (a) shows partially overlapping of cached data and the main request bboxes. (b) Shows cached data extraction and rectangulation for the remaining part in the main request. (c) Shows partitioning of the rectangles from (b) based on the locality information obtained (explained later) from the cached data. All the rectangulated regions from (c) will be assigned to a thread to created map images as final responses.

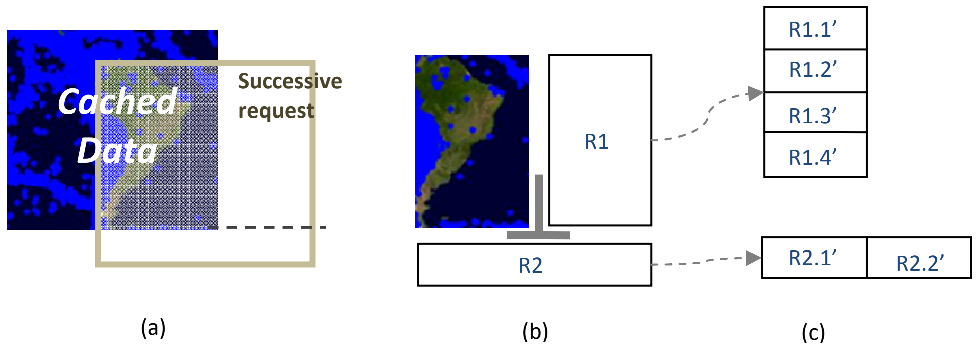


Figure 10: (a) Cached data extraction, (b) rectangulation, and (c) partitioning for parallel processing.

Research questions are summarized as below:

i. What to cache, how to cache and how to use

ii. How a server knows what requests come from what user to utilize cache

iii. How to map browser-based sessions across the servers.

iv. How to determine the number of the partitions required for the parallel processing

v. How to partition the rectangles (ex:R1 and R2)

vi. How to make locality-based query decomposition.

vii. How to create sub-queries for the partitioned regions (rectangles)

viii. How to assemble the results to sub-queries to create final response

These research questions are answered in the following two sub-sections. Chapter 1.3.2.2.1 explains the caching, cached data extraction and *rectangulation* processes as answers to i, ii, iii. Chapter 1.3.2.2.2 explains the parallel processing based on the range query partitioning and the locality principles as answers to iv, v, vi, vii, viii.

1.3.2.2.1. Caching, Cached-data Extraction and Rectangulation

We first explain the caching policy and techniques. Second, we introduce a novel approach for mapping browser-based session context to server-wide sessions in order to apply locality principles. Third, we explain techniques for cached data extraction and rectangulation.

1. *Caching:*

We apply the caching for only the critical data pre-defined in AWMS. Critical data is vector data encoded in GML common data model.

Caching will be utilized by the successive requests for the same user and session. For each separate session even from the same user there will be separate cached data kept at AWMS. To do that, we introduce browser-like sessions for each user across the servers. In other words, cached data is kept till the next request comes from the same session.

2. *Mapping Browser-based Session across the Servers.*

Since the proposed federated GIS is interacted through interactive decision making tools over the integrated data views, the session tracking and transfer issues have to be addresses to handle locality based query decomposition and parallel processing across the servers. Java Server Pages (JSP) defines session ID whenever a user opens a page to interact with the federated GIS system. A session is normally stored in a cookie which is available to all windows in the browser. The system access this ID by *session.getId()*. This returns a string unique user ID (uuid) which can be used for application specific purposes.

Whenever federated GIS client interacts with the system through AWMS, it sets its browser's session ID to the header of its SOAP messages sent to the Web Service. All the requests coming from the same browser has same session ID. Session IDs are created when the browser is opened and kept same until it is closed. Each browser has a separate and unique session ID. By setting this session ID to the header of SOAP messages AWMS can distinguish what client (browser) makes the requests and check its cached data and session information stored before.

Here is the pseudo code briefly explaining the steps:

```
WMSServicesSoapBindingStub binding;
binding = (WMSServicesSoapBindingStub)
              new WMServiceLocator().getWMSServices(new URL( service_address));
String sessionID = session.getid();
String channel_name = “WMS_getMap_Request”;
```

```
//Add SessionID to the SOAP message's header
binding.setHeader(service_address, channel_name, sessionID);
//See Appendix xx for the sample GetMap request
Object value = binding.getComprehensibleData(getMap_request);
```

Whenever a user access the system through the same browser its session number will be the same and AWMS keeps its local data and actions in the system differentiated based on its unique session ID.

3. *Cached Data Extraction and Rectangulation:*

According to OGC standards in GIS domain, queries are created with location parameter and location is defined in bounding box (bbox) formats. bbox is a formula defining the region as a rectangle through coordinates of bottom left corner and top right corner. Example: Q(minx, miny, maxx, maxy).

The proposed GIS services are OGC compatible and implemented in Web Service principles. They accept the requests in predefined XML-structured queries such as "getMap" for WMS and "getFeature" for WFS (see Chapter xx). Queries to WMS and WFS are actually window shape range queries. Range queries are formulated in bbox. After extraction of cached data falling in the main query range, the remaining part needs to be converted to the rectangular shapes in order to create valid sub-queries in the ranges defined by the bboxes. This is why we make rectangulation after cached data extraction from queried-region.

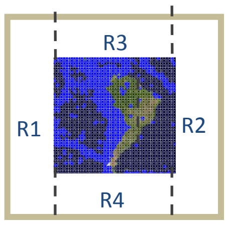
Cached-data extraction methodology is removing the regions in the main query which overlap with the cached data and then, creating the rectangular sub-regions from the remaining main query in the form of bboxes (see Figure 10 -b).

Since we use session-based caching (each session has separate cached data) based on the browser sessions, the position of the bboxes of the cached data and the following request determines our strategy for the cached data extraction and rectangulation. The bbox ranges of the cached data and main query can be positioned to each other in four possible ways.

Main query bbox is described as (minx, miny, maxx, maxy)

Cached data bbox is described as ($minx^c$, $miny^c$, $maxx^c$, $maxy^c$)

(1) The main query covers cached data (zoom-out action),



Meeting condition:

if($minx^c$>minx and $maxx^c$<maxx

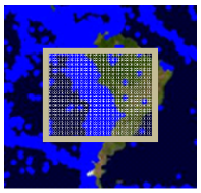
and $miny^c$>miny and $maxy^c$<maxy)

Figure 11: zoom-out case of rectangulation

Rectangles:

R1:	minx, miny, $minx^c$, maxy	R3:	$minx^c$, $maxy^c$, $maxx^c$, maxy
R2:	max^c, miny, maxx, maxy	R4:	$minx^c$, miny, $maxx^c$, $miny^c$

(2) The main query falls in cached data (zoom-in action)



Meeting condition:

if(minx>$minx^c$ and maxx<$maxx^c$

and miny>$miny^c$ and maxy<$maxy^c$)

Figure 12: zoom-in case of rectangulation

This case enables the fastest response. There is no need for query partitioning and data transfer from WFSs. It just uses cached GML to create map image based on the bboxes values of main query. A lot of performance gains.

(3) The main query partially overlaps with cached data (move action).

This case is explained in *Figure 10*, a and b.

Here is the formula of the rectangles:

R1: minx, miny, maxx, miny^c

R2: maxx^c, miny^c, maxx, maxy

In this case, there are four different sub-cases depending on the movement directions. These are (1) south-east, (2) south-west, (3) noth-east, and (4) north-west. The *Figure 10* illustrates the south-east case, and the rectangles above belong to that case. The rectangles for the other cases are also created similarly.

The rectangles obtained in this section go through the partitioning process explained in the following chapter.

1.3.2.2.2. Parallel-Processing with caching

The proposed parallel processing is based on range-query (defined in bbox) decomposition and throughout the document we call it partitioning. The partitioning is done with the locality principles to share the workload to the threads to reduce the response times to a reasonable level.

Partitioning will be done over the rectangular regions obtained at the end of rectangulation process. Issues to make parallel-processing with the locality principles are summarized as below:

i. How to determine the number of partitions and their sizes in bbox
ii. How to partition the rectangles
iii. How to create sub-queries corresponding to the partitions
iv. How to assemble the results to sub-queries for the main query

These issues are illustrated in Figure 13 below. In this specific example, main query includes three separate layers, and one of them is created with the critical data encoded in common data model, GML. The regions 1 and 2 in the main query are determined by the cached-data extraction and rectangulation processes explained in chapter 1.3.2.2.1. Grey region in the main query overlaps with the cached data. There is no need for data transfer for this region. This is obtained from the cache. For the other parts not overlapping with the cache (region 1 and 2), the system makes parallel processing for data access, query and plotting after creating partitions.

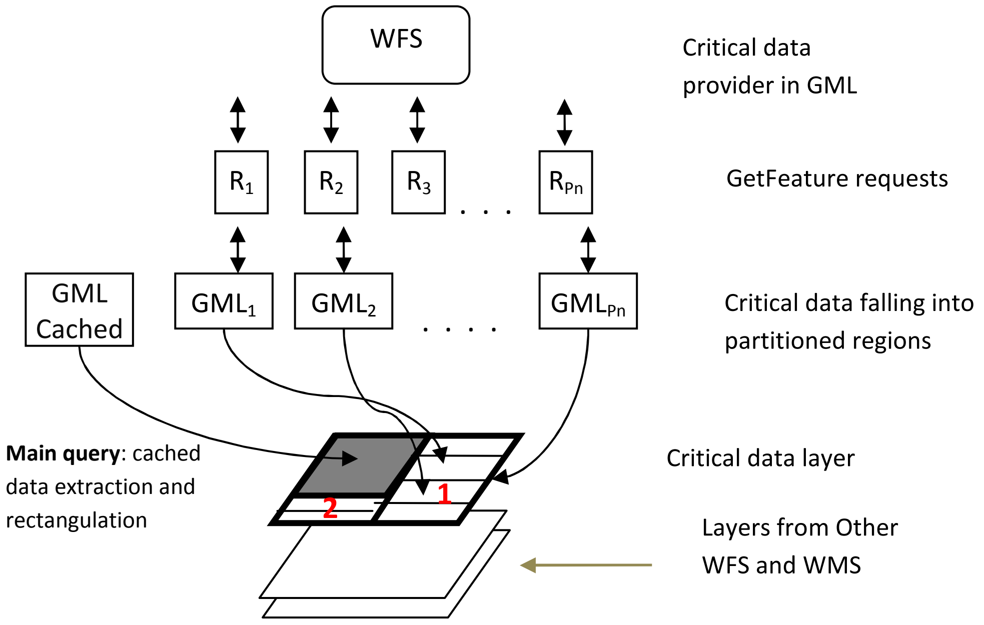

WFS
Critical data provider in GML
R1
R2
R3
RPn
GetFeature requests
GML Cached
GML1
GML2
GMLPn
Critical data falling into partitioned regions
Main query: cached data extraction and rectangulation
Critical data layer
2
1
Layers from Other WFS and WMS

Figure 13: Parallel processing and caching architecture in brief.

For the simplicity, we assume there are one critical data on which parallel processing and caching will be applied on, and one rectangle (obtained through the rectangulation process) to be partitioned in the system.

Terminologies to be used:

- *CD_size_kb*: Cached-data size measured in KB. Cached-data is kept in local file system as GML file. This is obtained from the system.
- *CD_size_br²*: Cached data bbox area.

$$CD\_size\_br^2 = (maxx^c - minx^c)*(maxy^c - miny^c)$$

- *R_size_br²*: Rectangle's bbox area (ex: Region-1 in critical data layer in Figure 13)

$$R\_size\_br^2 = (maxx - minx)*(maxy - miny)$$

- *Thr*: Threshold value. Allowed max data size falling in a partitioned region, and measured in KB.

Threshold value changes from data to data. It is predefined based on the experience of the developer. For example for the earthquake seismic data its value is 1000KB. This value defines a data size for which one thread (ordinary system) can response in a reasonable time period.

- Pn: the number of partition

Let's explain the research issues listed above.

i. Determining the number of partitions:

In order to define the number of partitions we use locality principles. Locality principle in this context is explained as following. If a region has a high volume of data, then the regions in close neighborhood also expected to have high volume of data. Example is the human population data. The urban areas have higher human population than the rural areas. The oceans (2/3 of the world) have no populations etc.

We partition the rectangles into equal regions because we don't know the size of the data falling in that region before getting it. In order to define the size (in bbox) we use cached data sizes expressed in bbox and KB. We assume (by using locality) cached data density is similar to the main request density, and by using the threshold value and un-cached main request part we calculate Pn as below:

Density of cached-data: $dcd = \frac{CD\_size\_kb}{CD\_size\_br^2}$

Allowable largets area to assign: $lat = \frac{threshold\ data\ size}{dencity\ of\ cached\ data} = \frac{thr}{dcd}$

$$Pn = \frac{R\_size\_br2}{largest\ area\ to\ assign} = \frac{R\_size\_br2}{lat}$$

If Pn is less than 1 then, don't make partition. In contrast, if it is bigger than 1, then partition into Pn regions. ii explains how to partition a rectangle into Pn number of regions.

ii. Query decomposition of the rectangulated regions with Pn:

We already know the bbox of the rectangles obtained through rectangulation process explained at Chapter 1.3.2.2.1. We also calculated the number of regions (in item i) into which each one of these rectangles is going to be partitioned as the set of bbox values.

Here, we explain how to partition a given rectangle into Pn number of bboxes. There are two alternative techniques here, one is partitioning the rectangle vertically and the other is partitioning it horizontally.

In case of vertical partitioning the step value is calculated as below, and partitioning is done along the Y-coordinate. (See Figure 14)

$$s_y = \frac{(\text{maxy} - \text{miny})}{Pn}$$

In case of horizontal partitioning the step value is calculated as below, and partitioning is done along the X-coordinate,

$$s_x = \frac{(\text{maxx} - \text{minx})}{Pn}$$

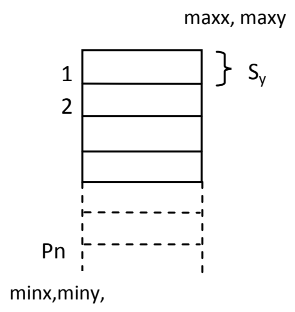


Figure 14: Partitioning a rectangle along the coordinate-y

Calculating the bboxes of the partitioned regions:

Vertical:

```
for (i=0; i<Pn*s_y; i=i+s_y;)
    print ( minx, miny – i, maxx, maxy-(i+s_y) ) ;
```

Horizontal:

```
for (i=0; i<Pn*s_x; i=i+s_x;)
    print ( minx-i, miny, maxx-(i+s_x), maxy) ;
```

After having the rectangles partitioned, the partitions go through the query creation process explained as following.

iii. Creating the queries for the sub-regions obtained at ii.

Throughout the rectangulation and partitioning, the only changing attribute of the main query is the bbox coordinate value. These are calculated in the previous step.

Based on the set of bbox values obtained at the end of partitioning process (ii) we need to create sub queries. Each partition is differentiated by only their bbox value, and they go through the query creation process. AWMS creates *getFeature* requests corresponding to these rectangles based on their bounding boxes. Other parameters and attributes required for creating *getFeature* request are obtained from the main query. All the parameters, attributes and their values (except for bbox values) will be the same for all the getFeature requests created for the partitions.

An example case of decomposing a rectangle obtained by rectangulation process and creating parallel queries is illustrated at Figure 15. In this example, rectangle is partitioned into 5 regions vertically.

$$\text{Pn} = 5 \quad \text{and} \quad s_y = \frac{(maxy - miny)}{Pn} = \frac{(45-40)}{5} = 1$$

You can see a sample getFeature created for bbox value "-110, 35, -100, 40" request at Figure 16. For the overhead coming from the sub-query creation explained here, see Table 8 and Figure 18.

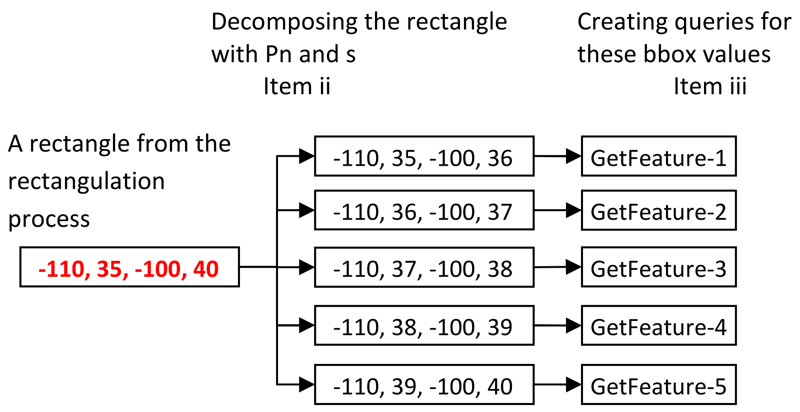
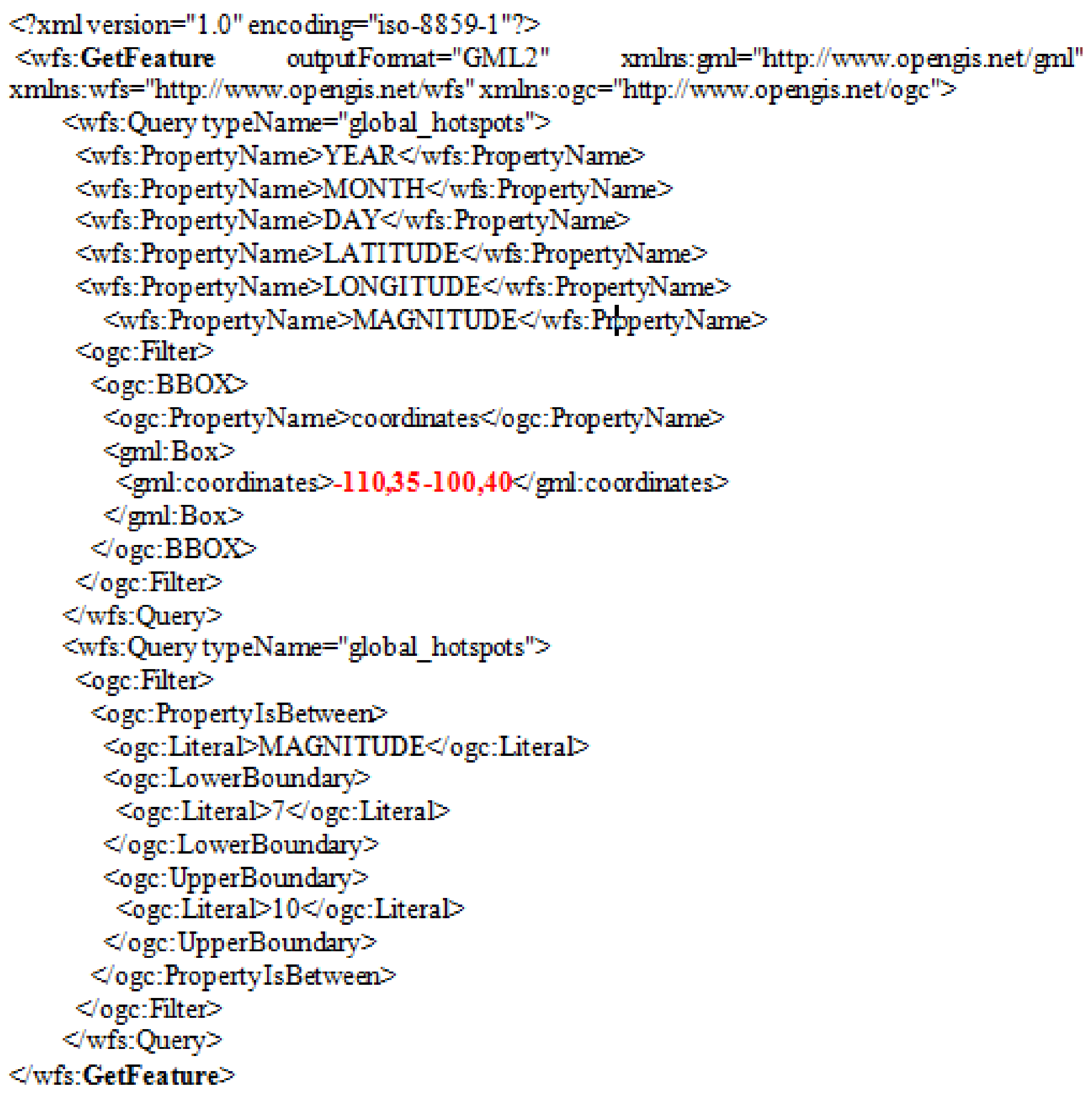


Figure 15: Example scenario of the partitioning a region into 5 sub-regions through the bbox value of a rectangle.

iv. Assembling the results from the sub-queries

Each query created at step iii is sent to WFS in a separate thread, and the returned results are stored locally at AWMS. After getting the data AWMS starts rendering and plotting the critical data over the other layers by parsing and extracting the geometry elements in returned GML. Data transfer, parsing and rendering issues are explained in the previous chapters.

In addition, all the GML data corresponding to main query for the specific bbox is kept as cached data in order to serve the following request coming from the same session.



```xml
<?xml version="1.0" encoding="iso-8859-1"?>
 <wfs:GetFeature     outputFormat="GML2"     xmlns:gml="http://www.opengis.net/gml"
xmlns:wfs="http://www.opengis.net/wfs" xmlns:ogc="http://www.opengis.net/ogc">
    <wfs:Query typeName="global_hotspots">
     <wfs:PropertyName>YEAR</wfs:PropertyName>
     <wfs:PropertyName>MONTH</wfs:PropertyName>
     <wfs:PropertyName>DAY</wfs:PropertyName>
     <wfs:PropertyName>LATITUDE</wfs:PropertyName>
     <wfs:PropertyName>LONGITUDE</wfs:PropertyName>
       <wfs:PropertyName>MAGNITUDE</wfs:PropertyName>
     <ogc:Filter>
      <ogc:BBOX>
       <ogc:PropertyName>coordinates</ogc:PropertyName>
       <gml:Box>
        <gml:coordinates>-110,35-100,40</gml:coordinates>
       </gml:Box>
      </ogc:BBOX>
     </ogc:Filter>
    </wfs:Query>
    <wfs:Query typeName="global_hotspots">
     <ogc:Filter>
      <ogc:PropertyIsBetween>
       <ogc:Literal>MAGNITUDE</ogc:Literal>
       <ogc:LowerBoundary>
        <ogc:Literal>7</ogc:Literal>
       </ogc:LowerBoundary>
       <ogc:UpperBoundary>
        <ogc:Literal>10</ogc:Literal>
       </ogc:UpperBoundary>
      </ogc:PropertyIsBetween>
     </ogc:Filter>
    </wfs:Query>
</wfs:GetFeature>
```

Figure 16: Sample GetFeature request for the partitioned region of bbox (-110, 35 -100, 40).

1.3.2.2.3. Performance Evaluation

Performance will be evaluated in three possible generalized situations categorized based on the cached data utilization. These are (as explained in different context in Chapter 1.3.2.2.1):

a. No usage of cached data
b. Complete usage of cached data. No need for parallel processing.
c. Partial usage of cached data (case:1/2 cached). Case is illustrated in Figure 13.

a. **No** usage of cached data:

This case happens when the successive query is randomly created and does not overlap with the cached data. In this case there is no cached data extraction and sub-rectangle created by rectangulation processes. There is only one rectangle which is the main query. In order to make performance evaluations, we test the system with different levels of partitions such as two and ten and assign them to separate individual threads for the parallel processing.

Since the major bottleneck of the performance is transferring GML data we first demonstrate the performance enhancement in data transfer (see Table 6 and Figure 17). The data is transferred from the databases through WFSs to the federator (AWMS). The measured transfer times are in milliseconds. Queries are decomposed based on their bbox values. Query bbox values and corresponding data sizes are displayed in Table 6.

Table 6: The performance results for various levels of partitioning and parallel processing.

GML Data Size -MB	Sample Bboxes of requests				Average GML-data Transfer Timings		
	minx	miny	maxx	maxy	Single-thred	2 threads	10 threads
0.01	-122.05	35.02	-121	35.65	2,632.83	2,773.94	6,498.76
0.1	-121.05	35.02	-120.39	36.15	7,833.16	6,498.12	7441.4375
0.5	-121.05	35.02	-119.8	36.99	30,415.09	18,860.06	13,095.76
1	-121.05	35.02	-118.07	37.15	58,996.75	31,210.44	19,829.11
5	-122.08	34.99	-116.44	40.23	288,095.42	229,499.89	98,660.00
10	-124.85	32.26	-113.56	42.75	573,934.20	395,691.75	159,055.00

Table 7: The standard deviations of data transfer timings in Table 6.

GML Data Size -MB	Standard Deviations		
	Single-thread	2 threads	10 threads
0.01	34.46	92.20	146.76
0.1	106.43	82.15	138.6915
0.5	125.17	125.17	277.66
1	134.87	180.03	310.56
5	181.07	160.99	214.55
10	270.82	368.11	249.80

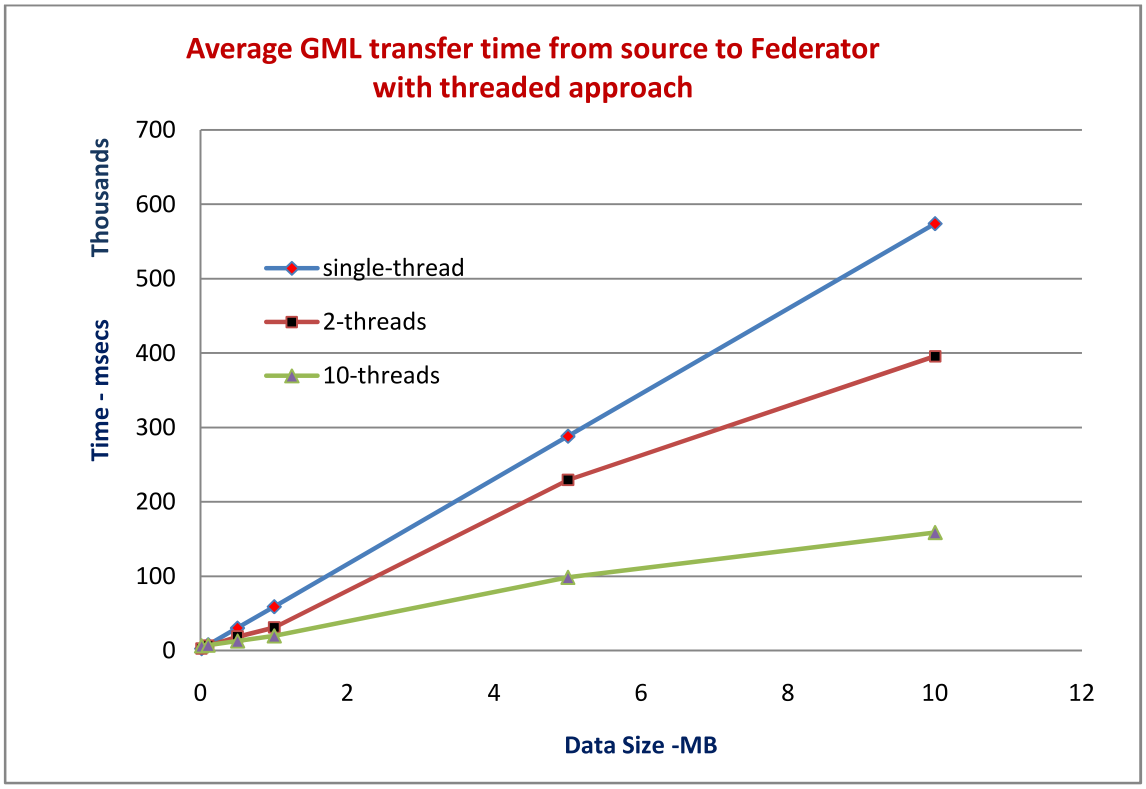


Figure 17: Response times in seconds for different levels of threaded run (2, 4, 8, 20), and data sizes.

We also need to compare the response times. Because proposed parallel processing causes some overheads in query partitioning, corresponding query creation and assembling the data to create resonse for the main query. As a sample case we chose 10-threaded case and compare the performance results below.

Detailed overhead timings and the response times in case of using proposed system:

Here is the more detailed analysis in terms of parallel processing's overhead timings for the case of partitioning the query into ten through the proposed partitioning techniques for parallel processing. We first measure and analyze the overhead times (see Table 8, Table 9 and Figure 18) and then compare the overall performance expressed in response times with the on-demand, single threaded general systems (Table 10 and Figure 19). The general system performance is given in Table 1

Table 8: The average overhead timings for the proposed parallel processing (10 threaded).

	Average Timings					
GML Data MB	Creating Partitions	Creating sub-queries	GML from Source to AWMS	Map-Image To User	Rendering map-data	***Total**-time Response*
0.01	438.00	103.06	6,498.76	30.79	72.81	*7,143.42*
0.1	449.26	104.65	7,441.44	31.12	183.06	*8,209.53*
0.5	419.63	101.56	13,095.76	31.89	192.34	*13,841.18*
1	453.17	104.32	19,829.11	33.45	270.47	*20,690.51*
10	452.22	115.55	159,055.00	39.13	1,025.67	*160,687.57*

Table 9: The standard deviations of the overhead items' timings given in Table 7.

	Standard Deviations					
GML Data MB	Creating Partitions	sub-queries	GML from Source to AWMS	Map-Image To User	Rendering map-data	***Total**-time Response*
0.01	32.61	14.29	146.76	7.64	7.41	*208.72*
0.1	26.94	11.57	138.69	7.95	23.25	*208.40*
0.5	22.61	9.71	277.66	8.22	24.47	*342.68*
1	29.85	10.43	310.56	8.25	40.09	*399.18*
10	28.02	15.65	249.80	9.21	51.49	*354.16*

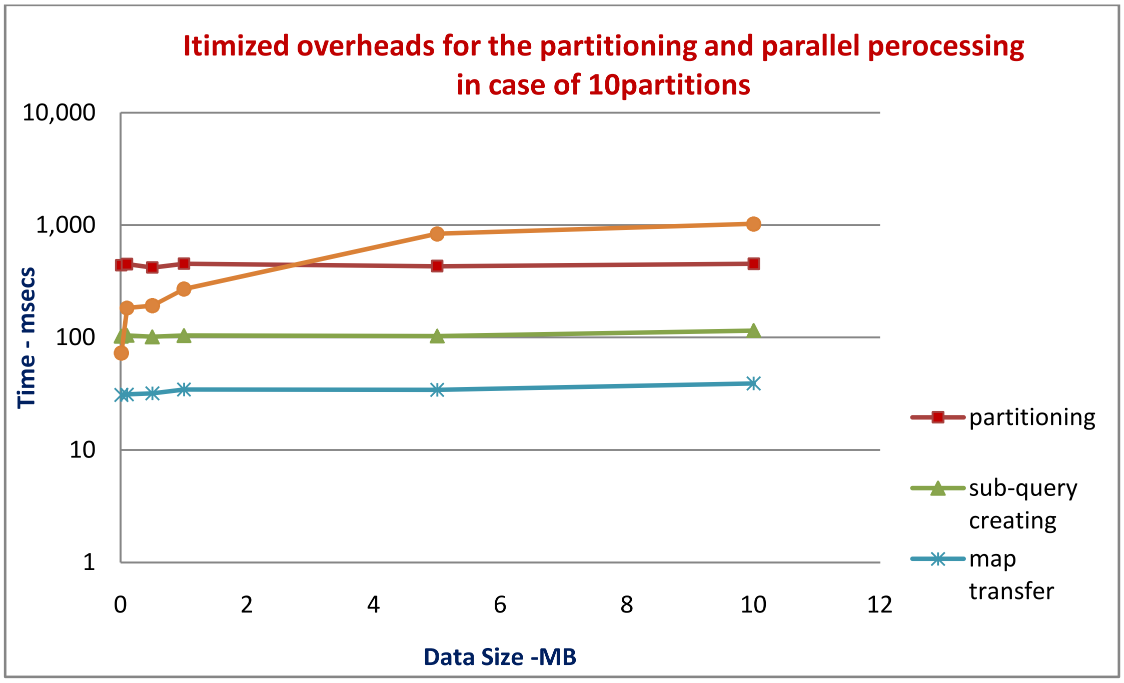


Figure 18: Overhead timings for the cases of partitioning into 10.

Only the timings of "rendering map data" depends on the data size, others are independent of data size but changes depending on the number of partitions.

Comparing the performance of the total response times again for the cases of 10 partitions:

Table 10: Comparing response times of 10-threaded parallel processing and single threaded ordinary system (also see *Figure 2*).

	Comparison of the average response timings			
GML Data Size - MB	PP with 10 Threads time	StdDev	Orinary systems time	StdDev
0.01	7,143.42	208.72	2,578.69	252.49
0.1	8,209.53	208.40	7,973.16	374.12
0.5	13,841.18	342.68	30,868.52	482.83
1	20,690.51	399.18	59,635.69	343.76
10	160,687.57	354.16	574,825.16	836.46

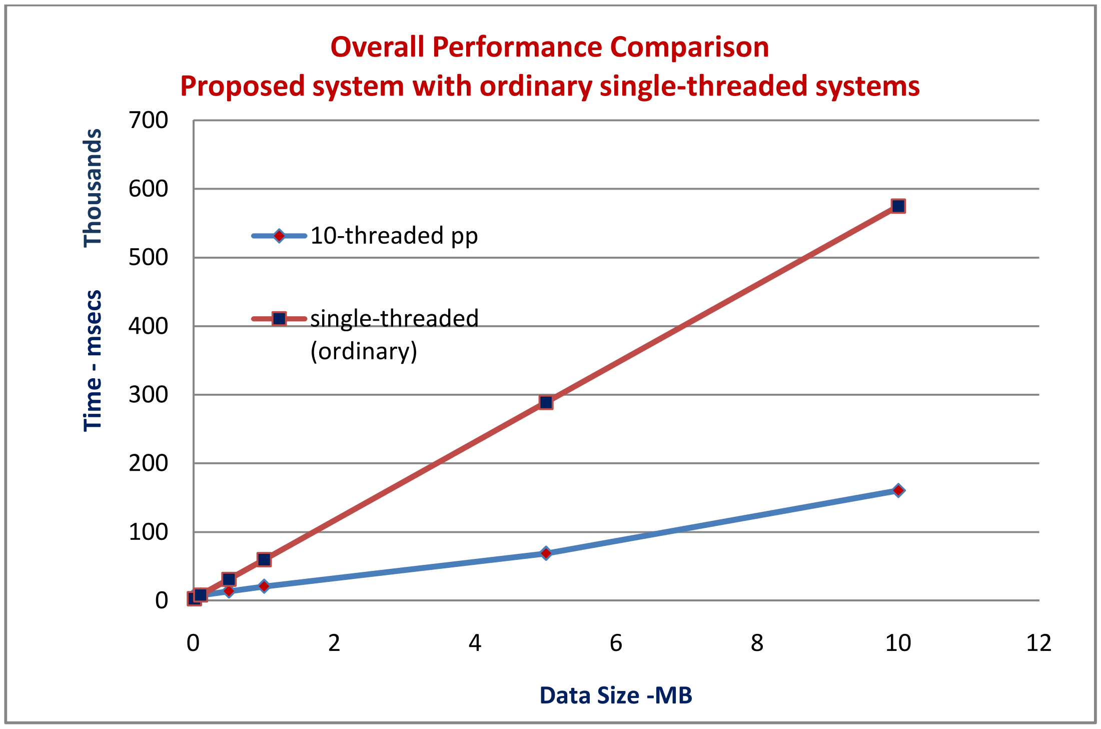


Figure 19: Performance comparison, parallel processing vs ordinary system.

Note that if the density and/or data size increase or the data is distributed more equally, then the performance gain will be higher.

As it is shown in first two lines of Table 10, there is no gain of using parallel processing for small sizes of data. For the requests over the small data sizes, total overhead times sometimes get higher than the total response times of single threaded cases.

The test results show that 10-threaded parallel processing for the data handling is almost 4-times faster. However, the performance does not increase in the same ratio at which the thread number increases. That is because of the overheads resulted from mainly the query decomposition and assembling the result sets for the main query etc. Moreover, the figure shows that the higher the data size the larger the performance gains.

b. **Complete** cached data utilization

In this case there is no need for rectangulation, the main query decomposition and threaded parallel processing. This case happens when the user query a smaller region falling in the previous map he got on his browser. It is mostly caused by zooming-in action. In this case, the cached data is enough for responding to the main request, and no other cascaded requests are needed. AWMS renders the map just by using the cached data. The only task needed is the cached data extraction and overlaying (plotting) over the other requested layers.

This case's performance results are same as the pre-fetching technique's performance results. Please see Figure 8 and Figure 9.

Since the most of the time is spent on data transfer and GML conversion (from any data to common data model at WFS), getting rid of these burdens by using cached data enhances the performance a lot. Note that, in order to show huge performance difference in one graph clearly, the ordinary system performance results are adjusted by taking their logarithmic values.

c. Partial cached data utilization (case:1/2 cached):

This case happens when the user moves (or drag and drop) the map to another region or makes zooming-out. In other words, when the user makes successive requests and their bbox values overlap. As explained before (*Figure 10* and Chapter 1.3.2.2.1), if only one point of main request falls in bbox boundaries of the cached-data, they are called as partially overlapped. Table 11 lists the sample request bboxes and their corresponding data sizes.

Here is the test scenario for the illustration of performance gains by using parallel processing with caching: it is assumed that 1/2 of the requested data is already cached. In other words, main request bbox values partially overlap with the cached data and half of the data size corresponds to main request is provided by the cached data. Other half is going to be requested by 10-threaded parallel processing. This is a kind of improvement over Table 10 and Figure 19. Instead of making processing for whole data we utilize cached data and make request for other half by using the proposed hybrid technique explained in Chapter 1.3.2.2.1.

Table 11 shows the average timing values for the selected sample bbox values and data sizes and Table 12 shows the corresponding standard deviation values. The cached data is accessed and processed in a way similar to the way of processing the pre-fetched data except for the pre-fetching part. The remaining un-cached data is queried and got from the cascaded WFS services in GML format. In order to access the data 10-threaded parallel processing is used. The number of partitions is arbitrarily chosen. The data size values given in the parenthesis are cached data sizes utilized by the system. Other values are the sizes of the main queries.

Table 13 shows the detailed steps and timings for the parallel processing timings given in sixth column of Table 11. It is actually a re-creation of Table 8 for the different data sizes.

Table 11: Performance results for the sample case scenario in which half of the data is provided by the cached data, and other half is requested by again 10-thread parallel processing.

GML Data Size -MB	Sample bounding boxes after rectangulations on which partitioning is done minx	miny	maxx	maxy	Average Timings cached-data processing	Remaining data 10-thread pp	Total Response
0.01(0.005)	-121.58	34.55	-121.45	34.69	734.33	6,329.05	7,063.38
0.1(0.05)	-121.65	34.36	-120.78	35	768.33	8,934.16	9,702.49
0.5(0.25)	-118.68	34.21	-118.39	34.5	782.45	12,109.67	12,892.12
1(0.5)	-119.16	34.21	-118.25	35.12	851.00	13,841.18	14,692.18
5(2.5)	-120.83	32.07	-117.15	36.18	1,209.79	44,191.61	45,401.40
10(5)	-120.83	32.07	-115.7	36.7	1,646.32	68,848.66	70,494.98

Table 12: The standard deviations for the timing values of cached data processing and remaining-data processing through parallel processing.

	Standard Deviation		
GML Data Size -MB	cached-data processing	remaining-data access and proc	Total Response
0.01(0.005)	62.45	395.01	457.46
0.1(0.05)	66.37	325.83	392.20

0.5(0.25)	79.56	281.97	361.53
1(0.5)	72.21	342.68	414.89
5(2.5)	68.52	522.37	590.89
10(5)	94.57	380.62	475.19

Table 13: etailed timings for the remaining data rendering with 10-threaded parallel processing given in Table 11

	Average Timings					
GML Data Size - MB	Creating Partitions	Creating sub-queries	GML from Source to AWMS	Map-Image To User	Rendering map-data	***Total**-time Response*
0.005	438.00	103.06	5,706.08	29.39	52.52	*6,329.05*
0.05	449.26	104.65	8,204.67	32.52	143.06	*8,934.16*
0.25	403.70	101.56	11,403.75	30.33	170.34	*12,109.67*
0.5	419.63	101.56	13,095.76	31.89	192.34	13,841.18
2.5	431.52	103.29	43,136.50	35.61	484.69	*44,191.61*
5	431.52	103.29	67,445.06	34.24	834.55	68,848.66

Table 14 and Figure 20 compare the performance results of the hybrid system (parallel processing with caching) with ordinary on-demand system for the selected 10 partitioning case. Ordinary system's performance results are from Table 1 and parallel processing with caching performance results are from Table 11.

.

Table 14: Comparison of the response times for the hybrid (caching and parallel processing) and ordinary non-caching single-threaded system.

	Comparison of the response times			
GML Data	Hybrid (half cached half pp)		Orinary systems	
Size - MB	time	StdDev	time	StdDev
0.01	7,063.38	357.46	2,578.69	252.49
0.1	9,702.49	322.20	7,973.16	374.12
0.5	12,892.12	361.53	30,868.52	482.83
1	14,692.18	414.89	59,635.69	343.76
5	45,401.40	590.89	288,594.12	772.41
10	70,494.98	475.19	574,825.16	836.46

As it is shown in first two lines of Table, there is no gain of using of hybrid techniques which we call as parallel processing with caching for the small sizes of data. In such cases, total overhead sometimes get higher than the total response times of single threaded cases. This problem is solved by using a threshold value to define if the partitioning is needed or not. This technique is also explained in 1.3.2.2.

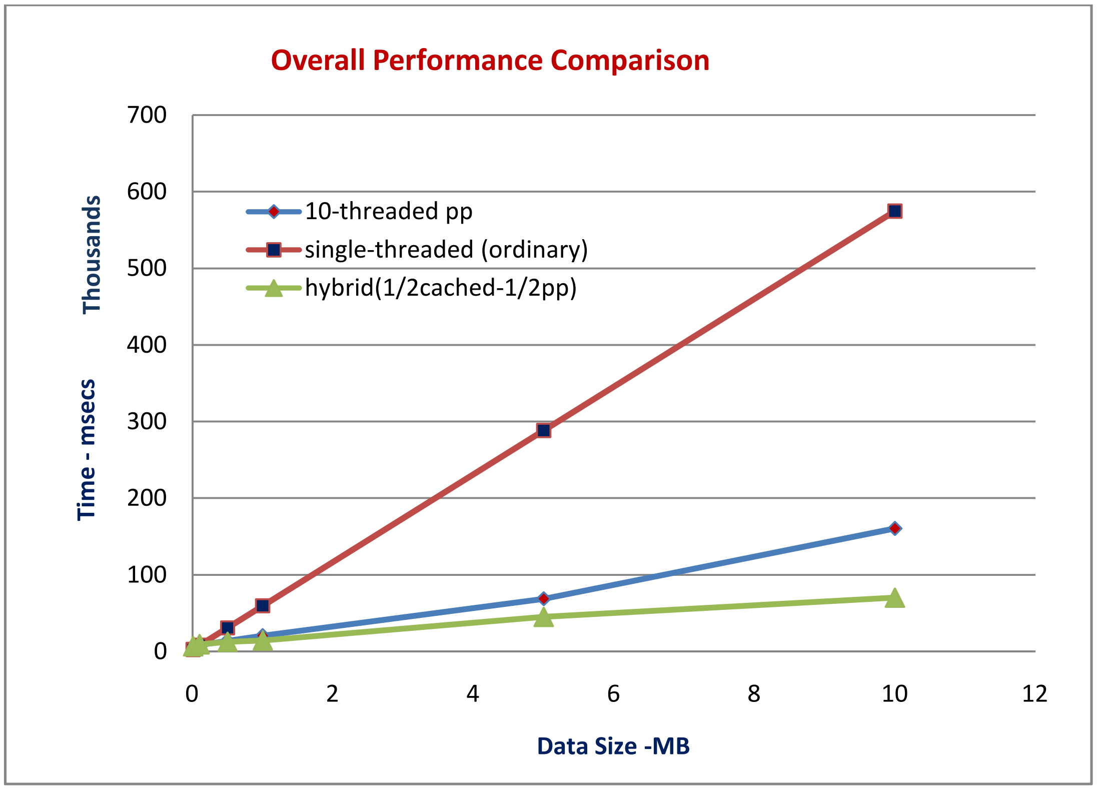


Figure 20: Comparing response times of the hybrid (caching and parallel processing) and ordinary system.

As it is shown the in Figure 20, for the given test scenario (1/2 of main query matches to the cached data and remaining data is obtained and processed with 10-threaded parallel processing) proposed system is almost 8 times faster than the ordinary on-demand one-threaded system. As the data size increases, that ratio increases.

REFERENCES

[Lu 2006] Wei Lu, Kenneth Chiu, and Yinfei Pan, "A Parallel Approach to XML Parsing". In *the 7th* IEEE/ACM International Conference on Grid Computing*, 2006.*

[Pallickara2003] Pallickara S. and Fox G., "NaradaBrokering: A Distributed Middleware Framework and Architecture for Enabling Durable Peer-to-Peer Grids" ACM/IFIP/USENIX, Rio Janeiro, Brazil June 2003.

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

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

[Alexander] Aleksander Slominski. XML Pull Parser, visited 04-15-02. http://www.extreme.indiana.edu/xgws.

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

[GML] Cox, S., Daisey, P., Lake, R., Portele, C., and Whiteside, A. (eds) (2003), OpenGIS Geography Markup Language (GML) Implementation Specification. OpenGIS project document reference number OGC 02-023r4, Version 3.0.

[WMS] de La Beaujardiere, J., Web Map Service, OGC project document reference number OGC 04-024. 2004.

[WFS] Vretanos, P. (2002) Web Feature Service Implementation Specification, OpenGIS project document: OGC 02-058, version 1.0.0. Volume, [Booth]Booth, D., Haas, H., McCabe, F., Newcomer, E., Champion, M., Ferris, C., and Orchard, D. "Web Service Architecture." W3C Working Group Note, 11 February 2004. Available from http://www.w3c.org/TR/ws-arch.

[Tran] Tran, P., Greenfield, P., and Gorton, I., *Behavior and Performance of Message-Oriented Middleware Systems.* . Proceedings of the 22nd international Conference on Distributed Computing Systems, ICDCSW. 2002.

[uddi] Bellwood, T., Clement, L., and von Riegen, C., UDDI Version 3.0.1: UDDI Spec Technical Committee Specification http://uddi.org/pubs/uddi-v3.0.1-20031014.htm. 2003.

[ogc] The Open Geospatial Consortium, Inc. web site: http://www.opengeospatial.org

[deegree] deegree project web site available at http://deegree.sourceforge.net/

[minmapserv]University of Minnesota Map Server, available at http://mapserver.gis.umn.edu/

[patterninfo] Tiampo, K. F., Rundle, J. B., McGinnis, S. A., & Klein, W. Pattern dynamics and forecast methods in seismically active regions. Pure Ap. Geophys. 159, 2429-2467 (2002).