Master Thesis

im Rahmen des
Universitätslehrganges „Geographical Information Science & Systems“
(UNIGIS MSc) am Interfakultären Fachbereich für GeoInformatik (Z_GIS)
der Paris Lodron-Universität Salzburg

zum Thema

„Performancetests OGC API - Features / WFS – ein Vergleich“

vorgelegt von

Björn Hinkeldey
U104411

Betreuer:
Dr. Christian Neuwirth

Zur Erlangung des Grades
„Master of Science – MSc”

Mönchengladbach, 23.03.2024

Inhaltsverzeichnis

Zusammenfassung

Im Zuge der Einführung neuer OGC-Standards für Geodatendienste steht mittlerweile mit den OGC API - Features eine Alternative zum WFS zur Verfügung. Auch wenn dieser neue Standard noch nicht vollständig verabschiedet ist, gibt es hiervon bereits zahlreiche Implementierungen, mit welchen Vektordaten bereitgestellt werden können.

Für zwei dieser verfügbaren FOSS-Implementierungen, OGC API – Features auf Basis von GeoServer und QGIS Server, soll die Performance mit derer von WFS verglichen werden.

Hierbei wurde eine Methode zur Performancemessung von WFS mit der Software JMeter, welche 2013 mit einer wissenschaftlichen Arbeit vorgestellt wurde, erfolgreich auf den neuen Standard übertragen.

Getestet wurde in einem Szenario mit Cloud-Servern unterschiedlicher Leistungsklassen, welche mit dem Client über das Internet verbunden waren.

Die Ergebnisse zeigen, dass für den QGIS Server die Bereitstellung mittels OGC API – Features eine deutliche Performancesteigerung bedeutet. Für den GeoServer stellt sich das aktuell noch anders dar, hier ist der WFS noch performanter. Da dort die Integration der Implementierung aber noch nicht vollständig abgeschlossen ist, kann sich dieses Ergebnis in Zukunft noch ändern.

Im Zuge der Arbeit konnte ein Fehler in der Software-Bibliothek Shapelib, welche zum Lesen und Schreiben von ESRI-Shapefiles in zahlreichen FOSS-Lösungen verwendet wird, identifiziert werden. Dieser Hinweis wurde der OpenSource-Community gemeldet und so konnte dieser Fehler, der seit mehr als 12 Jahren bestand, zeitnah behoben werden.

Verwendete Abkürzungen

API	Application Programming Interface
ATSR	Along Track Scanning Radiometer
BBOX	Bounding Box
CRS	Coordinate Reference System
EPSG	European Petroleum Survey Group Geodesy
ESA	European Space Agency
FOSS	Free and Open Source Software
FOSS4G	Free and Open Source Software for Geospatial
GIS	Geographisches Informationssystem
GLWD-2	Global Lakes and Wetlands Database Level 2
GML	Geography Markup Language
HTTP	Hypertext Transfer Protocol
JSON	JavaScript Object Notation
OAF	OGC API - Features
OGC	Open Geospatial Consortium
QoS	Quality of Service
RAM	Random Access Memory
TIGER	Topologically Integrated Geographic Encoding and Referencing system
vCPU	Virtual Central Processing Unit
W3C	World Wide Web Consortium
WCS	Web Coverage Service

WFS	Web Feature Service
WMS	Web Map Service
WPS	Web Processing Service
WWF	World Wildlife Fund
XML	Extensible Markup Language

1. Einleitung

1.1 Motivation

In Anbetracht der zunehmenden Beliebtheit von Web-APIs hat das Open Geospatial Consortium (OGC) im Jahr 2018 damit begonnen, eine Reihe von Standards zu entwickeln, welche moderne Web-Ansätze nutzen (Hobona et al. 2023). Die Webseite des OGC bietet einen Überblick über die neuen Standards, wovon einige als Nachfolger der bisherigen, als „legacy“ bezeichneten, OGC Web Service Standards (WMS, WFS, WCS, WPS usw.) zu verstehen sind.[1]

Bereits seit vielen Jahren erfolgt die Bereitstellung von Vektordaten durch den OGC-Standard Web Feature Service (WFS)[2], welcher eine Remote-Procedure-Call-over-HTTP-Architektur in Verbindung mit XML verwendet (Portele et al. 2022). Als der WFS-Standard in den späten 1990er und frühen 2000er Jahren entwickelt wurde, war dies der Stand der Technik (Portele et al. 2022).

OGC API - Features unterstützt ähnliche Funktionen, verwendet aber einen modernisierten Ansatz, welcher aktueller Web-Architektur und insbesondere der vom W3C und OGC empfohlenen Vorgehensweise für den Austausch von Daten-/Geodaten im Internet folgt (Portele et al. 2022). Auch wenn der Standard OGC API - Features keine Formate verpflichtend vorschreibt, so sind doch HTML (zur Ansicht und zum Durchsuchen der Daten mittels eines Webbrowsers) und GeoJSON (als Format zum Bereitstellen der Geodaten) empfohlen (Portele et al. 2022).

Auch wenn der Standard OGC API - Features zum Zeitpunkt der Erstellung der vorliegenden Arbeit noch nicht vollständig verabschiedet wurde (*Part 3: Filtering* und *Part 4: Create, Replace, Update and Delete* wurden bisher lediglich als Entwurf veröffentlicht)[3], so

1 OGC API Webseite https://ogcapi.ogc.org/ (zugegriffen: 25. Februar 2024)
2 https://www.ogc.org/standard/wfs/ (zugegriffen: 25. Februar 2024)
3 https://ogcapi.ogc.org/features/ (zugegriffen: 25. Februar 2024)

gibt es jedoch bereits heute zahlreiche Softwarelösungen, welche diesen implementiert haben (zumindest die verabschiedeten *Part 1: Core* und P*art 2: CRS by Reference)*.[4]

Somit stehen Datenanbieterinnen und Datenanbietern, welche Vektordaten OGC-konform bereitstellen wollen, vor der Entscheidung, ob Vektordaten über einen WFS oder per OGC API – Features angeboten werden sollen. Nach Seip (2015) ist neben der Verfügbarkeit und Konformität die Leistung (Performanz) ein wesentliches Kriterium für die Evaluierung und Beurteilung der Qualität eines Dienstes.

Aber wie kann die Performanz eines Dienstes, welcher Geodaten bereit stellt, gemessen werden? Es finden sich verschiedene Projekte und Verfahren, wie z.B. die *QGIS Server PerfSuite*[5], welche aber spezifisch für eine bestimmte Software ist oder das Projekt *MVT Benchmark*[6]*,* welches speziell zum Benchmarking für die Erstellung und Bereitstellung von MVT-Vektorkacheln aus einer PostGIS-Datenbank entwickelt wurde. Aber insgesamt gibt es, trotz der Relevanz der Thematik, recht wenig Forschung zur Performance von Geodatendiensten, wie auch schon Giuliani et al. (2013) feststellten.

Einen Ansatz zur Performancemessung von Downloaddiensten haben Giuliani et al (2013) in dem Artikel „*Testing OGC Web Feature and Coverage Service performance: Towards efficient delivery of geospatial data“* vorgestellt. Giuliani et al. (2013) erweiterten und validierten einen Ansatz, welcher von 2007 bis 2011 und 2013 im Rahmen der jährlichen FOSS4G-Konferenzen für die *FOSS4G Performance Shoot-outs*[7] verwendet wurde. Dabei sind Teams verschiedener Open Source WMS-Server-Projekte in einem freundschaftlichen Wettbewerb „gegeneinander“ angetreten, um die Performance von WMS-Servern unter gleichen Bedingungen zu testen. Hierbei gewonnenen Erkenntnisse sind als Verbesserungen zurück in die jeweiligen Projekte geflossen.[8]

Giuliani et al. (2013) kamen zu dem Ergebnis, dass der FOSS4G Ansatz erfolgreich auf die Performancemessung von verschiedenen WFS- und WCS-Implementierungen übertragen

4 https://www.ogc.org/resources/certified-products/ (zugegriffen: 25. Februar 2024) - listet OGC-zertifizierte Produkte für OGC API -Features, darüber hinaus gibt es weitere.
5 https://github.com/qgis/QGIS-Server-PerfSuite (zugegriffen: 02. März 2024)
6 https://github.com/pka/mvt-benchmark (zugegriffen: 02. März 2024)
7 https://wiki.osgeo.org/wiki/FOSS4G_Benchmark (zugegriffen: 03. März 2024)
8 https://www.slideshare.net/gatewaygeomatics.com/wms-performance-shootout-2011 (zugegriffen: 03. März 2024)

werden konnte und ihre Arbeit als Grundlage für weitere Forschung auf dem Gebiet Quality of Service (QoS) von Geodatendiensten dienen kann.

1.2 Forschungsfrage

Mit der vorliegenden Arbeit soll versucht werden,

a) mit der Methodik von Giuliani et al. (2013) Performancetests für aktuelle Implementierungen der OGC-Standards WFS und OGC API – Features anzuwenden. Hierbei soll, falls möglich, aus Gründen der Kontinuität der damals verwendete Polygon-Testdatensatz benutzt werden. Als Software sollen, wie bereits von Giuliani et al. (2013), Apache JMeter und GeoServer zur Anwendung kommen. Als weitere FOSS-Lösung, damals nicht Gegenstand der Untersuchungen, soll der QGIS Server eingesetzt werden. Die Testinfrastruktur soll mit einem Cloud Server und einem stationärem Client-PC, welche über das Internet miteinander verbunden sind, realisiert werden.

b) Aussagen über die Performances der getesteten OGC API – Features im Vergleich zu den aktuellen WFS-Implementierungen zu treffen. Sollten von Giuliani et al. (2013) verwendete Testdaten noch verfügbar sein, sollen auch die Ergebnisse der aktuellen Performancetests des WFS auf Basis von GeoServer mit den Ergebnissen von 2013 verglichen werden.

c) Untersuchung von Auswirkungen von verschiedenen Hardwarekonfigurationen auf die Performance durchzuführen. Durch die Verwendung eines Cloud-Servers besteht die Möglichkeit, die Hardware zu skalieren. Der Cloud-Server wird zuerst mit 4 vCPUS und 16 GB RAM konfiguriert. Nach Durchführung der Testreihen mit dieser Konfiguration soll untersucht werden, wie sich die doppelte Anzahl von vCPUs und die doppelte Menge an Arbeitsspeicher auswirken.

1.3 Gliederung

Nachdem in den vorherigen Abschnitten in das Thema eingeführt und die Forschungsfrage formuliert wurde, wird in Abschnitt 2 auf die Methodik dieser Arbeit eingegangen.

Abschnitt 2.1 stellt hierbei die verwendete Software zur Performancemessung und Bereitstellung der Geodatendienste auf Basis der OGC Standards WFS und OGC API – Features vor. Abschnitt 2.2 behandelt die verwendete Hardware. Nachdem somit die Testinfrastruktur vorgestellt wurde, wird im Abschnitt 2.3 näher auf die verwendeten Testdaten eingegangen, bevor Abschnitt 2.4 dem Aufbau und der Durchführung der eigentlichen Testläufe gewidmet ist.

Abschnitt 3 mit der Überschrift „Ergebnisse“ ist relativ kurz gehalten, denn die eigentlichen Ergebnisse, Log-Dateien und Reports der durchgeführten Performancetests werden aus Platzgründen und aufgrund der Art der Dateien in einem eigens eingerichteten GitHub-Repository bereitgestellt. In der vorliegenden Arbeit wird in diesem Abschnitt daher über die Durchführung der Test berichtet.

Die Diskussion der Ergebnisse folgt im Abschnitt 4. Zuerst wird in Abschnitt 4.1 eine bei den Tests mit QGIS Server aufgetretene Fehlermeldung thematisiert. In Abschnitt 4.2 werden die Ergebnisse ausgewählter Test-Setups graphisch aufbereitet gegenüber gestellt und diskutiert. Zum Abschluss werden in Abschnitt 4.3 die Ergebnisse mit Blick auf die verwendete DSL-Verbindung zwischen dem Client und Server behandelt.

Im letzten Abschnitt 5 wird ein Fazit im Hinblick auf die Forschungsfrage gezogen und ein Ausblick auf mögliche zukünftige Aktivitäten in der Folge dieser Arbeit gegeben.

2. Methodik

Giuliani et al. (2013) bieten an, dass alles im Rahmen der Arbeit entwickelte Material wie Skripte, Daten und Verfahren auf Anfrage von ihnen bezogen werden kann. Der Autor der vorliegenden Arbeit hat die Autoren von damals kontaktiert und daraufhin im Juni 2023 dankenswerterweise per E-Mail ein ZIP-Archiv erhalten. Hierbei handelt es sich um eine relativ unstrukturierte Sammlung diverser Materialien wie Word-/EXCEL-/Text-Dateien oder auch JMeter-Testplänen, die vor allem aufgrund fehlender Testdatensätze nicht als vollständig bezeichnet werden kann. In den folgenden Abschnitten wird auch auf diesen Fundus von damals eingegangen.

Trotzdem ist diese Sammlung für die vorliegende Arbeit hilfreich gewesen und hat tiefer gehende Einblicke in die damalige Vorgehensweise ermöglicht, die nur mit dem veröffentlichten Aufsatz alleine nicht möglich sind.

2.1 Software

2.1.1 JMeter

Für die Performancemessungen sowohl für die FOSS4G Benchmarks als auch für den auf WFS und WCS erweiterten Ansatz von Giuliani et al. (2013) wurde die Software JMeter[9] verwendet. Hierbei handelt es sich um eine von der Apache Software Foundation entwickelte FOSS Java-Anwendung, welche nicht explizit für Geodatendienste entwickelt wurde. JMeter kann die Performance von sowohl statischen als auch dynamischen Webanwendungen messen und diese einem Lasttest unterziehen.

9 https://jmeter.apache.org/ (zugegriffen: 03. März 2024)

JMeter ist plattformunabhängig und läuft auf den diversen gängigen Betriebssystem wie Windows, Mac OS und Linux, sofern eine kompatible Java-Umgebung zur Verfügung steht.[10]

Es ist unter der Apache License 2.0 veröffentlicht.[11]

Der Quellcode liegt auf GitHub[12] und wird aktiv weiterentwickelt, wie die folgende Übersicht der Codebeiträge zeigt:

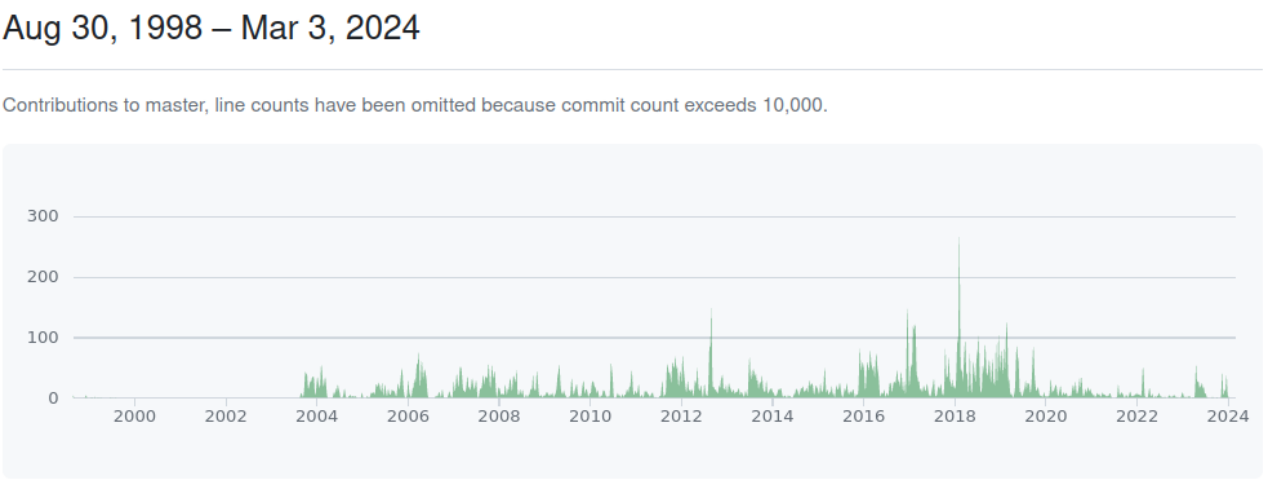

Abbildung 1: Codebeiträge (Contributions) zu Apache JMeter

Quelle: https://github.com/apache/jmeter/graphs/contributors (zugegriffen: 03. März 2024)

Aktuell ist derzeit die Version 5.6.3 und diese wird auch für die vorliegende Arbeit verwendet.

In der weiter oben erwähnten Dateiensammlung von Giuliani et al. (2013) findet sich in der Logdatei eines mit JMeter durchgeführten Testlaufs der Hinweis auf die JMeter-Version 2.3.4. Ob diese Version für alle damals durchgeführten Tests verwendet wurde, ist nicht bekannt. Insgesamt sind lediglich zwei Log-Dateien in der Datensammlung enthalten, je eine für WCS-Test am 07.05.2011 und einen WFS-Test am 17.05.2012 (die Zeitangaben basieren auf den Zeitstempeln in den Logdateien).

10 https://cwiki.apache.org/confluence/display/JMETER/JMeterAndOperatingSystemsTested (zugegriffen: 03. März 2024)
11 https://github.com/apache/jmeter?tab=Apache-2.0-1-ov-file#readme (zugegriffen: 03. März 2024)
12 https://github.com/apache/jmeter (zugegriffen: 03. März 2024)

2.1.2 GeoServer

Das OSGeo-Projekt GeoServer[13] war bereits vor 10 Jahren in der GIS-Community weit verbreitet und gehörte in der Version 2.1.1 neben ArcGIS Server[14] zu den beiden WFS-/WCS-Implemtierungen, welche von Giuliani et al. (2013) für die Performancetests verwendet wurden. ArcGIS Server scheidet für die vorliegende Arbeit aus, da nur FOSS zum Einsatz kommen soll. GeoServer hingegen wurde verwendet. Die Installation erfolgt per GeoServer Docker Image[15].

Im Gegensatz zu WFS sind OGC API – Features für GeoServer noch nicht Bestandteil der offiziellen Releases und stehen als sogenanntes *GeoServer community module* zur Verfügung. Community modules werden im Allgemeinen als experimentell angesehen und befinden sich oft noch in Entwicklung.[16] Dies triff laut GeoServer-Dokumentation derzeit auch auf die OGC API modules, zu denen auch eine Implementierung der OGC API – Features gehört, zu.[17]

Verwendet wurde ein 2.24-SNAPSHOT welcher bei den durchgeführten Tests am 10.02.2024, als sogenannter *Nightly build* zur Verfügung stand. Ein Nightly build wurde verwendet, weil das OGC API community module zusammen mit diesen gebaut wird und so unkompliziert zur Verfügung steht. Würde man ein offizielles GeoServer-Release mit einem community module verwenden wollen, müsste dieses laut Dokumentation[18] selbst aus dem Quellcode gebaut werden.

2.1.3 QGIS Desktop und Server

QGIS[19], als am weitesten verbreitete freie GIS-Software der Welt, trägt auch in steigendem Maße zu wissenschaftlicher Forschung bei (Rosas-Chavoya et al. 2022). Neben QGIS

13 https://geoserver.org/ / https://www.osgeo.org/projects/geoserver/ (zugegriffen: 03. März 2024)
14 https://enterprise.arcgis.com/de/server/ (zugegriffen: 03. März 2024)
15 https://github.com/geoserver/docker (zugegriffen: 03. März 2024)
16 https://docs.geoserver.org/stable/en/user/community/index.html (zugegriffen: 03. März 2024)
17 https://docs.geoserver.org/stable/en/user/community/ogc-api/index.html (zugegriffen: 03. März 2024)
18 https://docs.geoserver.org/latest/en/user/community/index.html (zugegriffen: 03. März 2024)
19 https://qgis.org (zugegriffen: 05. März 2024)

Desktop gibt es auch eine Server Komponente, für welche neben WFS auch bereits der Standard OGC API – Features implementiert ist.[20] Somit bietet es sich als weitere Serversoftware für die Performancetests im Rahmen dieser Arbeit an. Im Gegensatz zu der OGC API - Features Implementierung bei GeoServer ist dieser Standard bereits fester Bestandteil von QGIS Server und steht zur Verfügung, sobald ein WFS bereit gestellt wird.

Für die vorliegende Arbeit wurde QGIS Server manuell mit den Standardeinstellungen in Verbindung mit Apache Webserver[21] v2.4.52 installiert.

Einzige Ausnahme:

Die Umgebungsvariable `QGIS_SERVER_API_WFS3_MAX_LIMIT` wurde auf den Wert `999999` gesetzt, damit für OGC API – Feature-Requests an QGIS Server auch alle angefragten Objekte ausgeliefert werden.

QGIS Desktop wurde für die Konfiguration der QGIS-Projektdatei für den QGIS-Server verwendet.

Zur Anwendung kam sowohl für QGIS Desktop und Server jeweils die Version 3.34.2-Prizren aus dem offiziellen QGIS-Repository.

2.2 Hardware

Für die Bereitstellung der OGC API – Features und WFS mittels GeoServer und QGIS Server wird ein Cloud Server der Firma Hetzner mit Standort Nürnberg verwendet.[22] Hier gibt es diverse Konfigurationsmöglichkeiten und die Hardwareausstattung des Cloud Servers, wie Anzahl vCPUs und RAM, lässt sich auch ohne Neuinstallation der Software verändern.

Dies ist sehr praktisch, da so für die vorliegende Arbeit unterschiedliche Hardware-konfigurationen mit vertretbarem Kostenaufwand verwendet werden konnten. Konkret

20 https://docs.qgis.org/3.34/en/docs/server_manual/ (zugegriffen: 05. März 2024)
21 https://httpd.apache.org/ (zugegriffen: 19. März 2024)
22 https://www.hetzner.com/de/cloud/ (zugegriffen: 09. März 2024)

kamen die Varianten CCX23 und CCX33 mit Dedicated vCPU zur Anwendung. Auf dem Cloud-Server lief als Linux-Betriebssystem Ubuntu 22.04.3 LTS.

Servertyp	**vCPU**[23]	**RAM**	**Datenspeicher**	**Betriebssystem**	**Standort**
CCX23	4	16 GB	NVMe SSD	Ubuntu 22.04.3 LTS	Nürnberg (D)
CCX33	8	32 GB	NVMe SSD	Ubuntu 22.04.3 LTS	Nürnberg (D)

Tabelle 1: Verwendete Cloud-Server für GeoServer und QGIS Server

Als Client, auf welchem JMeter zum Einsatz kam, wurde ein PC mit folgender Konfiguration am Standort Mönchengladbach verwendet:

CPU	**RAM**	**Datenspeicher**	**Betriebssystem**	**Standort**
AMD® Ryzen 9 3950x 16-core processor	128 GB	NVMe SSD Force MP600	Ubuntu 22.04.3 LTS	Mönchengladbach (D)

Tabelle 2: Konfiguration verwendeter Client PC

Die Kommunikation zwischen Client und Cloud Server erfolgte über eine Standard 100 MBit/s DSL-Verbindung.

2.3 Testdatensatz

Die für die Performancetests verwendeten Daten sind ein sehr interessanter Parameter. Die Untersuchungen von Giuliani et al. (2013) liegen bekanntlich bereits mehr als ein Jahrzehnt zurück. In dieser Zeit wurden Software und Hardware weiter entwickelt, aber die damals verwendeten Daten könnten, sofern diese heute noch verfügbar sind, eine Konstante im Vergleich der Tests von heute zu damals darstellen.

23 Laut Anbieter werden AMD Milan EPYC™ 7003 und AMD Genoa EPYC™ 9654 Prozessoren verwendet.

Giuliani et al. (2013) verwendeten damals für die WFS-Tests insgesamt 3 Datensätze:

- Polygondatensatz: Global Lakes and Wetlands Database Level 2 (GLWD-2)

- Liniendatensatz: TIGER Datensatz für Texas mit Straßen und Flüssen (2008)

- Punktdatensatz: ESA ATSR World Fire Atlas (1997 bis 2008)

Erfreulicherweise ist der Polygondatensatz nach wie vor, wenn auch über eine andere URL als damals, über die Webseite des WWF verfügbar.[24] Das hat womöglich auch damit zu tun, dass die dem Datensatz zugrunde liegende Forschungsarbeit von Lehner und Döll (2004) laut ScienceDirect bisher 1635 Mal zitiert worden und augenscheinlich auch 20 Jahre nach Veröffentlichung immer noch von Bedeutung ist.[25]

Für die Quelle des Liniendatensatz verweisen Giuliani et al. (2013) auf eine im FOSS4G-Wiki verlinkte Datei, welche aus dem TIGER-Datensatz für Texas generiert wurde. Mittlerweile ist diese Datei nicht mehr abrufbar.[26] Die zugrunde liegenden TIGER-Daten sind wohl noch beim United States Census Bureau[27] verfügbar, aber es ist nicht mehr nachvollziehbar, wie genau der Testdatensatz aus diesem generiert wurde.

Der Punktdatensatz ist ebenfalls nicht mehr auffindbar. Es existieren noch Datensätze zum ESA ATSR World Fire Atlas[28], aber auch hier ist nicht mehr nachvollziehbar, wie der Testdatensatz damals zusammengestellt wurde.

Somit ist von diesen drei Datensätzen heute nur noch der Polygondatensatz verfügbar, aber genau dieser Datensatz wurde damals fast ausschließlich für die Testreihen verwendet.

In dem Datenpaket, welches Giuliani et al. (2013) dem Autor der vorliegenden Arbeit zur Verfügung gestellt haben, findet sich auch eine CSV-Datei (siehe Anlage A) mit den Bounding-Boxes, welche damals für die WFS-Testreihen verwendet wurden. Es handelt sich hierbei um insgesamt 79 Bounding-Boxes von verschiedenen Ländern.

---

24 https://www.worldwildlife.org/publications/global-lakes-and-wetlands-database-small-lake-polygons-level-2 (zugegriffen: 10. März 2024)
25 https://www.sciencedirect.com/science/article/abs/pii/S0022169404001404 (zugegriffen: 10. März 2024)
26 https://wiki.osgeo.org/wiki/Benchmarking_2009#Download (zugegriffen: 19. März 2024)
27 https://www.census.gov/ (zugegriffen: 23. März 2024)
28 http://due.esrin.esa.int/page_wfa.php (zugegriffen: 23. März 2024)

Somit konnte, trotz des großen Zeitabstandes zu den Tests von damals, für die aktuellen Testreihen ein identischer Testdatensatz verwendet werden.

Es handelt sich hierbei um ein Shapefile, bestehend aus den Dateien *.shp, *.shx, *.sbn, *.sbx, *.dbf, *.avl, mit insgesamt 244892 Objekten und je 7 Attributen.

Der Datensatz beinhaltet demnach keine .prj Datei. Die Dokumentation gibt an, dass die Daten in einem geographische Koordinatensystem vorliegen. Für die vorliegende Arbeit wurde, wie auch bereits von Giuliani et al. (2013), als Koordinatenbezugssystem EPSG:4326 verwendet.

Die weltweite Ausdehnung beträgt in Längen- und Breitengraden:

```
minX,minY,maxX,maxY
-180,-55.5872078,180,83.5759506
```

2.4 Tests

Wie bereits in Abschnitt 2.1.1 ausgeführt, werden die Performancetests im Rahmen dieser Arbeit mit JMeter durchgeführt.

JMeter verfügt über eine grafische Benutzeroberfläche, mit welcher Testpläne erstellt und als .jmx-Datei gespeichert werden können.

Insgesamt wurden 4 Testpläne aufgestellt:

- GeoServer WFS

- GeoServer OGC API – Features

- QGIS Server WFS

- QGIS Server OGC API – Features

Die WFS-Testpläne entsprechen denen, die Giuliani et al. (2013) verwendet haben, wobei

natürlich die Server-Endpoints als auch die Namen der FeatureTypes, unter denen Geoserver und QGIS Server den Testdatensatz GLWD-2 auf Basis der Shapedatei bereitstellen, abweichen. Wie auch bereits von Giuliani et al. (2013), wurde die Version 1.1.0 des WFS-Standards verwendet. Die Server liefern auf den HTTP-GET-Request die Antwort im GML-Format.

WFS-Endpoint GeoServer:

`http://<server>:<port>/geoserver/lakes/ows?`

WFS-Endpoint QGIS Server:

`http://<server>/cgi-bin/qgis_mapserv.fcgi?`

Abgesetzt wurden jeweils GetFeature-Requests mit folgenden Parametern (für die Dienste wurde jeweils EPSG:4326 als Standard-KBS eingerichtet, so dass eine Angabe des SRS-Parameters im Request nicht erforderlich war):

Parameter Key	**Parameter Value**	**Bemerkung**
service	WFS	
request	getfeature	
version	1.1.0	
bbox	xmin,ymin,xmax,ymax	z.B. 60,29,75,38 aus CSV
typename	name	z.B. glwd_2_shp

Tabelle 3: Verwendete Parameter für WFS-Requests

Die Testpläne sind so konfiguriert, dass die Requests für den BBOX-Parameter Werte der Reihe nach aus der CSV-Datei (siehe Abschnitt 2.3) erhalten. Wurden die letzten BBOX-Werte aus der Datei verwendet, wird wieder mit dem ersten Datensatz begonnen.

Es werden also GetFeature-Requests mit einem räumlichen Filter, einer BBOX, abgesetzt.

Wie in der Einleitung beschrieben, liegt für den Standard OGC API - Features zum Zeitpunkt der Erstellung der vorliegenden Arbeit der *Part 3: Filtering* nur im Entwurf vor.

Aber dabei geht es um komplexere Filter und bereits *Part 1: Core* definiert das räumliche Filtern unter Verwendung eines BBOX-Parameters und sowohl GeoServer als auch QGIS Server unterstützen dies (Portele et al. 2022). Somit ist es möglich, für die Testpläne für die Implementierungen von OGC API -Features, genau wie bei den WFS-Tests, die existierende CSV-Datei mit den BBOX-Angaben zu verwenden.

Die Server-Endpoints für die verwendeten OGC API – Features Implementierungen sehen wie folgt aus, wobei *collections* hier in etwa den WFS-FeatureTypes entspricht:

OGC API – Features Endpoint GeoServer:

`http://<server>:<port>/geoserver/ogc/features/v1/collections/lakes:glwd_2/items?`

OGC API – Features Endpoint QGIS Server:

`http://<server>/cgi-bin/qgis_mapserv.fcgi/wfs3/collections/glwd_2_shp/items.geojson?`

Entsprechend des Standards sind, im Vergleich zum WFS, weniger Parameter nötig.

Die Server sollen gemäß der Empfehlung des Standards OGC API – Features die Antwort für den HTTP-GET-Request im Format GeoJSON liefern. Beim GeoServer ist zur Bestimmung des Ausgabeformats ein f-Parameter notwendig. Beim QGIS-Server ist das Ausgabeformat im Endpoint enthalten.

Zusätzlich zum BBOX-Parameter wird noch ein limit-Parameter verwendet, um sicher zu stellen, dass auch alle Objekte ausgeliefert werden (siehe auch Abschnitt 2.1.3 zur Umgebungsvariable `QGIS_SERVER_API_WFS3_MAX_LIMIT`).

Parameter Key	**Parameter Value**	**Bemerkung**
f	application/geo%2Bjson	nur GeoServer
limit	999999	fester Wert (der Testdatensatz hat 244892 Objekte)
bbox	xmin,ymin,xmax,ymax	z.B. 60,29,75,38 aus CSV

Tabelle 4: Verwendete Parameter für OGC API - Features-Requests

Das Absetzen der so definierten Requests wird in JMeter in den Testplänen durch so genannte *Thread Groups* gesteuert. Wobei für jede *Thread Group* festgelegt wird, wie viele Request gleichzeitig und wie oft abgesetzt werden. Dies erfolgt über die Angabe von Threads je Thread Group. Die folgende Tabelle, die dem Aufbau von Giuliani et al. (2013) entspricht, gibt einen entsprechenden Überblick:

Threadgroup lfd. Nr.	**Anzahl Threads (Benutzer)**	**Request je Thread**	**Gesamtzahl Requests je Thread Group**
1	1	100	100
2	2	100	200
3	4	100	400
4	8	100	800
5	16	100	1600
			3100 (Summe)

Tabelle 5: Aufbau der JMeter Thread Groups

Die Thread Groups eines Testplans werden nacheinander abgearbeitet. Mit den Threads werden Benutzer simuliert. Beispielsweise gibt es in Thread Group 1 auch einen Thread (Benutzer) der wiederum 100 Requests nacheinander absetzt. Dann folgt die Thread Group 2 in der 2 Threads (Benutzer) parallel, aber jeder für sich nacheinander, 100 Requests absetzen. Somit werden für die Thread Group 2 insgesamt 200 Requests abgesetzt.

Ist der Testplan vollständig durchgelaufen, sind in Summe, nach Abarbeitung aller Thread Groups, 3100 Requests an den Server gesendet worden.

Über die grafische Benutzeroberfläche von JMeter können zwar die Testpläne ausgeführt werden, dies ist aber nur testweise zum Aufsetzen eines Testplans empfohlen. Laut JMeter-Dokumentation muss für optimale Resultate der CLI-Mode (Command Line Mode) verwendet werden.[29]

JMeter kann bei der Ausführung Log-Dateien erzeugen, sowohl bzgl. der abgesetzten Requests als auch bzgl. der Ausführung von JMeter selbst.

29 https://jmeter.apache.org/usermanual/get-started.html#non_gui (zugegriffen: 10. März 2024)

Wie in Abschnitt 2.1.1 ausgeführt, wurde für die vorlegende Arbeit eine sehr viel aktuellere Version von JMeter im Vergleich zu den Untersuchungen von Giuliani et al. (2013) verwendet (JMeter 5.6.3 statt 2.3.4). Deshalb kann praktischer Weise die mit JMeter 3.0.0[30] eingeführte Möglichkeit der dynamischen HTML-Dashboard-Erzeugung[31] aus den Log-Dateien zur Anwendung kommen. Giuliani et al. (2013) haben sich damals eine Auswertung, basierend auf den Log-Dateien, mit einem Skript selbst erstellt.

Die Verwendung von Cloud-Server und DSL-Verbindung ist ein Unterschied zum Setup von Giuliani et al. (2013), bei welchem ausschließlich lokal vorhandene Hardware verwendet wurde, welche über 1GB LAN vernetzt wurde.

Dies hat den Nachteil, dass die zur Verfügung stehende Bandbreite schwanken kann, bildet aber reale Nutzungsbedingungen natürlich besser ab. Um mögliche Einflüsse gering zu halten, wurde dafür Sorge getragen, dass während der durchgeführten Test keine anderen Geräte im Netzwerk des Clients aktiv waren. Auch wurden die diversen Testläufe hintereinander am selben Tag ohne erneuten Aufbau der DSL-Verbindung durchgeführt.

Vor und nach einem Testlauf wurde die verfügbare Bandbreite sowohl beim Upload vom Client zum Server als auch beim Download vom Server zum Client ermittelt und geloggt. Hierfür kam das Tool iperf3[32] zum Einsatz.

Giuliani et al. (2013) haben für jedes Testsetup die Performancemessung mit JMeter drei Mal durchgeführt und für die Auswertung nur den jeweils dritten Durchgang herangezogen weil dieser als am stabilsten angenommen wurde. Dieses Vorgehen wird für die vorliegende Arbeit genau so beibehalten.

Um unterschiedliche Hardwarezuweisungen seitens des Cloud-Server-Anbieters zu vermeiden, wurde ein „rescaling“ des Cloud-Servers von CCX23 auf CCX33 (vgl. Abschnitt 2.2) erst durchgeführt, nachdem die 4 Testpläne (GeoServer / QGIS Server mit je OGC API Features / WFS) auf CCX23 durchgeführt wurden.

Um die verschiedenen Befehle unmittelbar nacheinander über die Kommandozeile absetzen zu können, wurde ein eigens erstelltes Skript run.sh (siehe Anlage B) verwendet.

30 https://jmeter.apache.org/changes_history.html (zugegriffen: 10. März 2024)
31 https://jmeter.apache.org/usermanual/generating-dashboard.html (zugegriffen: 10. März 2024)
32 http://software.es.net/iperf/ (zugegriffen: 10. März 2024)

Dieses Skript sorgt zudem dafür, dass die Log-Dateien für einen Testlauf der Übersichtlichkeit halber in einem eigenen Ordner gespeichert werden, dessen Name wie folgt aufgebaut ist:

(*Name des JMeter-Testplans*) + (*Zeitstempel der Ausführung*) + (*Serverinformation*)

Der verwendete JMeter-Testplan und die Serverinformation werden als Parameter beim Aufruf des Skriptes mit angegeben.

Beispiel:

Der Aufruf

```
./run.sh test_1_wfs.jmx hetzner_ccx23
```

erzeugt einen Ordner mit dem Namen

```
test_1_wfs_20240210T154303_hetzner_ccx23
```

und legt dort alle zu dem Testlauf zugehörigen Dateien ab.

Bei jeder Ausführung des Skripts `run.sh` über die die Kommandozeile werden in dem oben beschrieben Ordner folgende Dateien erzeugt:

- JMeter-Log-Datei (.log)

- JMeter-Ergebnisse des Testlaufs (.jtl)

- Bandbreitenmessung Upload vor Ausführung des JMeter-Testplans

- Bandbreitenmessung Upload nach Ausführung des JMeter-Testplans

- Bandbreitenmessung Download vor Ausführung des JMeter-Testplans

- Bandbreitenmessung Download nach Ausführung des JMeter-Testplans

- ein Ordner mit dem Namen „report“, der den HTML-Report und dafür nötige und entsprechend aufbereitete Daten enthält.

3. Ergebnisse

Nach diversen Testläufen, begleitend zum Aufbau der Testinfrastruktur, wurden die Performancetests am 10.02.2024 im Zeitraum 15:43 bis 20:48 durchgeführt.

Die Ergebnisse und die dazugehörigen Log-Dateien und HTML-Reports, wie im vorherigen Abschnitt beschrieben, stehen alle im folgendem GitHub-Repository zur Verfügung:

https://github.com/pathmapper/performance

Dort finden sich ebenfalls die verwendeten JMeter-Testpläne, das Skript zur Ausführung eines Performancetests und das QGIS-Projekt, welches für den QGIS Server verwendet wurde.

Der verwendete Testdatensatz ist aus lizenzrechtlichen Gründen nicht enthalten, lässt sich aber über die Webseite des WWF beziehen (vgl. Abschnitt 2.3). Die CSV-Datei mit den verwendeten BBOXes ist ebenfalls im GitHub-Repository verfügbar.

Die Testläufe wurden in folgender Reihenfolge durchgeführt:

1. QGIS Server WFS CCX23
2. QGIS Server OGC API - Features CCX23
3. GeoServer WFS CCX23
4. GeoServer OGC API - Features CCX23
5. GeoServer WFS CCX33
6. GeoServer OGC API - Features CCX33
7. QGIS Server WFS CCX33
8. QGIS Server OGC API - Features CCX33

Jeder Testlauf wurde mindestens dreimal durchgeführt und es wurde kontinuierlich überwacht, ob alle Dateien auch im jeweiligen Ordner abgelegt wurden.

Bei 4 Testläufen konnte eine der Bandbreitenmessungen nicht durchgeführt werden, was zu einer leeren Log-Datei hierfür führte. Wenn dies bei dem jeweils dritten Testlauf eines Test-Setups aufgetreten ist, wurde der Test wiederholt, bis alle Ergebnisse vorlagen. So wurden insgesamt 27 Testläufe durchgeführt.

Die Log-Dateien der Bandbreitenmessungen wurden kontinuierlich überprüft. Die Werte für den Download waren durchweg sehr konstant. Hingegen schwankten die Werte für den Upload, waren aber immer mehr als ausreichend.

Zudem wurde mit dem Tool htop[33] währen der Testläufe die Auslastung die CPU- und RAM-Auslastung optisch überwacht. Sowohl bei CCX23 als auch CCX33 waren jeweils ausreichend Reserven vorhanden. Die CPU-Auslastung betrug nie 100% und die RAM-Auslastung betrug bei nie mehr als 3GB.

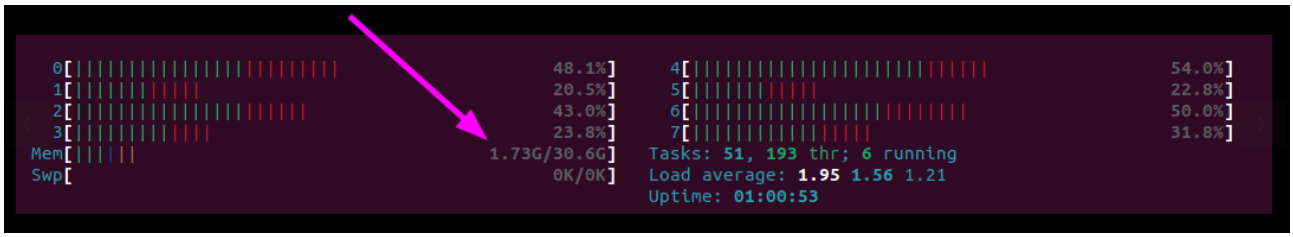

Abbildung 2: Arbeitsspeicherauslastung OAF CCX33 bei 16 Threads unter 2GB (von 32 GB)

Bei der Überprüfung der Log-Dateien des QGIS-Servers fanden sich zahlreiche Fehlermeldungen:

```
ERROR 1: Inconsistent shape count for bin
```

Dazu mehr im folgenden Abschnitt.

33 https://htop.dev/ (zugegriffen: 15. März 2024)

4. Diskussion

4.1 Fehlermeldung „Inconsistent shape count for bin“

Zunächst war nicht klar, woher diese Fehlermeldung stammt, was diese bedeutet und was die Ursache hierfür ist.

Eine Suche im Quellcode von QGIS ergab keinen Treffer. Aber mit dem Wissen, dass QGIS für das Lesen von Shapedateien die Softwarebibliothek GDAL[34] verwendet und GDAL wiederum eine eigene Kopie der Bibliothek Shapelib[35], konnte die Fehlermeldung dort lokalisiert werden.[36]

Shapelib ist eine FOSS-Bibliothek für den Umgang mit Shapedateien, die auch in anderen Softwareprojekten wie z.B. PostGIS[37] und GRASS GIS[38] verwendet wird.

Beim Shapefile-Format handelt es sich um ein proprietäres Datenformat, welches die Firma ESRI nur zum Teil öffentlich dokumentiert hat.[39]

Ein Shapefile besteht bekanntermaßen aus mehreren Dateien, wobei einige Dateien optional sind.

Wie in Abschnitt 2.3 beschrieben, sind für den verwendeten GLWD-2 Datensatz folgende Dateien vorhanden: *.shp, *.shx, *.sbn, *.sbx, *.dbf, *.avl

Bei .sbn / .sbx handelt es sich um einen räumlichen Index und hier hat auch die gezeigte Fehlermeldung ihren Ursprung.

Verkürzt besagt die Fehlermeldung, dass der räumliche Index inkonsistent ist und daher in der Folge nicht verwendet wird.

34 https://gdal.org (zugegriffen: 15. März 2024)
35 https://github.com/OSGeo/shapelib (zugegriffen: 15. März 2024)
36 https://github.com/OSGeo/shapelib/blob/644559c027f2bb564f9d0c0657ebbab0b97708df/sbnsearch.c#L540 und https://github.com/OSGeo/shapelib/blob/644559c027f2bb564f9d0c0657ebbab0b97708df/sbnsearch.c#L642 (zugegriffen: 15. März 2024)
37 https://postgis.net/ (zugegriffen: 15. März 2024)
38 https://grass.osgeo.org/ (zugegriffen: 15. März 2024)
39 https://www.esri.com/content/dam/esrisites/sitecore-archive/Files/Pdfs/library/whitepapers/pdfs/shapefile.pdf (zugegriffen: 15. März 2024)

Im Folgenden der Debug-Output von gdalinfo mit einem räumlichen Filter:

```
ogrinfo -ro -so -spat 5 5 6 6 glwd_2.shp --debug on

Shape: DBF Codepage = LDID/87 for glwd_2.shp

Shape: Treating as encoding 'ISO-8859-1'.

GDAL: GDALOpen(glwd_2.shp, this=0x55765f5e0090) succeeds as ESRI
Shapefile.

INFO: Open of `glwd_2.shp'

      using driver `ESRI Shapefile' successful.

OGR: GetLayerCount() = 1



Layer name: glwd_2

Metadata:

  DBF_DATE_LAST_UPDATE=2003-05-20

Geometry: Polygon

ERROR 1: Inconsistent shape count for bin

SHAPE: Used spatial index, got 0 matches.

Feature Count: 13

Extent: (-180.000000, -55.587208) - (180.000000, 83.575951)

Layer SRS WKT:

(unknown)

GLWD_ID: Integer64 (10.0)

TYPE: String (12.0)

POLY_SRC: String (12.0)
```

```
AREA_SKM: Real (12.1)

PERIM_KM: Real (12.1)

LONG_DEG: Real (10.2)

LAT_DEG: Real (10.2)

GDAL: GDALClose(glwd_2.shp, this=0x55765f5e0090)
```

Hier erscheint die gleiche Fehlermeldung wie in den Log-Dateien des QGIS Server, und der Hinweis, dass der räumliche Index verwendet wurde, aber keinen Treffer ergab. Trotzdem konnten 13 Objekte für die BBOX (5,5,6,6) gefunden werden. In QGIS Desktop konnte manuell für verschieden BBOXes verifiziert werden, dass GDAL trotz des Fehlers bzgl. des räumlichen Index, alle Objekte liefert.

Der Fehler hat somit Auswirkung auf die Geschwindigkeit mit der Objekte für einen räumlichen Filter, wie z.B. eine BBOX, geliefert werden, aber nicht auf die Korrektheit der gesuchten Objekte insgesamt.

Das hat natürlich einige Fragen aufgeworfen, stammt der Datensatz doch aus der wissenschaftlichen Arbeit von Lehner und Döll (2004) und wird in einem wissenschaftlichen Kontext bis heute verwendet[40].

Und wenn der räumliche Index wirklich kaputt ist, warum ist Giuliani et al. (2013) das bei den eigenen Tests nicht aufgefallen, obwohl für Requests im Rahmen der Testreihen ja auch der BBOX-Parameter mit Werten aus der besagten CSV-Datei verwendet wurde?

Es stand also der Verdacht im Raum, dass die Fehlermeldung nicht zutreffend ist („false positive“) und so wurden vom Autor der vorliegenden Arbeit 2 Tickets erstellt:

- ein Ticket im GitHub-Repository von Shapelib[41]

- ein Ticket im GitHub-Repository von GDAL[42]

Die Reaktion aus der OpenSource-Community folgte prompt. Keine 48 Stunden später wurde bestätigt, dass es sich um einen grundlegenden algorithmischen Fehler handelte, der

40 https://www.sciencedirect.com/science/article/pii/S0022169404001404 (zugegriffen: 15. März 2024)
41 https://github.com/OSGeo/shapelib/issues/106 (zugegriffen: 15. März 2024)
42 https://github.com/OSGeo/gdal/issues/9430 (zugegriffen: 15. März 2024)

schon immer bestand und auf einer falschen Annahme über die Reihenfolge einiger Elemente in der .sbn-Datei basierte.[43] Pull Requests zur Fehlerbehebung wurden eingereicht und innerhalb einer Woche in die Bibliotheken GDAL und Shapelib integriert.[44] In GDAL wird ab Version 3.8 der Fehler behoben sein und somit auch in QGIS Desktop und QGIS Server, sobald dort GDAL 3.8 verwendet wird.

Für die vorliegende Arbeit bedeutet dies, dass für die Tests mit QGIS Server kein räumlicher Index verwendet wurde. Hinsichtlich der Tests mit GeoServer ergab eine weitere Recherche[45] und Verifikation, dass hier ohnehin kein räumlicher Index im Format .sbn / .sbx verwendet werden kann und statt dessen ein Quad-Tree Index (.qix-Datei) erzeugt und verwendet wird.

Hierbei handelt es sich um einen von der OpenSource-Community entwickelten räumlichen Index als Alternative zu .sbn/.sbx. 2012 ist gelungen den von ESRI nicht veröffentlichten, proprietären räumliche Index zu verstehen (Reverse Engineering)[46], weshalb es heutzutage überhaupt möglich ist, diesen in FOSS-Anwendungen zu verwenden.

4.2 Vergleich ausgewählter Test-Setups

Als Maß für die Performance wurde, wie auch schon von Giuliani et al. (2013) der Durchsatz (Throughput) verwendet. Der Durchsatz wird als Anzahl der Anfragen pro Zeit berechnet. Die Zeit wird vom Beginn der ersten Stichprobe bis zum Ende der letzten Stichprobe berechnet. Dies schließt alle Intervalle zwischen den Stichproben ein.

Die Formel lautet: Durchsatz = (Anzahl der Anfragen) / (Gesamtzeit)[47]

43 https://github.com/OSGeo/gdal/issues/9430#issuecomment-1987262251 (zugegriffen: 15. März 2024)
44 https://github.com/OSGeo/shapelib/pull/114 und https://github.com/OSGeo/gdal/pull/9439 (zugegriffen: 15. März 2024)
45 https://docs.geoserver.org/stable/en/user/data/vector/directory.html (zugegriffen: 15. März 2024)
46 https://trac.osgeo.org/gdal/ticket/4719, http://erouault.blogspot.com/2012/06/gdalogr-using-shapefile-native-sbn.html, http://geospatialpython.com/2011/10/your-chance-to-make-gis-history.html (zugegriffen: 16. März 2024)
47 https://jmeter.apache.org/usermanual/glossary.html (zugegriffen: 10. März 2024)

Die folgenden Tabellen und Diagramme stellen die Ergebnisse der verschiedenen durchgeführten Testläufe für ausgewählte Test-Setups zum Vergleich und zur Diskussion dar.

4.2.1 Vergleich WFS/OAF CCX23 QGIS Server

Threads	WFS CCX23 QGIS Server	OAF CCX23 QGIS Server
1	4,64	4,28
2	8,60	8,65
4	10,93	13,73
8	12,11	20,43
16	11,92	21,68

Tabelle 6: Vergleich WFS/OAF CCX23 QGIS Server

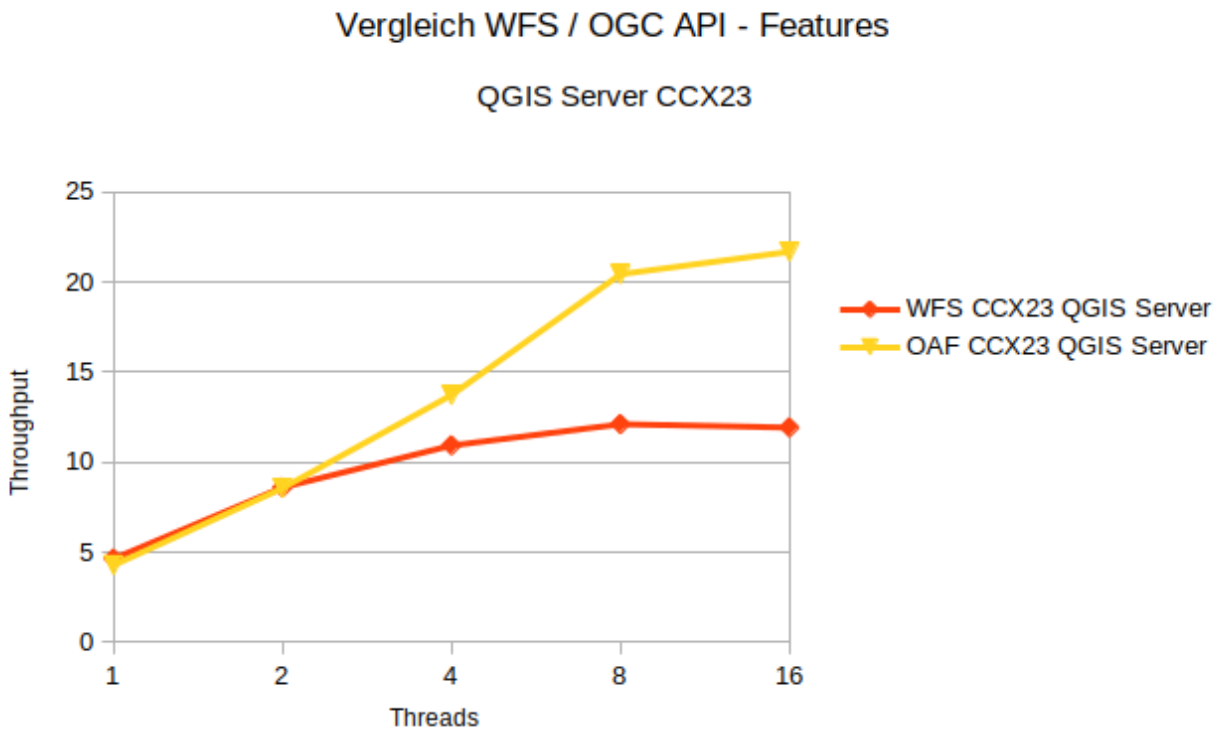

Abbildung 3: Vergleich WFS/OAF CCX23 QGIS Server

Für die kleinere der beiden getesteten Hardwarekonfigurationen des Cloud-Servers (CCX23) bieten die OGC API – Features ab 4 Threads eine bessere Performance, welche bei den Threadgroups mit 8 und 16 Threads weiter ansteigt. Der Anstieg der Performance von 8 auf 16 Threads ist nicht mehr so stark wie im Vergleich zum Anstieg bei 4 auf 8 Threads.

Bei 16 Threads bieten die OGC API -Features eine um ca. 82% bessere Performance als die WFS.

4.2.2 Vergleich WFS/OAF CCX23/CCX33 QGIS Server

Werden zu dem vorherigen Vergleich WFS/OAF CCX23 QGIS Server die Ergebnisse der Testläufe mit der stärkeren Hardwarekonfiguration CCX33 hinzu gefügt, ergibt sich folgendes Bild:

Threads	WFS CCX23 QGIS Server	OAF CCX23 QGIS Server	WFS CCX33 QGIS Server	OAF CCX33 QGIS Server
1	4,64	4,28	4,97	4,57
2	8,60	8,65	8,40	9,19
4	10,93	13,73	11,30	15,60
8	12,11	20,43	12,36	22,53
16	11,92	21,68	11,95	21,96

Tabelle 7: Vergleich WFS/OAF CCX23/CCX33 QGIS Server

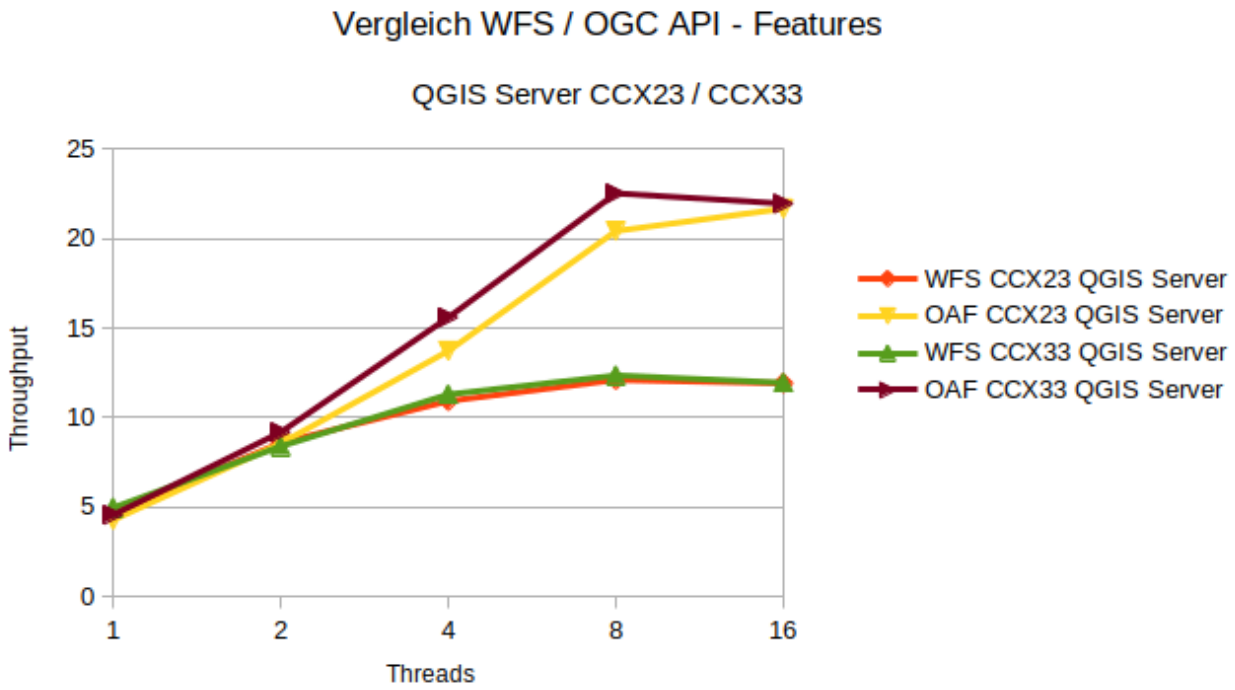

Abbildung 4: Vergleich WFS/OAF CCX23/CCX33 QGIS Server

Threads	**Performanceunterschied OAF CCX33 / CCX 23 (gerundet)**
1	+7%
2	+7%
4	+14%
8	+10%
16	+1%

Tabelle 8: Performanceunterschied OAF CCX33 / CCX 23

Durch die stärkere Hardware besteht ein leichter prozentualer Performancegewinn bei den OGC API - Features, welcher aber bei 8 und vor allem bei 16 Threads rückläufig ist. Ursache könnte hier das Ausreizen der zur Verfügung stehenden Bandbreite sein.

4.2.3 Vergleich WFS/OAF CCX23 GeoServer

Threads	WFS CCX23 Geoserver	OAF CCX23 Geoserver
1	10,17	5,73
2	17,17	10,84
4	23,86	15,56
8	26,36	20,30
16	27,15	20,38

Tabelle 9: Vergleich WFS/OAF CCX23 GeoServer

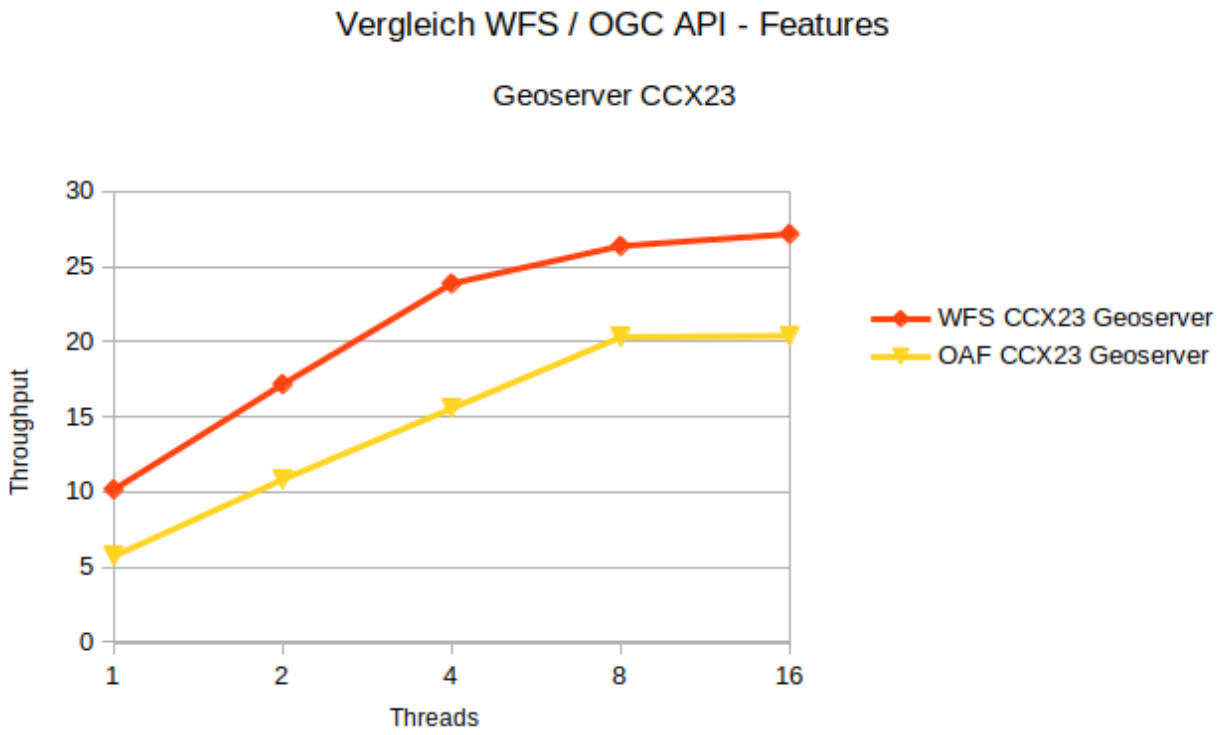

Abbildung 5: Vergleich WFS/OAF CCX23 GeoServer

Beim Vergleich OGC API - Features / WFS mit GeoServer zeigt sich ein anderes Bild im Vergleich zum QGIS Server, da hier der WFS performanter ist.

Eine mögliche Erklärung könnte sein, dass OGC API - Features aktuell nicht fester Bestandteil vom GeoServer (vgl. Abschnitt 2.1.2) sind, sich noch in der Entwicklung befinden und die Einbindung als *community module* mit Nachteilen in der Performance im Vergleich zu einer Integration in den GeoServer-Core einhergeht.

4.2.4 Vergleich WFS/OAF CCX23/CCX33 GeoServer

Threads	WFS CCX23 GeoServer	OAF CCX23 GeoServer	WFS CCX33 GeoServer	OAF CCX33 GeoServer
1	10,17	5,73	10,24	5,70
2	17,17	10,84	16,77	11,56
4	23,86	15,56	24,22	16,67
8	26,36	20,30	27,46	20,35
16	27,15	20,38	27,39	20,21

Tabelle 10: Vergleich WFS/OAF CCX23/CCX33 GeoServer

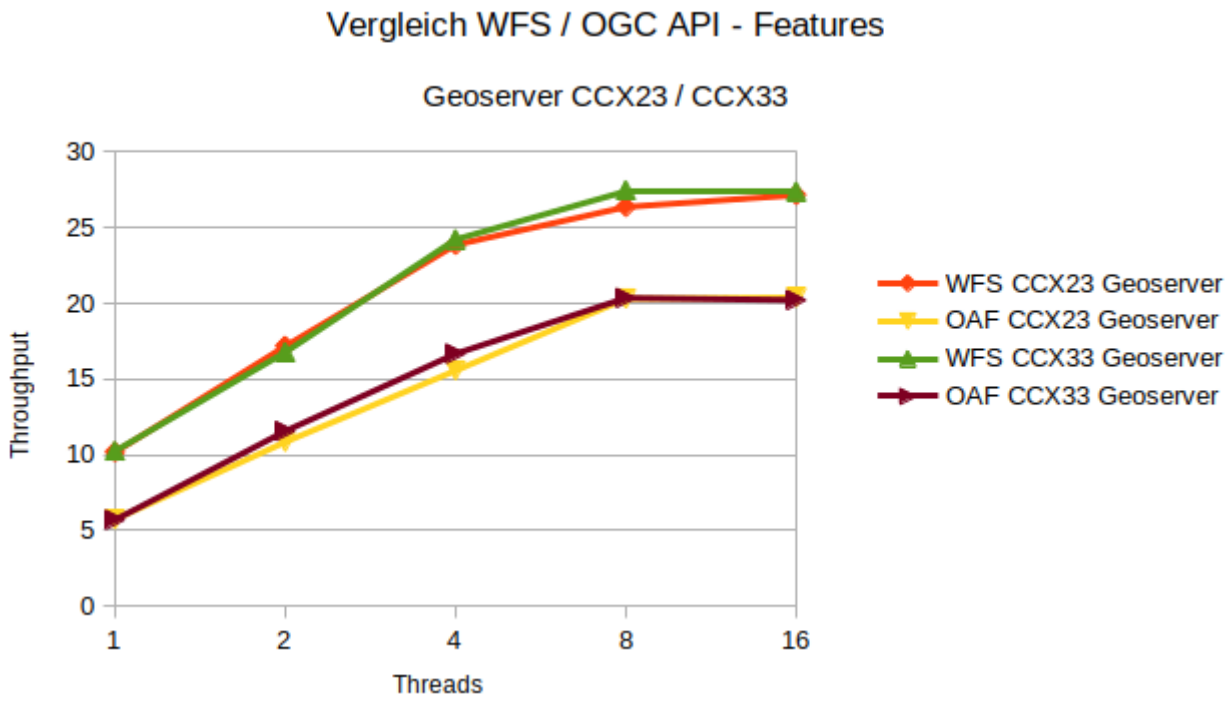

Abbildung 6: Vergleich WFS/OAF CCX23/CCX33 GeoServer

Die stärkere Hardware CCX33 wirkt sich beim GeoServer im Vergleich zu CCX23 kaum merklich aus. Das gilt sowohl für WFS als auch für OGC API – Features.

4.2.5 Vergleich WFS/OAF CCX23 GeoServer/QGIS Server

Threads	WFS CCX23 GeoServer	OAF CCX23 GeoServer	WFS CCX23 QGIS Server	OAF CCX23 QGIS Server
1	10,17	5,73	4,64	4,28
2	17,17	10,84	8,60	8,65
4	23,86	15,56	10,93	13,73
8	26,36	20,30	12,11	20,43
16	27,15	20,38	11,92	21,68

Tabelle 11: Vergleich WFS/OAF CCX23 GeoServer/QGIS Server

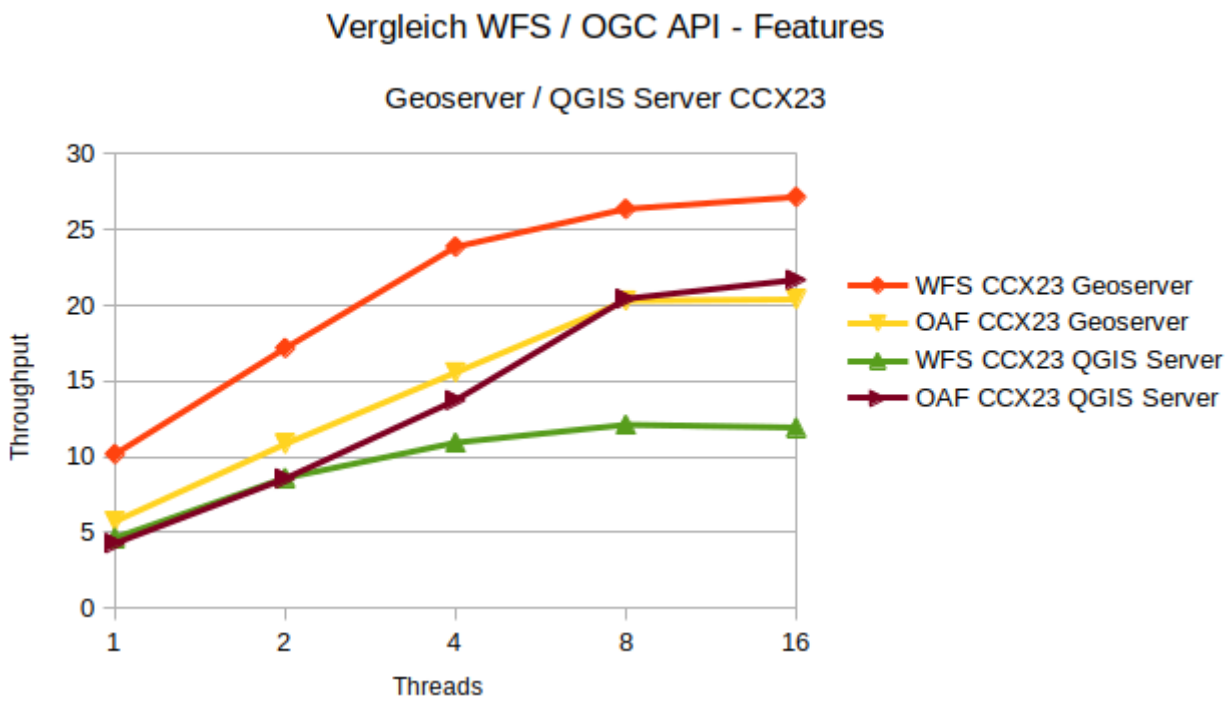

Abbildung 7: Vergleich WFS/OAF CCX23 GeoServer/QGIS Server

Der Vergleich zwischen den beiden getesteten Server-Implementierungen fällt bei WFS mit mehr als der doppelten Performance eindeutig für GeoServer aus.

Bei den OGC API - Features sind beide Server ungefähr gleich performant, bei 1 bis 4 Threads leichte Vorteile beim GeoServer, bei 8 Threads sind beide Server gleichauf und bei 16 Threads bestehen leichte Vorteile für den QGIS Server.

Wie in Abschnitt 4.1 erläutert, verwendet QGIS Server bei den durchgeführten Tests im Gegensatz zu GeoServer keinen räumlichen Index, so dass die Performance für WFS und OGC API - Features von QGIS Server noch etwas ansteigen dürfte, sobald hier auch der Geschwindigkeitsvorteil eines entsprechenden Index genutzt werden kann.

Schon bei den von Giuliani et al. (2013) durchgeführten Tests zeigte der WFS auf Basis des GeoServer eine deutlich bessere Performance im Vergleich zum damals getesteten ArcGIS Server.

4.2.6 Vergleich WFS/OAF CCX23 GeoServer/QGIS Server

Threads	WFS CCX33 GeoServer	OAF CCX33 GeoServer	WFS CCX33 QGIS Server	OAF CCX33 QGIS Server
1	10,24	5,70	4,97	4,57
2	16,77	11,56	8,40	9,19
4	24,22	16,67	11,30	15,60
8	27,46	20,35	12,36	22,53
16	27,39	20,21	11,95	21,96

Tabelle 12: Vergleich WFS/OAF CCX23 GeoServer/QGIS Server

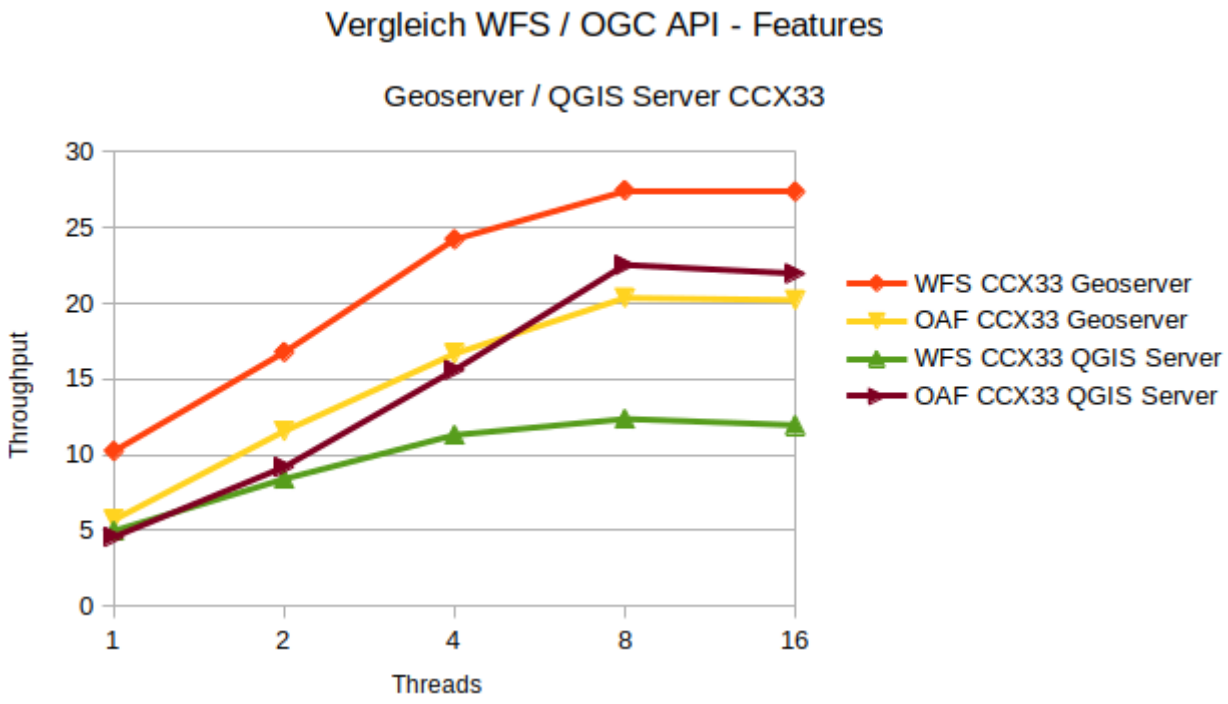

Abbildung 8: Vergleich WFS/OAF CCX23 GeoServer/QGIS Server

Beim Serververgleich unter Verwendung der stärkeren Hardware CCX33 zeigt sich der QGIS Server gegenüber dem GeoServer bei 8 Threads leicht verbessert. Wie schon zuvor gesehen, gibt es insgesamt eine leichte Verbesserung für die Performance der OGC API – Features auf Basis QGIS Server bei Verwendung der stärken Hardware. Bei den WFS wirkt sich die leistungsstärkere Hardware bei beiden getesteten Servern nicht positiv auf die Performance aus.

4.2.7 Vergleich WFS/OAF CCX23/2013 GeoServer

Threads	WFS CCX23 Geoserver	WFS Geoserver 2013
1	10,17	6,5
2	17,17	12,1
4	23,86	13,5
8	26,36	13,2
16	27,15	12,6

Tabelle 13: Vergleich WFS/OAF CCX23/2013 GeoServer

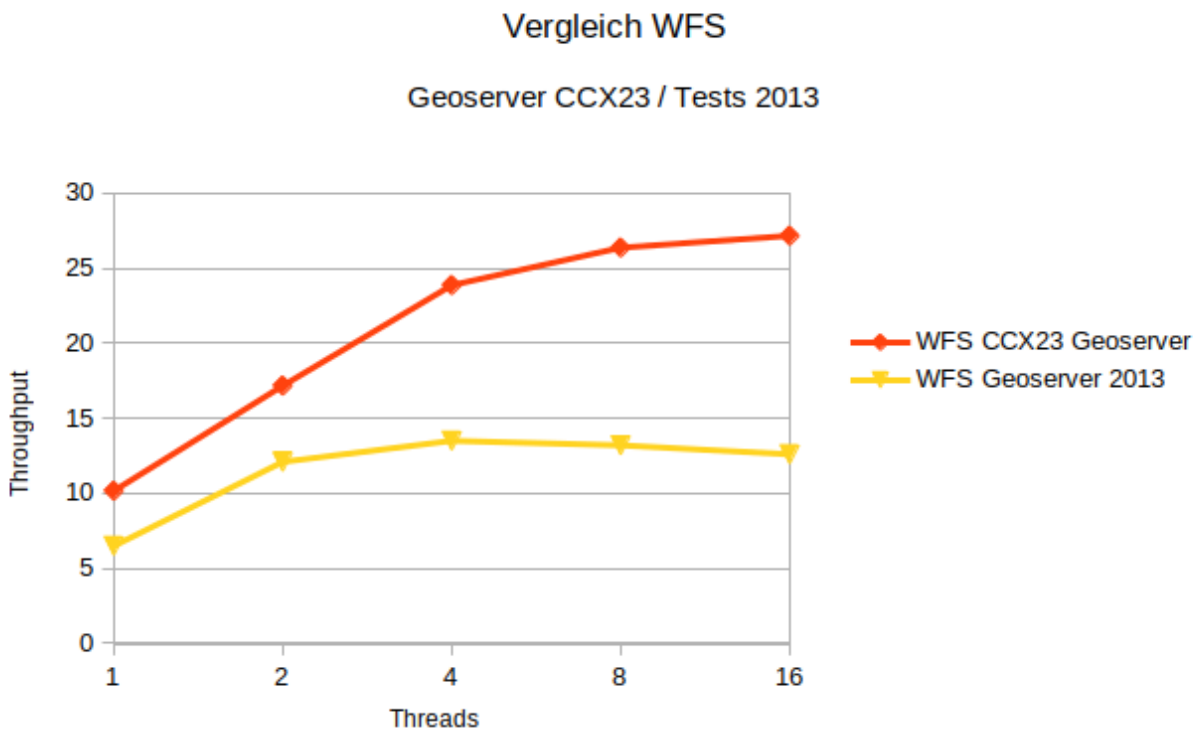

Abbildung 9: Vergleich WFS/OAF CCX23/2013 GeoServer

Auf aktueller Hardware zeigt der GeoServer im Vergleich zum entsprechenden Test von Giuliani et al. (2013) stark verbesserte Performance. Giuliani et al. (2013) vermuteten für die nachlassende Performance bei mehr als 4 Threads einen Flaschenhals in Bezug auf die Festplattenzugriffe, da auch vermehrt Fehler des Dienstes festgestellt wurden.

Damals kamen HDD zur Anwendung. Dies ist mit den heute verwendeten SSD nicht mehr der Fall, der Performanceunterschied steigt ab 4 Threads weiter an und die Performance ist bei 16 Threads mehr als doppelt so hoch wie vor rund 10 Jahren (und das vor dem Hintergrund, dass damals eine sehr viel höhere Bandbreite zur Verfügung stand).

4.3 DSL-Verbindung

Die verwendete DSL-Verbindung erwies sich hinsichtlich der Bandbreite für den Download als sehr stabil.

Die folgende Tabelle listet den gesamten Durchsatz (Throughput, vgl. Abschnitt 4.2) je Testplan, also über alle Threadgroups mit jeweils insgesamt 3100 Requests, auf:

Test-Setup	**Testlauf #1**	**Testlauf #2**	**Testlauf #3**	**Testlauf #4**	**Testlauf #5**	**Min**	**Max**	**Spannweite**
QGIS Server WFS CCX23	11,00	11,10	11,00			11,00	11,10	0,10
QGIS Server OAF CCX23	16,00	16,28	16,42			16,00	16,42	0,42
QGIS Server WFS CCX33	11,14	11,17	11,15			11,14	11,17	0,03
QGIS Server OAF CCX33	16,78	17,21	17,52	17,45		16,78	17,52	0,74
GeoServer WFS CCX23	23,13	24,18	24,14	24,22	24,31	23,13	24,31	1,18
GeoServer OAF CCX23	17,37	17,22	17,27			17,22	17,37	0,15
GeoServer WFS CCX23	23,86	24,09	24,64			23,86	24,64	0,78
GeoServer OAF CCX33	17,64	17,67	17,48			17,48	17,67	0,19

Tabelle 14: Durchsatz je Testlauf für alle Testläufe

Wie ersichtlich, sind die Werte für die verschiedenen Testläufe je Test-Setup sehr konstant.

Die Spannweite je Test-Setup reicht von 0,10 bis 1,18 Transaktionen/s, wobei mit höherem Gesamtdurchsatz die Spannweite etwas zunimmt.

5. Schlussfolgerungen und Ausblick

Mit der vorliegenden Arbeit konnte gezeigt werden, dass sich die Methodik von Giuliani et al. (2013) auch für Performancetests von Implemtierungen des WFS-Nachfolge-Standards OGC API – Features anwenden lässt.

Da der Polygon-Testdatensatz GLWD-2 und die verwendeten BBOXes auch heute noch verfügbar sind, konnten die Performancetests mit den gleichen Daten wie damals durchgeführt werden.

Durch diese Kontinuität konnte ein WFS-Testszenario von damals mit einer aktuellen GeoServer-Version auf aktueller Hardware durchgeführt werden. Hier zeigte sich eine starke Performanceverbesserung im Vergleich zu den Ergebnissen von Giuliani et al. (2013).

Die Frage, ob WFS oder OGC API - Features performanter ist, lässt sich nicht pauschal beantworten. Es kommt auf die konkrete Implementierung an.

Beim QGIS Server konnte eine bessere Performance der OGC API – Features um ca. 80% im Vergleich zum WFS festgestellt werden.

Beim GeoServer sieht das anders aus. Hier ist der WFS den OGC API – Features hinsichtlich der Performance im gegebenen Testszenario noch überlegen. Allerdings handelt es sich hier noch um eine experimentelle Implementierung, so dass abzuwarten ist, wie sich die Performance verhält, sollte dieser neue OGC-Standard zukünftig ausgereift im GeoServer-Core verfügbar sein.

Durch die Verwendung eines Cloud-Servers konnte gezeigt werden, dass sich eine Testumgebung für Performancetests von Geodatendiensten relativ kostengünstig ohne Verfügbarkeit von eigener Serverhardware einrichten und betreiben lässt.

Hier konnten unkompliziert zwei unterschiedliche Hardwareausstattungen für die Testläufe verwendet werden. Wobei die, hinsichtlich Arbeitsspeicher und Anzahl vCPUs, doppelt so leistungsstarke Hardwareausstattung nur bei den OGC API – Features auf Basis des QGIS Server zu einem kleinen Performancegewinn führte.

Ob die These, dass bei vielen gleichzeitigen Zugriffen (im vorgestellten Szenario 16 Threads) die Bandbreite der limitierende Faktor ist, könnte in einer zukünftigen Arbeit untersucht werden. Sollte dies der Fall sein, dann würde sich eine stärkere Hardwareausstattung ggf. lohnen.

Ansonsten gilt weiterhin, was teilweise bereits auch Giuliani et al. (2013) als mögliche zukünftige Forschungsthemen benannten, wie z.B.

- Speicherung der Testdaten in anderen Datenformaten (z.B. GeoPackage, FlatGeobuf, GeoJSON) oder Datenbanken (z.B. PostgreSQL mit PostGIS)
- Verwendung von anderen Koordinatenbezugssystemen mit „on-the-fly“-Reprojektion
- Verwendung von anderen Dienstetypen / OGC-Standards
- Verwendung von anderen Implementierungen von OGC API – Features (z.B. deegree oder UMN Mapserver)
- Verwendung von anderer Serverkonfiguration (z.B. QGIS Server Umgebungsvariablen wie `QGIS_SERVER_FORCE_READONLY_LAYERS`)

Zudem führte die Arbeit dazu, dass ein mehr als 12 Jahre alter Fehler in der FOSS-Bibliothek Shapelib hinsichtlich des räumlichen Index einer Shapedatei identifiziert und behoben werden konnte.

Hier könnte zukünftig untersucht werden, wie stark sich die Verfügbarkeit eines räumlichen Index auf die Performance der von QGIS Server bereitgestellten Dienste auswirkt und ob es Performanceunterschiede zwischen dem ESRI-Shapefile-Index (.sbn / .sbx) und dem offenen Quad-Tree-Index (.qix) gibt.

Literaturverzeichnis

Giuliani, G., Dubois, A., & Lacroix, P. (2013). Testing OGC Web Feature and Coverage Service performance: Towards efficient delivery of geospatial data. *Journal of Spatial Information Science*, (7), 1–23. https://doi.org/10.5311/JOSIS.2013.7.112

Hobona, G., Simmons, S., Masó-Pau, J., & Jacovella-St-Louis, J. (2023). OGC API Standards for the Next Generation of Web Mapping. *Abstracts of the ICA*, *6*, 1–2. https://doi.org/10.5194/ica-abs-6-91-2023

Lehner, B., & Döll, P. (2004). Development and validation of a global database of lakes, reservoirs and wetlands. *Journal of Hydrology*, *296*(1–4), 1–22. https://doi.org/10.1016/j.jhydrol.2004.03.028

Portele, C., Vretanos, P. (Peter) A., & Heazel, C. (2022). OGC API - Features - Part 1: Core corrigendum. OGC 17-069r4, Open Geospatial Consortium. http://www.opengis.net/doc/IS/ogcapi-features-1/1.0.1

Rosas-Chavoya, M., Gallardo-Salazar, J. L., López-Serrano, P. M., Alcántara-Concepción, P. C., & León-Miranda, A. K. (2022). QGIS a constantly growing free and open-source geospatial software contributing to scientific development. *Cuadernos de Investigación Geográfica*, *48*(1), 197–213. https://doi.org/10.18172/cig.5143

Seip, C. (2015). Evaluation and Monitoring of Service Quality illustrated by the Example of the German Marine Spatial Data Infrastructure (MDI-DE). *Photogrammetrie - Fernerkundung - Geoinformation*, *2015*(4), 313–329. https://doi.org/10.1127/pfg/2015/0272

Anhang

A Bounding Boxes zur Verwendung für den BBOX-Parameter

minX,minY,maxX,maxY

60,29,75,38

12,-18,24,-4

19,40,21,43

-74,-55,-54,-22

43,39,47,41

45,38,52,43

29,-4,31,-2

16,43,20,45

-109,-56,-66,-17

74,18,135,54

12,-13,31,5

11,-5,19,4

-82,-4,-67,13

-85,20,-74,23

32,35,35,36

42,11,43,13

8,55,15,58

-9,19,12,37

-92,-5,-75,2

25,22,36,32

```
36,12,43,18
-17,21,-9,28
33,3,48,15
-61,-53,-58,-51
-9,49,2,61
-9,49,2,61
40,41,47,44
-17,11,-14,13
19,35,30,42
13,42,19,47
68,7,97,33
44,25,63,40
39,29,49,37
34,29,36,33
35,29,39,33
69,39,80,43
102,10,108,15
125,33,131,39
47,29,48,30
100,14,108,23
35,33,37,35
9,20,25,33
80,6,82,10
-13,28,-1,36
27,45,30,48
```

20,41,23,42

-12,10,4,25

92,10,101,29

18,42,20,44

30,-27,41,-10

-17,15,-5,27

12,-29,25,-17

0,12,16,24

-88,11,-83,15

80,26,88,30

52,17,60,26

61,24,75,37

-81,-18,-69,0

117,5,127,21

124,38,131,43

29,-3,31,-1

22,3,39,23

-18,12,-11,17

-13,7,-10,10

41,-2,51,12

19,42,23,46

36,32,42,37

13,7,24,23

97,6,106,20

67,37,75,41

```
26,36,45,42

119,22,122,25

30,-1,35,4

56,37,73,46

-73,1,-60,12

102,8,109,23

42,12,55,19

22,-18,34,-8

25,-22,33,-16
```

B Skript run.sh zur Ausführung des Performancetests

```
#!/bin/bash


# ./run.sh test_plan.jmx serverinfos


START=$(date +%s)

TIMESTAMP=$(date +%Y%m%dT%H%M%S)

TEST_PLAN=$1

TEST_PLAN_NAME="${TEST_PLAN%.*}"

SERVER_INFOS=$2

WORKING_DIRECTORY=`pwd`

JMETER_DIRECTORY=/home/user/dev/jmeter/apache-jmeter-5.6.3/bin


echo TIMESTAMP: $TIMESTAMP

echo TEST_PLAN: $TEST_PLAN

echo SERVER_INFOS: $SERVER_INFOS

echo WORKING_DIRECTORY: $WORKING_DIRECTORY

echo JMETER_DIRECTORY: $JMETER_DIRECTORY


RESULT_DIRECTORY=$WORKING_DIRECTORY/"$TEST_PLAN_NAME"_"$TIMESTAMP"_"$SERV
ER_INFOS"

OUTPUT_FILENAME=$RESULT_DIRECTORY/"$TEST_PLAN_NAME"_"$TIMESTAMP"_"$SERVER
_INFOS"
```

```
mkdir -p "$RESULT_DIRECTORY"/report


iperf3 -c <server> -> $RESULT_DIRECTORY/bandwidth_upload_before_test.txt

iperf3 -c <server> -R ->
$RESULT_DIRECTORY/bandwidth_download_before_test.txt


$JMETER_DIRECTORY/jmeter -
Jjmeter.reportgenerator.overall_granularity=10000 -n -t $TEST_PLAN -j
$OUTPUT_FILENAME.log -l $OUTPUT_FILENAME.jtl -e -o
$RESULT_DIRECTORY/report


iperf3 -c <server> -> $RESULT_DIRECTORY/bandwidth_upload_after_test.txt

iperf3 -c <server> -R ->
$RESULT_DIRECTORY/bandwidth_download_after_test.txt


END=$(date +%s)

EXECUTION_TIME=$((END- START))


HOURS=$((EXECUTION_TIME / 3600))

MINUTES=$(( (EXECUTION_TIME % 3600) / 60 ))

SECONDS=$((EXECUTION_TIME % 60))


echo "Execution time: $HOURS hours, $MINUTES minutes, and $SECONDS
seconds"
```