

A Comparison of Raster Big Data Benchmarking on Geoserver and Rasdaman

by

Ankit Kumar Karn

Bachelor Thesis in Computer Science

Submission: May 23, 2022

Supervisor: Prof. Peter Baumann

Statutory Declaration

Family Name, Given/First Name	Karn, Ankit Kumar
Matriculation number	30002475
Kind of thesis submitted	Bachelor Thesis

English: Declaration of Authorship

I hereby declare that the thesis submitted was created and written solely by myself without any external support. Any sources, direct or indirect, are marked as such. I am aware of the fact that the contents of the thesis in digital form may be revised with regard to usage of unauthorized aid as well as whether the whole or parts of it may be identified as plagiarism. I do agree my work to be entered into a database for it to be compared with existing sources, where it will remain in order to enable further comparisons with future theses. This does not grant any rights of reproduction and usage, however.

This document was neither presented to any other examination board nor has it been published.

German: Erklärung der Autorenschaft (Urheberschaft)

Ich erkläre hiermit, dass die vorliegende Arbeit ohne fremde Hilfe ausschließlich von mir erstellt und geschrieben worden ist. Jedwede verwendeten Quellen, direkter oder indirekter Art, sind als solche kenntlich gemacht worden. Mir ist die Tatsache bewusst, dass der Inhalt der Thesis in digitaler Form geprüft werden kann im Hinblick darauf, ob es sich ganz oder in Teilen um ein Plagiat handelt. Ich bin damit einverstanden, dass meine Arbeit in einer Datenbank eingegeben werden kann, um mit bereits bestehenden Quellen verglichen zu werden und dort auch verbleibt, um mit zukünftigen Arbeiten verglichen werden zu können. Dies berechtigt jedoch nicht zur Verwendung oder Vervielfältigung. Diese Arbeit wurde noch keiner anderen Prüfungsbehörde vorgelegt noch wurde sie bisher veröffentlicht.

.....
Date, Signature

Abstract

There are a lot of geospatial information produced daily in geographical domain. This data could be natural observations or simulations, coming from a lot of sensor devices, could be in vector or raster format. In this paper, we are going to investigate about raster data which is suitable for mathematical model and analysis and allows researchers to store and analyze geospatial data. There are different platforms currently present that helps to store and analyze these raster data. But there is lack of efficient benchmarking results that analyzes performance of these platforms and publish the results so, that geospatial researchers could make an informed decision while choosing the platform. It has been primary focus of my thesis and in this thesis, we have analyzed performance of two popular platforms for raster data i.e. geoserver and rasdaman. Geoserver is an opensource server for providing geospatial data whereas Rasdaman(Raster Data Manager) is a tool that provides whole spectrum of functionality on spatio-temporal raster data. So, we use benchmarking tools to perform spatial operations on these platforms and measure how efficiently these platforms could execute these geospatial operations.

Contents

1	Introduction	1
1.1	Geoserver	1
1.2	Rasdaman	2
1.3	Geospatial Operations	3
2	Related field of work	4
3	Methodology	5
3.1	Data and Hardware specification	5
3.2	Creating Coverages	6
3.2.1	Creating Coverage in Geoserver	6
3.2.2	Creating Coverage in Rasdaman	7
3.3	Benchmarking tools	9
3.4	Setting up Benchmarking environment	9
3.4.1	Setting up parameters for WMS request in Jmeter	10
3.4.2	Setting up benchmark in hyperfine	11
3.5	Iterative single scale benchmarking	12
3.6	Multi scale Benchmarking	12
4	Results	14
4.0.1	Performance based on throughput	15
4.0.2	Performance based on Response time	18
4.0.3	Performance based on Latency	19
4.0.4	Result from Multiscale benchmarking test	19
5	Conclusions	20

1 Introduction

The use of geospatial data is increasing in many applications, including traffic management, ride-hailing services, and food sector, etc. The volume of geospatial data is predicted to increase by 20% every year[9]. The increase in volume of data has resulted in ever-growing number of big data platforms that could support spatial analysis. If we define geospatial data in simple words, it is a description of object in reference to its position on earth surface. We generally measure the position of an object using Latitude and Longitude. Angular measures can then be expressed in digital degrees or in degrees, minutes and seconds. We all know that earth is sphere or ellipsoid but is generally represented on 2d planar surface. This process of representing curved surfaces into a plane is known as projection. Till date, there is no mathematical method developed that can modify curved surface into planar surface without distortion. So, projections mostly modify data and include some deformation about lengths, areas or shapes. So, now we have original geographic coordinate system that takes longitude and latitude to predict position on earth as well as we have projected coordinate system that demonstrates Earth's spherical surface into two dimensional Cartesian coordinate plane.

A spatial reference system identifier is an unique code that aids to identify spatial reference system. Spatial reference system is a file containing various parameters about projection ellipsoid and datum. We will discuss about it in more detail later in the paper. So, now we know how we can represent a position of any object on earth's surface but now the other factor is how can we represent geometrical shapes on earth surface. There are mainly two approaches to deal with spatial database when representing a real object i.e. vector data and raster data. Vector data takes a set of discrete locations as input and builds geometrical shapes such as lines and polygons. For example, town can be represented as point if you draw map of world with country capitals shown. On the other hand, raster data used a regular tessellation, defining cells where one or more values are uniform. The shape of cells are square even if it is not a constraint. For example, we can use raster data to build elevation level of earth's surface where each cell can store height over sea level in meters. Symbol enables you to add information to the feature on the map. For example, colours can be used to show humidity level. In this paper, we will mainly discuss about raster data. Raster data are generally used to represent values which are continuously changing in space.

We are in the era of big raster data. We daily produce millions of satellite images and petabytes of data yearly. The volume of data is growing rapidly hence, it becomes crucial to select the right big data platform for respective data size. When selecting the right platform, many factors come into play such as speed, accuracy, interface, and consistency. So, my thesis actually focuses to compare two already present big data platforms i.e. geoserver and rasdaman and compare their efficiency using various benchmarking tools.

1.1 Geoserver

Geoserver is an open source software, purely made in Java to support storage and analysis of geospatial data. The geoserver could work with both vector and raster data. Geoserver could support a lot of input formats such as shp file, tiff file, data from database, POSTGIS, multiple formats like Image mosaic, Arcgrid and from services (WCS, WMS, WFS) and provides OGC standard output like WCS, WMS WFS e.t.c. Geoserver also

eases visualization of big geospatial data on platforms like Leaflet/Mapbox/Openlayer. GeoServer is the reference implementation of the Open Geospatial Consortium (OGC) Web Feature Service (WFS) and Web Coverage Service (WCS) standards, as well as a high performance certified compliant Web Map Service (WMS).[7]

Geoserver is also a kind of monolithic application in which data application and access code are combined into single program as single platform. . Geoserver is currently developed under spring framework of Java. It also uses user based framework called wicket so, that extension and modules of geoserver are distinct. REST API is generally based on Model View Controller(MVC). The core application that processes geospatial data is known as dispatcher. The dispatcher generally takes request and then dispatches it to appropriate service like WMS, WCS e.t.c. and then they process the request and provide necessary output. Geoserver has extensions popularly known as plugins through which geoserver adds new features or connect with other application.

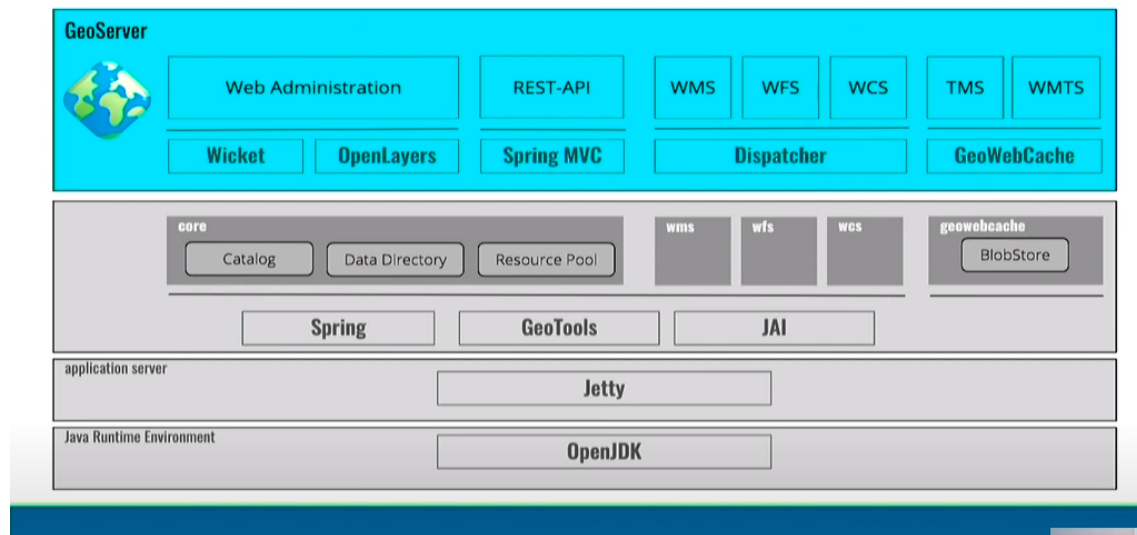


Figure 1: Geoserver architecture

In Geoserver, we have to create workspace in order to work with geospatial data. Workspace can be defined as container which helps to organise other items. Workspace is generally used to organise actions on similar geospatial data. The interface helps to easily add and delete workspaces. Under the workspace, we can connect to stores which contains vector or raster data. A data source could be coming from various different sources. As we upload data from data source, geoserver automatically converts those data into homogeneous layers which can be then previewed and several spatial operations could be applied. People added to the given workspace can view the files of just that particular workspace. In this way, different projects can be maintained and privacy is also maintained. Hence, geoserver maintains a level of data heirarchy which in turn aids to improve privacy.

1.2 Rasdaman

Rasdaman also known as Raster Data Manager is an domain independent Array DBMS which makes it suitable for all application where raster data management is an issue.[1]

Array DBMS provides database services specifically for arrays and raster data. It refers that it can store data items (pixels, vovs..) which are generally stored in a grid of 2,3 or more dimensions. Generally arrays are stored in table with an unique OID (Object Identifier) to identify that particular array. Therefore, there is possibility of foreign referencing between the regular arrays and the regular relational tuples. Then queries are run to perform various analysis on these tables. Queries have similar statement but instead, it accepts array expression which enables it to perform even further complex computations. Such a model largely reduces computation time for raster data. The bigger advantage is that it also enables rasdaman to work with multidimensional data. It can work with 2d raster graphics data, 3d time series data, 4d ocean related and climate related data. Such a powerful query is then passed into rasdaman server. At rasdaman server, it accepts these query strings and generates a tree known as operation tree. Also, it takes tree as argument and applies algebraic optimisation rules to improve processing. So, out of 150 algebraic rules, 110 of them are attempting to optimise while the work of other 40 is to transform the query into standard state of an attribute also known as canonical form. Hence, parsing as well as optimisation altogether takes less than a millisecond.

A java servlet, also known as petascope is actively running on rasdaman which enables it to offer Web services interface. The interface provides interactive user interface for geospatial data. It also accepts data from various inputs such as external files or the database. It accepts raster data coming from various sources and then convert them into various coverage. Then, simple query could be run to perform geospatial operation and provide output in form of WCS,WMS,WCPs and WPS. Rasdaman is the reference implementation for WCS and WCPs services.

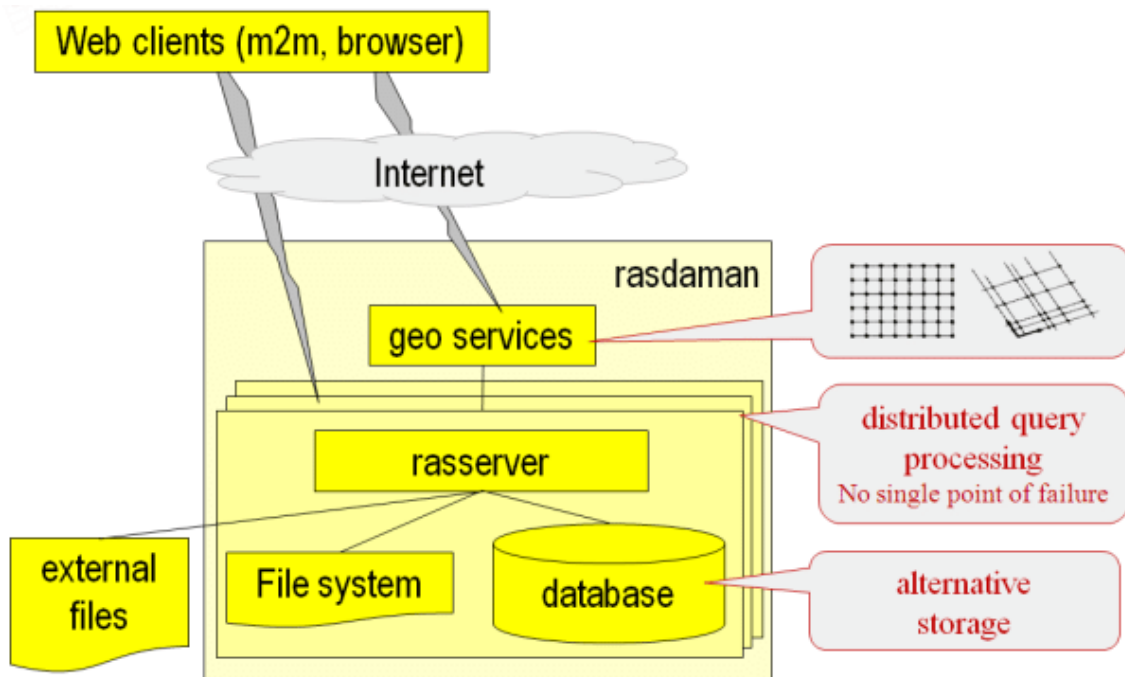


Figure 2: Rasdaman architecture

1.3 Geospatial Operations

So, there a lot of geospatial data coming from a lot of instances i.e. it could be coming from natural observations or it could also be simulation. These data could also be

coming from sensor hardware. The upstream service converts these various data into homogeneous coverages for raster data and feature types for vector data. These data is then available as layers for computation and hence provide output in various services

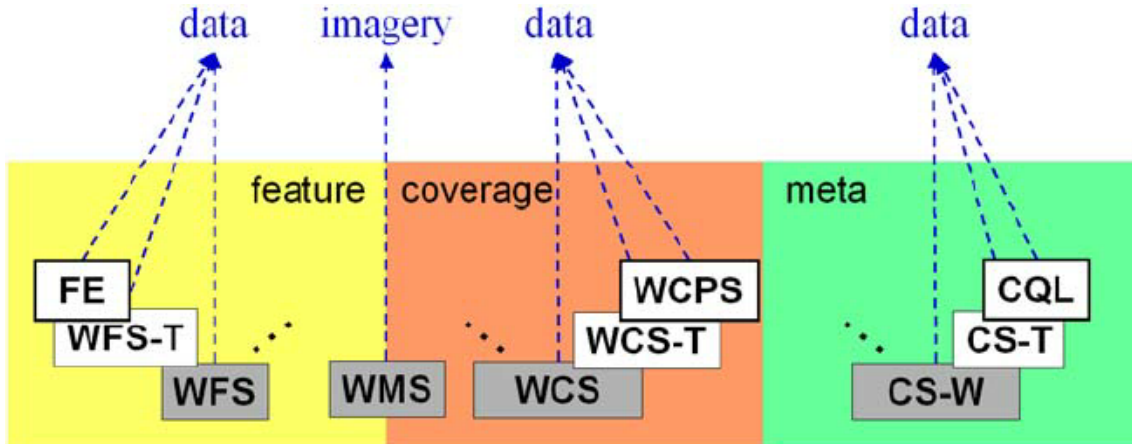


Figure 3: OGC standard web services

OGC provides web based tool for accessing and providing analysis on these data. Web Feature Service is developed to get the features related to the given coverage. The GetFeatureInfo interface by WFS provides a way to obtain attribute information about geographic features displayed in a map.[10] The output of this service provides data which could be used again for further processing. On the other hand, WCS or Web Coverage Service helps to extract the actual data without any computation. The GetCoverage returns a coverage in a well known format.[2] It also provides data that could be used for further processing. WMS or Web Map Service is between WFS and WCS. WMS takes coverage and outputs an image in different formats available for human interaction. GetMap retrieves a map image for specific area and content. [4]. Generally, in this paper we will be mainly discussing on WMS requests as they are supported by both Geoserver and Rasdaman.

2 Related field of work

Recently, there have been a lot of work related to benchmarking various kind of geospatial data. Regarding vector data, S. Ray, B. Simion, Brown, and A.D. Jackpine developed a benchmark named as "Jackpine" for vector data.[8] Jackpine implemented nine spatial analysis operations and five topological relationships on three vector data types: point, lines, and polygons. Jackpine implemented benchmarking for array databases but only tested data loading and array subsetting operators. For raster data,[6] David Haynes, Philip Mitchell and Eric Shook run benchmarking test on different big data platforms. They performed various local operations on 3 datasets (GLC, MERIS, NLCD) on big data platforms such as POSTGIS, Scidb, and GeoTrellis. SciDB and GeoTrellis, exhibited superior performance methods on the local operations as PostGIS was unable to finish the computation for the raster add operator as the data volume grew. GeoTrellis was also the superior platform when performing focal operations. For raster data benchmarking, FOSS4G has been prominent in running benchmark tests for geoserver and rasdaman and publishing the results for geospatial community. In May 2013, they performed raster

data benchmark for geoserver and mapserver [3]. Their result was accepted by a lot of other benchmarkers and referenced by platforms like geoserver. They used Jmeter to benchmark large raster data under browser environment. They find out that considering speed, mapserver performed better than geoserver with some exceptions. But in relation to user interface, geoserver was at upper hand than mapserver as mapserver is a console based tool. In this paper, we are using the same benchmarking tool as used by [3] i.e. Jmeter and would then take a step further to analyse performance by tweaking various parameters.

3 Methodology

This section defines most importantly the experimental setup and execution of the benchmarking process. The first section talks about data and hardware setup and then the second section explains the environment to set up for benchmark, benchmarking tools finally leading to benchmarking tests and then results.

3.1 Data and Hardware specification

First of all, we selected the dimension of data. Since maximum dimension supported by geoserver is 2D data while rasdaman could support data of multidimensions. So, we selected two dimensional geotiff file as a data format. According to the hardware specification, we selected the data size of 3GB since, the memory size of the device was 8GB. We want to make sure that memory size will not have effect on processing of request or receiving response. Also, we decided to select data of irregular time interval for this benchmarking test. So, there was data available on rasdaman domain related to Average Chlorophyll from year 07.2002 to 05.2015 that fulfills all the requirement and hence, we use the same file in this benchmarking test. AverageChlorophyll is a zip file whose total size is around 3000MB. Inside zip file, there are different year wise files arranged from 07.2002 to 05.2015 and files arranged in order of increasing month. Each file represent the month of that respective year and the spatial extent of data was (-180,-90,180,90). Each month file is of size 19.5 MB.

Property	Value
File format	geotiff
Datasetname	AverageChlorophyll
Geospatial extent	(-180.-90.180.90)
Pixel width	3600 pixel
Pixel height	1800 pixel
Time series	irregular

Table 1: Dataset specification

We used virtual machine to conduct benchmarking test. The device on which virtual machine were installed consisted of 8 cores and 16 threads. The memory size was 8GB. The operating system used was Linux Ubuntu 20.04.4 since rasdaman is supported by linux operating system and geoserver in both linux and windows operating system, our ideal choice to perform benchmark was linux operating system. To make sure that other running processes do not interfere with the benchmarking performance, we used Virtual machine to setup benchmark tests.

3.2 Creating Coverages

This would be our first step after setting up the device and dataset, we need to upload the geotiff files as coverages in rasdaman and geoserver. There were two tasks i.e. first was to upload coverages and next to combine coverages. The process is really different in geoserver and rasdaman. We combined monthwise data i.e. there was one tiff file made by combining all 2002 files similarly one tiff file by combining all tiff files from 2002-3 and so on making final file combining all tiff files from 2002-15. The lowest datasize was 116.64MB and maximum datasize was 3149.28MB. So, finally there were 14 coverage files made that were ready to be tested by rasdaman and geoserver.

3.2.1 Creating Coverage in Geoserver

In Geoserver, we need to create workspace at first, once we create workspace, we need to create a store. Once you click a store, you get to choose which data you need to select. You can select a geotiff file, give a name for the layer, and a little bit information about data and upload the tiff file. The geotiff file could be uploaded from many different sources. Then the uploaded file could be viewed as layer and you can publish it. You can click on Edit Layers to edit layer names and abstract. Moreover, there are some dependent variable that can be calculated from the uploaded data such as bounding boxes. Then we can view layers in Openmap. To combine tiff files in geoserver, there are two options: i) using an image mosaic plugin and ii) using layer groups. To use image mosaic, you can just select for image mosaic instead of geotiff at the time of adding stores. The other option is after adding layers, we can group layers and add different layers, give a title to the grouped layer and compute bounds from the layered group.

New Workspace

Configure a new workspace

Basic Info **Security**

Name

Namespace URI

The namespace uri associated with this workspace

☐ Default Workspace

☐ Isolated Workspace

Save **Cancel**

New data source

Choose the type of data source you wish to configure

Vector Data Sources

- Directory of spatial files (shapefiles) - Takes a directory of shapefiles and exposes it as a data source
- GeoPackage - GeoPackage
- PostGIS - PostGIS Database
- PostGIS (JNDI) - PostGIS Database (JNDI)
- Properties - Allows access to Java Property files containing Feature information
- Shapefile - ESRI(tm) Shapefiles (*.shp)
- Web Feature Server (NG) - Provides access to the Features published a Web Feature Service, s

Raster Data Sources

- ArcGrid - ARC/INFO ASCII GRID Coverage Format
- GeoPackage (mosaic) - GeoPackage mosaic plugin
- GeoTIFF - Tagged Image File Format with Geographic information
- ImageMosaic - Image mosaicking plugin
- WorldImage - A raster file accompanied by a spatial data file

Other Data Sources

- WMS - Cascades a remote Web Map Service
- WMTS - Cascades a remote Web Map Tile Service

Edit Layer

Edit layer data and publishing

benchmark:COLOR_CHLORA_2002-07

Configure the resource and publishing information for the current layer

Data **Publishing** **Dimensions** **Tile Caching** **Security**

Edit Layer

Basic Resource Info

Name
COLOR_CHLORA_2002-07

☒ Enabled

☒ Advertised

Title
COLOR_CHLORA_2002-07

Abstract

Keywords

Current Keywords
COLOR_CHLORA_2002-07
WCS
GeoTIFF

New Keyword

Vocabulary

Add Keyword

Metadata links

No metadata links so far

Add link Note only FGDC and TC211 metadata links show up in WMS 1.1.1 capabilities

Layer Groups

Define and manage layer groupings

- Add new layer group
- Remove selected layer group(s)

<< < 1 > >> Results 1 to 14 (out of 14 items)

<input type="checkbox"/> Layer Group
<input type="checkbox"/> serve_2015
<input type="checkbox"/> serve_2014
<input type="checkbox"/> serve_2013
<input type="checkbox"/> serve_2012
<input type="checkbox"/> serve_2011
<input type="checkbox"/> serve_2010
<input type="checkbox"/> serve_2009
<input type="checkbox"/> serve_2008
<input type="checkbox"/> serve_2003
<input type="checkbox"/> test_server_2002
<input type="checkbox"/> serve_2007
<input type="checkbox"/> serve_2006
<input type="checkbox"/> serve_2005
<input type="checkbox"/> serve_2004

Figure 4: Creating Coverage in Geoserver

As quoted in [3] imagemosaic plugin didn't work as expected, we also reach to the same conclusion and would recommend to use layer groups for merging geotiff files.

3.2.2 Creating Coverage in Rasdaman

In rasdaman, we generally use a special file ingredient.json where we define parameters how tiff files will be uploaded and combined. A typical ingredient.json file looks like this:

3.3 Benchmarking tools

After having the data in respective database, now we are more keen to decide which benchmarking tools we use to benchmark the performance. In this literature, we use two popular tools for benchmarking known as Jmeter and hyperfine. Jmeter is developed by Apache Software Foundation(ASF) which is mostly used for performance and load testing.[5] It is a platform independent software and can be used in any operating system and is used in unit testing for many applications. It is also used by many benchmarkers to benchmark geospatial platforms. While hyperfine is a command line based tool which is used for quick benchmark and will return response time after sending the request for several times. We can pass various parameters to hyperfine as argument and use it to investigate effects of different factor such as caching,number of runs on benchmark test

3.4 Setting up Benchmarking environment

Now, the next step is to set up benchmarking environment in Jmeter.First of all, we configure Jmeter and create a Testplan. After creating testplan, we define number of threads. Threads define how many consecutive requests are made at each time. We define it to 4 and ramp up second to 60 seconds. This ensures that the time would completely depend on speed of server processing the request. The ramp up period defines at what time a new thread request is added. For example here 60 seconds for 4 requests refers that in every 15 second, new thread request will be added i.e. in 30 seconds, 2 thread request sent at a time, in 45 seconds,3 thread requests will be sent at a time and so on. Then we go on and add a loop controller. Loop controller controls how a loop should run.After we add an http request which has 3 parameters:

Field Name	Value
Servename or IP	localhost
Port	8080
Path	rasdaman/ows

Table 2: a sample HTTP request parameter setup in Jmeter

This is a table showing sample HTTP request setup in Jmeter. Servename is basically setup to localhost. The port must be defined in which geoserver or rasdaman is setup. Since, both of them are by default setup to port 8080, one of their port must be changed. Finally path is generally rasdaman/ows for rasdaman and geoserver/ows for geoserver. We also setup method to 'GET' since we try to send the query and read the response.Now since, we setup standard template for benchmarking test, we need to go a step further and set up specific parameters for specific request such as WMS and WCS.

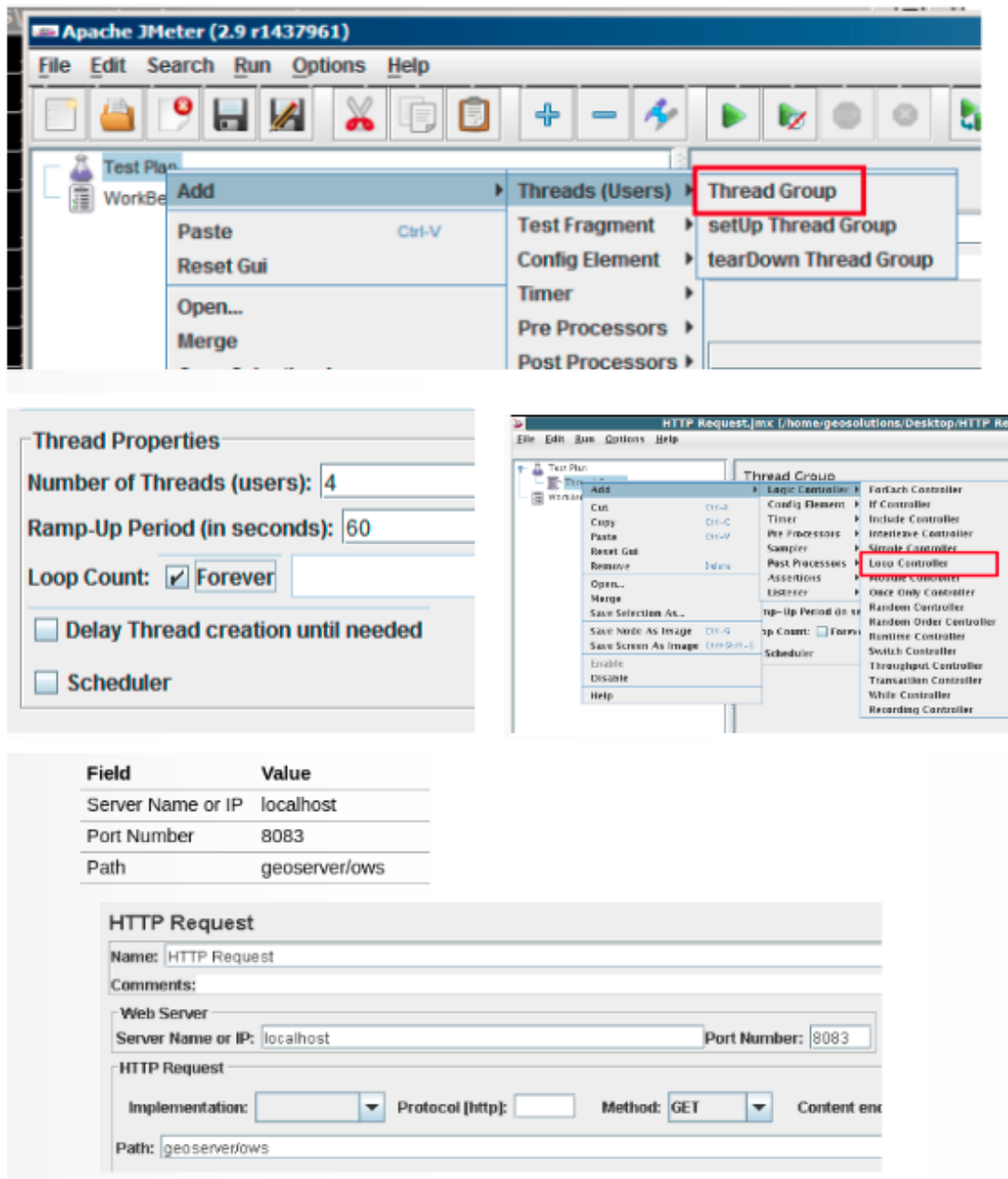


Figure 6: Configuring Testplan in Jmeter

3.4.1 Setting up parameters for WMS request in Jmeter

Now we need to select parameters for sending WMS request. The Jmeter will then take parameters as arguments and outputs a query that can be sent as a request to the server. The typical parameter file for WMS request looks something like this:

Parameters Body Data Files Upload		
Send Parameters With the Request:		
Name:	Value	URL Encode?
request	GetMap	<input type="checkbox"/>
service	WMS	<input type="checkbox"/>
version	1.3.0	<input type="checkbox"/>
layers	benchmark:serve_2015	<input type="checkbox"/>
styles		<input type="checkbox"/>
srs	EPSG:4326	<input type="checkbox"/>
width	750	<input type="checkbox"/>
height	500	<input type="checkbox"/>
format	image/png	<input type="checkbox"/>
bbox	-180,-90,180,90	<input type="checkbox"/>

Figure 7: Parameters for WMS request

The first parameter defines the type of request. In this case we use GetMap functionality so we define request as GetMap and service as WMS. Then we define the WMS version i.e. the version that has been configured in your system e.g. 1.1.0, 1.1.1, 1.3.0 e.t.c. Then in layer section, we need to define layer name to which response must be generated. If style is empty, then default styling is selected. The srs points to specific SRID(Spatial Reference) which contains information about ellipsoid. The error in SRID could result in wrong value in outputs. Then we keep width and height to be fixed and there are different formats in which output can be generated such as image/png, image/gif, image/tiff e.t.c. and bbox also known as bounding box defines the dimension for the box that comes as a result. We change layer name by keeping all the other parameter constant so, that request size is constant. Now we can see in request header link like this:

```
GET http://0.0.0.0:4040/geoserver/ows?request=GetMap&
service=WMS&version=1.3.0&layers=benchmark:serve_2015&
styles=&srs=EPSG:4326&width=750&height=500
&format=image/png&bbox=-180,-90,180,90
```

3.4.2 Setting up benchmark in hyperfine

Now we can take the above link and put it as argument in hyperfine. Then we enter the command and hyperfine would output the result by sending a fixed number of request and measuring response time and averaging response time. The hyperfine would take different parameters with the argument. You can use -w for warm cache up to perform a number of programs before the actual benchmark.

```
hyperfine --warmup 3 'GET ...'
```

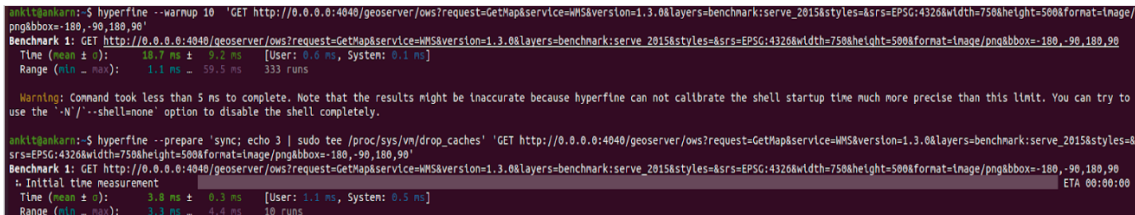
Also we can use -prepare parameter for cold cache. We would write something like:

```
hyperfine --prepare 'sync; echo 3 | sudo tee /proc/sys/vm/drop_caches' 'GET..'
```

So, we can investigate the effect of caching on benchmarking tests. We could also configure how many times a request can be made using this command:

```
hyperfine --runs 500 'GET...'
```

So, this means that 500 requests are made and response are measured and averaged. The sample request in hyperfine would look something like this:



```
ankittgankar@:~$ hyperfine --warmup 10 'GET http://0.0.0.0:4040/geoserver/ows?request=GetMap&service=WMS&version=1.3.0&layers=benchmark:serve_2015&styles=&srs=EPSG:4326&width=750&height=500&format=Image/png&bbox=-180,-90,180,90'
Benchmark 1: GET http://0.0.0.0:4040/geoserver/ows?request=GetMap&service=WMS&version=1.3.0&layers=benchmark:serve_2015&styles=&srs=EPSG:4326&width=750&height=500&format=Image/png&bbox=-180,-90,180,90
Time (mean ± σ): 10.7 ms ± 9.2 ms [User: 0.0 ms, System: 0.1 ms]
Range (min ~ max): 1.1 ms ~ 59.5 ms 333 runs

Warning: Command took less than 5 ms to complete. Note that the results might be inaccurate because hyperfine can not calibrate the shell startup time much more precise than this limit. You can try to use the '-N'/'--shell=none' option to disable the shell completely.

ankittgankar@:~$ hyperfine --prepare 'sync; echo 3 | sudo tee /proc/sys/vm/drop_caches' 'GET http://0.0.0.0:4040/geoserver/ows?request=GetMap&service=WMS&version=1.3.0&layers=benchmark:serve_2015&styles=&srs=EPSG:4326&width=750&height=500&format=Image/png&bbox=-180,-90,180,90'
Benchmark 1: GET http://0.0.0.0:4040/geoserver/ows?request=GetMap&service=WMS&version=1.3.0&layers=benchmark:serve_2015&styles=&srs=EPSG:4326&width=750&height=500&format=Image/png&bbox=-180,-90,180,90
Initial time measurement
Time (mean ± σ): 3.0 ms ± 0.3 ms [User: 1.1 ms, System: 0.3 ms]
Range (min ~ max): 0.3 ms ~ 4.4 ms 10 runs
ETA 00:00:00
```

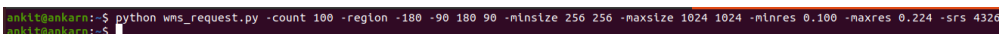
Figure 8: Sample hyperfine request

3.5 Iterative single scale benchmarking

After the import process is done, we go on to execute the benchmark. The query is run in Jmeter for 60 seconds since our ramp up time is 60 seconds. So, as we know we have already divided data into different data sizes yearwise and uploaded to rasdaman and geoserver as coverages(layer). Now we keep all the parameters constant and change the layer name to let them point to different layers and then run the test. It tries to send requests for 60 seconds and record throughput. Throughput is amount of requests sent per unit time(second). Time is calculated from start of first request to the end of sending the last request. It also includes interval between samples as it also represents the load on the server. We also measure Latency i.e. time from which first request has been sent and then response has been recorded.

3.6 Multi scale Benchmarking

So, we have successfully setup benchmarking test for varying data sizes. But we also want to investigate what would be benchmarking result if we vary other parameters by keeping datasize constant. For this experiment, we choose smallest size i.e. 116.34MB as ideal size. The parameters that we are gonna vary is bounding box(bbox), width, and height. So, I wrote a python file that generates specified number of different values for bbox, width and height within given range. We need to run command something like this:



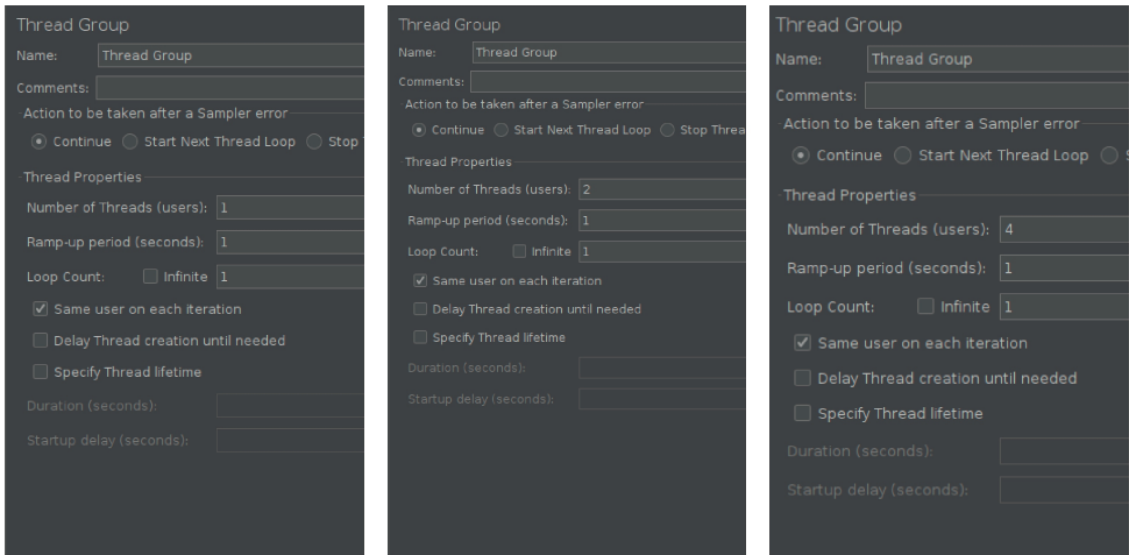
```
ankittgankar@:~$ python wms_request.py -count 100 -region -180 -90 180 90 -minsize 256 256 -maxsize 1024 1024 -minres 0.100 -maxres 0.224 -srs 4326
ankittgankar@:~$
```

As you can see count parameter defines how many different values must be created. For this experiment, we use 100 different values for bbox, width and height, region defines the value for the bounding box. You can almost define the same minsize and maxsize for the image. The minres defines the minimum pixel size and maxres defines the maximum pixel size. You should also be aware that minres and maxres does not accept negative

values and can be defined by looking at image property. These parameters are generally defined from the property of input file. We can alternatively also type:

```
gdalinfo filename
```

to see related upper and lower bounds for tile size, resolution and bounding boxes of respective file. Now we go to Jmeter add a TestPlan. We add three new thread groups known as 1, 2 and 4. Then for each thread group number of thread = name of group ramp up period and loop count equals 1. It looks something like this:



In TestPlan section, click on Run thread groups consecutively. Add a loop controller to the first thread group and then add http request In http request add the following fields as parameters:

Name:	Value
bbox	\${bbox}
height	\${height}
width	\${width}

Go to the bottom and uncheck Follow redirects and Use Keepalive options. Then go to CSV Data Set Config element and configure the following fields.



The filename should contain the name of file you have generated. The variable names must be: width, height and box and Delimiter must be ';'. Then we copy the loop controller and click on the other thread group and paste the configuration. So, the loop controller

must be configured in this format:
Thread group 1: 100
thread group 2: 50
thread group 3: 50
So, the number of requests are also constant i.e. 400 requests. Now we add HTTP request Defaults and we enter following basics configuration for port number, path and http request as entered above:

Name	Value	URL Encode?	Content-Type	Include Equals?
request	GetMap	<input type="checkbox"/>	text/plain	<input checked="" type="checkbox"/>
service	WMS	<input type="checkbox"/>	text/plain	<input checked="" type="checkbox"/>
version	1.3.0	<input type="checkbox"/>	text/plain	<input checked="" type="checkbox"/>
layers	AverageChloroColor2002_15	<input type="checkbox"/>	text/plain	<input checked="" type="checkbox"/>
styles		<input type="checkbox"/>	text/plain	<input checked="" type="checkbox"/>
crs	EPSG:4326	<input type="checkbox"/>	text/plain	<input checked="" type="checkbox"/>
format	image/png	<input type="checkbox"/>	text/plain	<input checked="" type="checkbox"/>

Now for parameters, we define all the other parameter except the width, height and bbox. It should look something like above. Add listeners as we added in the single test. Lastly we need to add Response assertion where we need to fill fields

Apply to	Main sample only
Response field to test	Response Headers
Pattern Matching Rules	Contains
Pattern to test	Content-Type: image/png

Table 3: Request assertion table

It should look something like this:

Now save the Jmeter file and run the test.

4 Results

So now we will move on to results we obtained from our benchmark tests. Here is the result we obtained when we differ the file size by keeping all other parameters constant. The result could be plotted in a table like this:

4.0.1 Performance based on throughput

Datasize(in MB)	Throughput _{ras} (/s)	Throughput _{geo} (/s)	Deviation _{ras} (%)	Deviation _{geo} (%)
116.64	1912.35079	13.03131	0.003	0.508
349.92	2178.24318	4.77216	0.001	1.389
583.2	2099.43214	2.92715	0.003	2.247
816.48	2044.58916	2.06495	0.001	3.125
1049.76	2009.15283	1.50962	0.002	4.396
1283.04	1988.34237	1.17044	0	5.634
1516.32	1914.19017	1.04526	0.002	6.349
1749.6	2044.28069	1.00484	0.002	6.557
1982.88	1781.84415	0.91414	0.003	7.273
2216.16	1814.15255	0.8335	0.002	7.843
2449.44	1777.54974	0.72357	0.003	9.091
2682.72	2100.98758	0.67199	0.003	9.756
2916	1778.11102	0.60918	0.004	10.811
3149.28	1810.63837	0.61314	0.002	10.811

Table 4: Result from benchmark tests

Here is the table explaining results from benchmark test. The first column explain datasize in MB, the second one is the throughput observed in rasdaman for given datasize, third one is throughput observed in geoserver, then deviation in rasdaman and last column explains deviation in geoserver. So, as we see the throughput is very high in rasdaman as compared to geoserver irrespective of datasize. We see that throughput is very consistent in rasdaman with an average of 1946 requests per second. While in geoserver, the throughput is fairly okay for small datasize but as we increase datasize the throughput in geoserver decreases and at large datasize like 3149MB, it is barely processing a request in a second. The average throughput of geoserver throughout the experiment remains 2.27 requests per second. So, on average for increasing datasize rasdaman was 854 times faster than geoserver for processing WMS requests. Then, we also look at deviation in rasdaman and geoserver. The deviation in values is lesser in rasdaman than geoserver. On average rasdaman deviation was 0.002 while geoserver deviation was 6.12 which refers that if the experiment is conducted again, the rasdaman would be giving the same result while the values for geoserver could deviate a little bit from the acquired values. There was a similar trend seen in deviation values as it was seen in throughput values i.e. the deviation increases as we increase the datasize. We will discuss about the trend in more detail once we look at the graph:

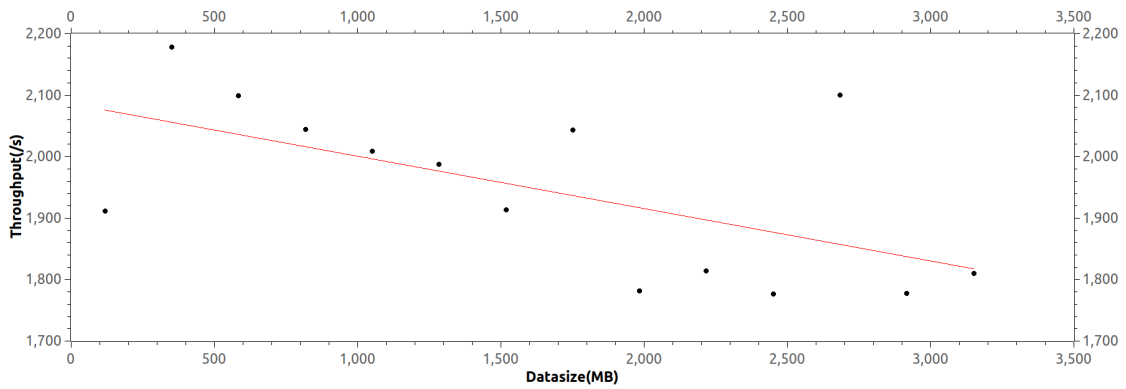


Figure 9: Throughput of rasdaman

This graph shows that there is a linear correlation between datasize and rasdaman inputs. As datasize increase, the throughput i.e. number of requests made decreases. Hence, we can assure that rate of change is constant and small i.e. there is not much fluctuations in value of throughput as we increase data size. Also, we can see the range of y axis between 1700 and 2200 requests per second.

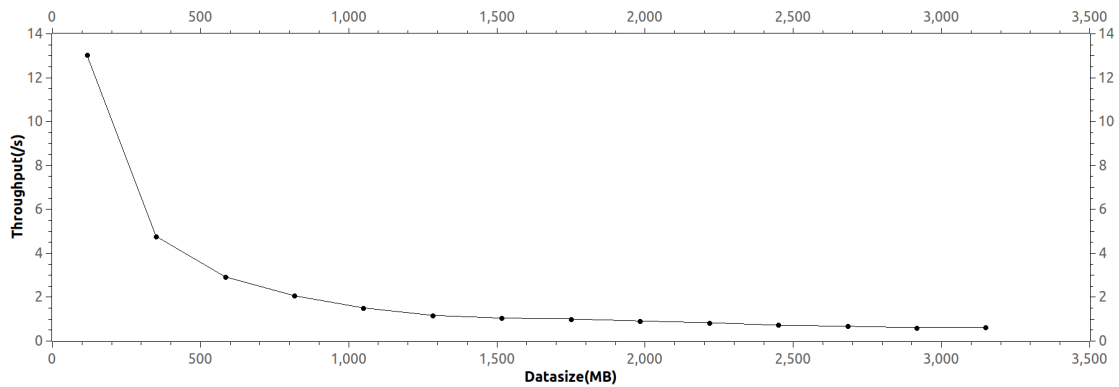


Figure 10: Throughput of geoserver

Meanwhile in geoserver also, we can see that as datasize increases, the value of throughput decreases and gets constant to somewhere near 1. Rate of change is not constant and we can see a greater change in rate in comparison to rasdaman i.e. greater fluctuation on increasing datasize at first and then becoming constant i.e. reaching its optimum level. The range of y axis is between 0 to 14 which is way lesser than that of rasdaman. Now we try to combine both graphs and see the actual change

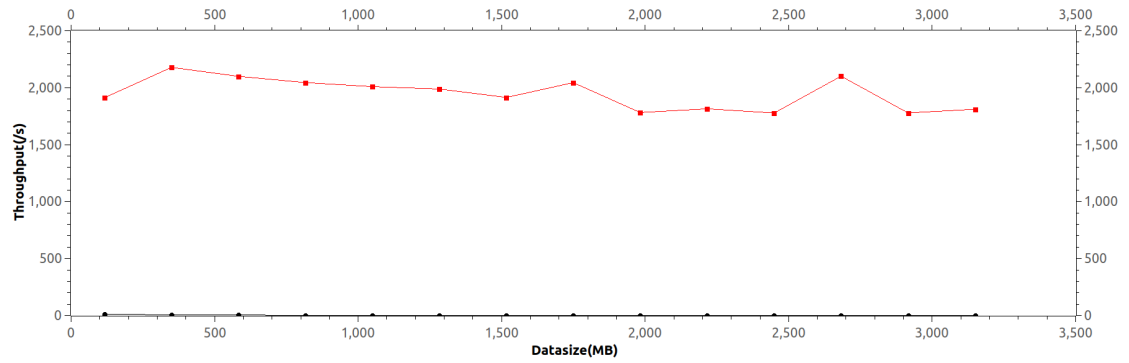


Figure 11: Throughput of geoserver and rasdaman

Here we can see that geoserver with its throughput occupies lower y values of first quadrant while rasdaman occupies higher y values. It is clearly seen that rasdaman perform better than geoserver and there is no point where geoserver and rasdaman performance is comparable. In throughput, rasdaman has clearly outperformed geoserver i.e. rasdaman can process more requests per unit time as compared to geoserver.

4.0.2 Performance based on Response time

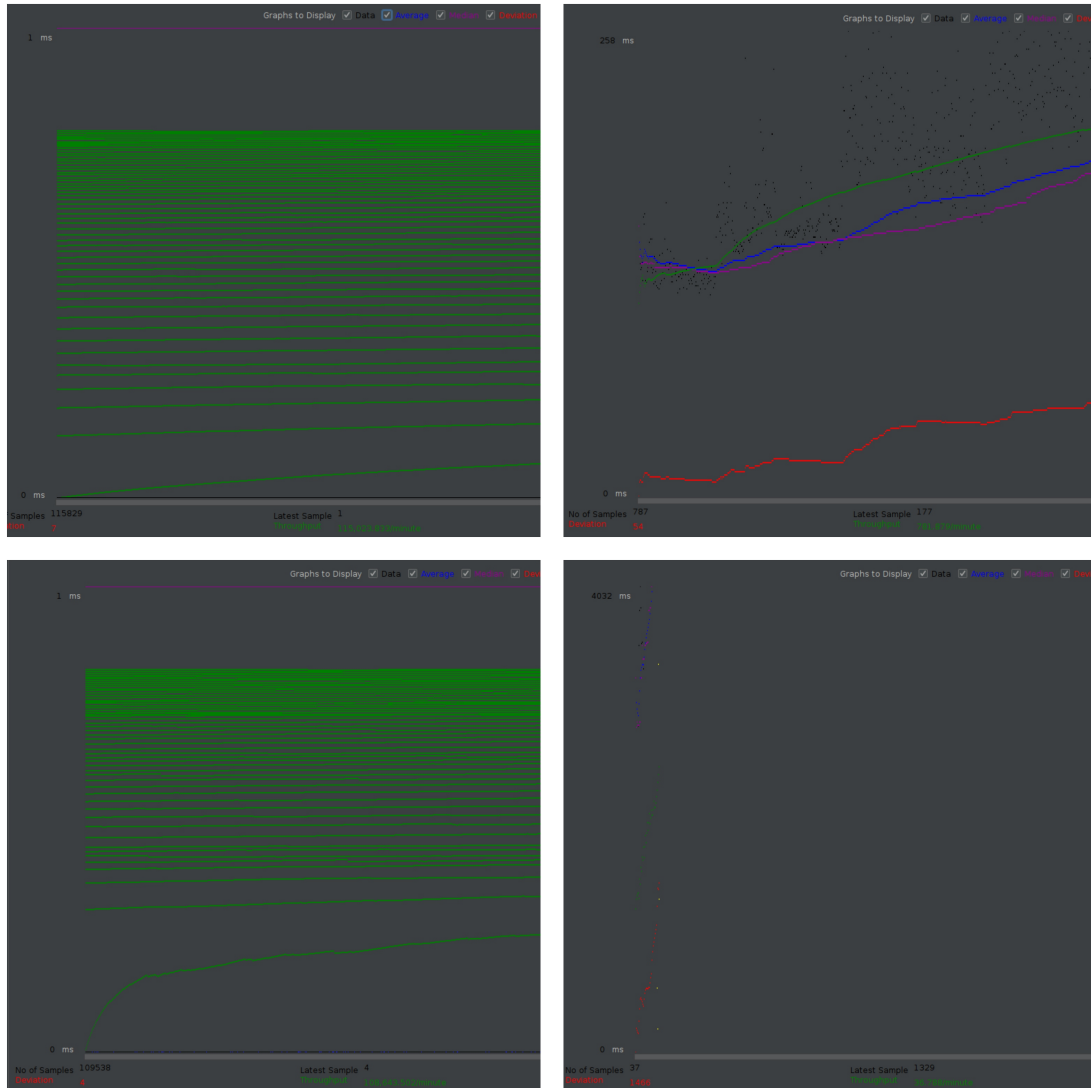


Figure 12: Request time graph for geoserver and rasdaman

So, as we see on the left side, we can see request time graph for rasdaman. On top, it is for smallest data(116.64MB) and bottom is for largest data (3000MB) and similarly for geoserver. First, of all Jmeter has a limit of 2000 requests and as more than 2000 requests are done, it resets all the values and a new line is seen. That is the reason we can see multiple lines in rasdaman as it has sent way more than 2000 requests while even for the smallest data, geoserver could not send 2000 requests in 1 second. While we can see here too rasdaman is consistent with its performance as size increases. For the biggest data size the geoserver is struggling to process even 1 request in 4000 ms. So, as data size increases efficiency of geoserver decreases while rasdaman is consistent with its efficiency. The red line shows the deviation from the actual throughput values. In geoserver, you can see that as execution time increases, the deviation also increases while there is no fluctuation seen in rasdaman.

4.0.3 Performance based on Latency

Datasize(in MB)	LatencyGeoserver(ms)	LatencyRasdaman(ms)
116.34	123	1
3000	11301	28

Table 5: Result from benchmark tests

As we can see for smaller data like 116.34MB, latency is around 123 ms for geoserver while 1ms for rasdaman and for bigger data latency of rasdaman is 403.6 times better than latency of geoserver. Latency actually measures the efficiency of processing the request and as the result shows that efficiency of processing request in rasdaman server is way higher than geoserver. The good latency speed ensures that the server processes the data faster. So, we can say that given the same data size and environment, rasdaman will generate response faster than geoserver. Finally, as we discussed caching will always have an impact on benchmarking. We want to know whether this impact will effect our data and up to what extent. So, we performed both warm and cold cache on geoserver yearwise layers using both hyperfine and Jmeter and analyzed the result.

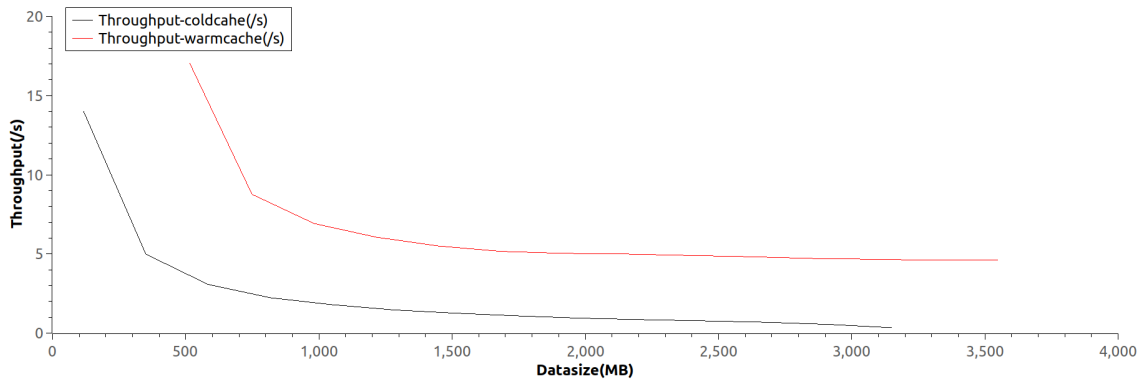


Figure 13: Effect of caching on benchmarking tests

As we can see both the graph follows the same trend. We generally try to model our benchmark test for warm cache because we need to simulate environment similar to one in browser. The cold cache, where cache is deleted at every iteration, however is same graph little bit lower cause it takes more time to process the request. We are more interested in finding the difference and whether the difference is significant for values. As we deducted the values and averaged them, we got an average difference of 0.19. The difference is not significant compared to the difference we have in the reading of rasdaman and geoserver. Hence, we conclude that caching will not have a significant impact on our benchmarking tests.

4.0.4 Result from Multiscale benchmarking test

We tried to differ values of variables width, height, and bboxes by keeping all other variable constant. So, we generated 100 different values of variables and supplied to test plan and saw the result. The result could be described as follows:

Datasize(in MB)	Throughputras(/s)	Throughputgeo(/s)	Latency_ras(ms)	Latency_geo(ms)
116.64	79.27070947	12.37738651	2	1128

Table 6: Result from multiscale benchmark tests

The trend in latency was almost the same as expected. But there was a difference in throughput. For rasdaman, throughput varied from minimum of 8 request per second to maximum 249 request per second. After running test for several times and averaging then the average for rasdaman was around 79.3 requests per second while geoserver's minimum was 10 requests per second and maximum was 15 requests per second which averaged to around 12.4 request per second. We can still see that rasdaman has a better performance value than geoserver on average but it is still comparable.

5 Conclusions

So, as we configured benchmark test for geoserver and rasdaman, we observed varying results. In an environment simulated as of browser controlling all the controlled variables, we observed that rasdaman is way efficient than geoserver for handling big geospatial raster data. We measured the result in 3 parameters, the first one was the amount of request that could be made in 1 second and rasdaman clearly outperformed geoserver. Secondly, we observed on how requests are being sent to geoserver and rasdaman and there also we observed that rasdaman was more efficient in processing request than geoserver. Finally, we also observed on the time it takes to process request and generate response and there too rasdaman was clearly more efficient.

We observed that on average for varying datasize, rasdaman was 854 times faster than geoserver for processing request and generating response and deviation for rasdaman was as low as 0.002 which refers that if the test is conducted again with the same paramters, it would precisely give the same result. If we compare latency on average, rasdaman is 393.9 times faster than geoserver in generating response.

We then went on to investigate the effect of caching on the benchmarking test. We found out that the effect is not that significant according to our results. We then conducted multiscale benchmarking by varying other variables and keeping datasize constant and we saw a comparable throughput for geoserver and rasdaman. Even in this environment, rasdaman is 6.6 times faster than geoserver. Hence, we conclude that in related to speed and large geospatial raster data, rasdaman would perform better than geoserver in relation to speed.

References

- [1] P. Baumann et al. "The Multidimensional Database System RasDaMan". In: *SIGMOD Rec.* 27.2 (June 1998), pp. 575–577. ISSN: 0163-5808. DOI: [10.1145/276305.276386](https://doi.org/10.1145/276305.276386). URL: <https://doi.org/10.1145/276305.276386>.
- [2] Peter Baumann. "Beyond rasters: introducing the new OGC web coverage service 2.0". In: *Proceedings of the 18th SIGSPATIAL International Conference on Advances in Geographic Information Systems*. 2010, pp. 320–329.
- [3] Michael Billmire. "WMS Server Benchmarking for Large Raster Formats". In: May 2013.

- [4] J.D. Blower et al. "A Web Map Service implementation for the visualization of multidimensional gridded environmental data". In: *Environmental Modelling Software* 47 (2013), pp. 218–224. ISSN: 1364-8152. DOI: <https://doi.org/10.1016/j.envsoft.2013.04.002>. URL: <https://www.sciencedirect.com/science/article/pii/S1364815213000947>.
- [5] Emily H Halili. *Apache JMeter*. Packt Publishing Birmingham, 2008.
- [6] David Haynes, Philip Mitchell, and Eric Shook. "Developing the Raster Big Data Benchmark: A Comparison of Raster Analysis on Big Data Platforms". In: *ISPRS International Journal of Geo-Information* 9 (Nov. 2020), p. 690. DOI: [10.3390/ijgi9110690](https://doi.org/10.3390/ijgi9110690).
- [7] Open Source Geospatial Foundation: <https://docs.geoserver.org/latest/en/user/introduction/>
- [8] Suprio Ray, Bogdan Simion, and Angela Demke Brown. "Jackpine: A benchmark to evaluate spatial database performance". In: *2011 IEEE 27th International Conference on Data Engineering*. 2011, pp. 1139–1150. DOI: [10.1109/ICDE.2011.5767929](https://doi.org/10.1109/ICDE.2011.5767929).
- [9] Bharti Sharma et al. "Benchmarking geospatial database on Kubernetes cluster". In: *EURASIP Journal on Advances in Signal Processing* 2021 (July 2021). DOI: [10.1186/s13634-021-00754-2](https://doi.org/10.1186/s13634-021-00754-2).
- [10] Panagiotis Vretanos et al. "Web Feature Service Implementation Specification, Version 1.1. 0." In: (2005).