Vergleich von PostGIS und Rasdaman als Geodatenbanken für großvolumige Bilddatenbestände eines mobilen Mappingsystems

Comparing PostGIS and Rasdaman for Storing Large-size Georeferenced Images of a Mobile Mapping System

Nicole Hein[1], Jörg Blankenbach[1]

[1]Geodätisches Institut und Lehrstuhl für Bauinformatik & Geoinformationssysteme, RWTH Aachen University · nicole.hein@gia.rwth-aachen.de

Zusammenfassung: Das Ziel des Projektes *RiverView* ist die Entwicklung eines unbemannten Messkatamarans für das Monitoring von kleinen und mittleren Fließgewässern inklusive eines räumlichen Informationssystems für die Datenverwaltung. In diesem Beitrag wird RiverView vorgestellt und insbesondere auf die Entwicklung des Geodatenbankmanagementsystems für die zeitlich und räumlich referenzierte Persistierung von Bilddaten eingegangen. Dabei werden vor allem die beiden Datenbankmanagementsysteme *PostgreSQL* mit den Erweiterungen *PostGIS* und *PostGIS Raster* sowie *Rasdaman* betrachtet. Vorgestellt werden Untersuchungen zur Effizienz des Im- und Exports von Bilddaten sowie deren Bewertung.

Schlüsselwörter: RiverView, PostGIS, Rasdaman

Abstract: *The goal of the project RiverView is the development of an unmanned boat for monitoring of small and midsize flowing waters including the development of a spatial information system for data management. In this paper RiverView is introduced and a special focus is set on the development of the geo database management system for temporally and spatially referenced storage of image data. In particular both the PostgreSQL with its extension PostGIS and PostGIS Raster and the Rasdaman approach are examined. Studies on efficiency for import and export times and its ratings are presented.*

Keywords: *RiverView, PostGIS, Rasdaman*

1 Einführung

Das Projekt *RiverView – Gewässerzustandsbezogenes Monitoring und Management* ist Teil der BMBF-Fördermaßnahme „Regionales Wasserressourcen-Management für den nachhaltigen Gewässerschutz in Deutschland (ReWaM)“, die wiederum unter dem Dach des Förderschwerpunkts „Nachhaltiges Wassermanagement (NaWaM)“ betrieben wird.

Im Rahmen von *RiverView* wird die Grundlage für ein neues, ganzheitliches Monitoring von mittleren und kleinen Fließgewässern sowie ein räumliches Informationssystem geschaffen (Engels et al., 2016). Das Monitoring beruht auf einem unbemannten Messboot (Katamaran), das mit diverser Sensorik für die Erfassung der Verhältnisse Über- und Unterwasser ausgestattet ist. Für die Überwasserdatenerfassung wird eine spezielle Mappingplattform entwickelt, die mit einem omnidirektionalen Kamerasystem als primären Erfassungssensor ausgestattet ist (Effkemann et al., 2017). Die aus sechs Einzelkameras bestehende Panoramakamera ist in der Lage, kontinuierlich Bilder und Videos aufzuzeichnen. Auch Unterwasser werden Gewässer- und Bilddaten, die orts- und zeitbezogen sind, gesammelt und später in

AGIT – Journal für Angewandte Geoinformatik, 3-2017, S. 2-12. © Wichmann Verlag, VDE VERLAG GMBH · Berlin · Offenbach. ISBN 978-3-87907-633-8, ISSN 2364-9283, eISSN 2509-713X, doi:10.14627/537633001.
Dieser Beitrag ist ein Open-Access-Beitrag, der unter den Bedingungen und unter den Auflagen der Creative Commons Attribution Lizenz verbreitet wird (http://creativecommons.org/licenses/by-nd/4.0/).

einer Geodatenbank gespeichert. Durch Schnittstellen werden diese Daten über ein räumliches Informationssystem unterschiedlichen Nutzergruppen zur Verfügung gestellt. Aufgrund der Auswertung der simultan erfassten Über- und Unterwasserdaten wird dieses Projekt zu einem besseren Verständnis von komplexen Prozessen in der Gewässerbewirtschaftung beitragen. Die Sensoren des Trägersystems sowie die Analyse der heterogenen Daten, die während einer Messfahrt erfasst werden, stellen eine neuartige Lösung für ein nachhaltiges Wasserressourcen-Management dar.

Der Aufbau des Projekts sowie die Beschreibung der Projekt-Komponenten kann aus RiverViewWeb (2015) sowie Lanuv (2012) entnommen werden.

2 Geodatenverwaltung in RiverView

Im Zusammenhang mit *RiverView* obliegt dem Geodätischen Institut der RWTH Aachen u. a. der Aufbau eines Geodatenbankmanagementsystems mit Bezug zu Raum und Zeit. In den folgenden Unterabschnitten werden zwei Datenbankmanagementsysteme für diesen Zweck evaluiert: *PostgreSQL* und *Rasdaman*. Die Betrachtung findet unter dem Gesichtspunkt statt, ob diese als Datenhaltungskomponente für das Projekt *RiverView* geeignet sind.

2.1 Datenbankmanagementsysteme (DBMS)

Aus der Vielzahl der vorliegenden Datenbankmanagementsysteme wurde eine Auswahl von infrage kommenden Produkten für die effiziente Speicherung der Daten getroffen. Nach der Evaluierung möglicher Systeme, wurden zwei DBMS ausgewählt und genauer untersucht.

PostgreSQL

Ein mittlerweile weitverbreitetes DBMS für die Speicherung von Geodaten ist das objektrelationale System *PostgreSQL* (PostgreSQL, 2016)) mit der Abfragesprache *SQL*, das zurzeit in der Version 9.6.2 (09.02.2017) vorliegt und der *PostgreSQL Lizenz* (vergleichbar mit BSD-Lizenz) unterliegt. Mit der Erweiterung *PostGIS* (PostGIS, 2016) ist es möglich, mit Geometrien und räumlichen Referenzsystemen bzw. Funktionen zu arbeiten. *PostGIS* liegt in der Version 2.3.2 (31.01.2017) vor und verwendet die Schnittstelle *Generalized Search Tree* (*GiST*), womit eine eigenständige Definition von Sortier- und Suchverfahren ermöglicht wird. *PostGIS Raster* ist eine Ergänzung zur *PostGIS*-Erweiterung und hält die Möglichkeit zur persistenten Speicherung von Bild- und Rasterdaten bereit. Weiterführende Informationen sind in Raster (2016) zu finden.

In der *PostgreSQL*-Datenbank entsteht eine Relation, die in jeder Zeile einen Teil des Rasters oder ein Raster-Objekt enthält. Die Bildkacheln (entstanden durch Partitionierungsmechanismen) bzw. Raster-Objekte stehen in der Spalte des Typs *raster*. Die Spalte *rid* bezeichnet die ID des Raster-Objekts oder der Kachel und wird sequenziell (mit einer eigens dafür angelegten Sequenz) inkrementiert. Zusätzlich kann eine weitere Spalte mit dem Dateinamen des Rasters hinzugefügt werden. Jedes Raster-Teil bzw. -Objekt hat eine Pixelgröße, eine Breite und eine Höhe. Des Weiteren besteht es aus einem oder mehreren Bändern bzw. Kanälen, die wiederum eine Ansammlung von Pixeln enthält. Es hat diesbezüglich eine variable An-

zahl von Bändern sowie einen Pixeltyp pro Band. Zusätzliche Werte, die enthalten sind, sollen hier nicht näher betrachtet werden (Raster, 2016). Weiterführende Informationen hierzu sind unter PostGIS9 (2016) zu finden.

Rasdaman

Rasdaman (engl. raster data manager) (Rasdaman, 2016) ist eine Rasterdatenbank, die speziell darauf ausgerichtet ist, große Rasterdatenbestände (z. B. Bilddaten) effizient zu speichern. Das DBMS ist skalierbar und weist eine hohe Flexibilität mit zugleich großer Performanz auf. Diese wird unter anderem durch den Aufbau des DBMS, aber auch durch die entwickelte RasQL-Abfragesprache erreicht.

Das *Rasdaman*-System ist in einer Client-Server-Architektur verwirklicht, wobei das (relationale) DBMS mit der *Rasdaman*-Komponente serverseitig implementiert ist und somit die Anfrageverarbeitung komplett auf dem Server stattfindet. Die multidimensionalen Rasterdaten (Bilddaten, MDD) und deren Metadaten werden durch eine Anwendung via *SQL* oder einen Web-Browser vom Client an den Server übertragen. Die Besonderheit ist, dass *Rasdaman Community* mit einem relationalem DBMS, *PostgreSQL* oder *SQLite*, arbeitet, um die Daten zu persistieren. Der Unterschied liegt in der Speicherung der Bilddaten: Wird *SQLite* als zugrunde liegende Datenbank genutzt, so werden lediglich die Metadaten in diese eingespeist. Die entstandenen Kacheln der Bilddatei werden im Dateisystem abgelegt. Die objektrelationale Datenbank *PostgreSQL* wird für *Rasdaman* genutzt, wenn die Kacheln der Bilddaten in *PostgreSQL BLOB*s gespeichert werden sollen (Store, 2016).

Um Anfragen von anderen Systeminstanzen zu verarbeiten, existiert die *Petascope*-Komponente im *Rasdaman*-System, die Anfragen von visuellen oder Kommandozeilen-Clients entgegennimmt und durch einen Parser an die *Rasdaman*-Funktionseinheit weiterleitet. Raster-Objekte werden in Kacheln zerlegt und in eine relationale Datenbank importiert. Mithilfe des INSERT-Statements können verschiedene Partitionierungen der Raster realisiert werden; es werden keine, regelmäßige oder unregelmäßiger Partitionierungen unterschieden. In Baumann et al. (1997) wird darüber berichtet, dass eine willkürliche Partitionierung in rechteckige Kacheln für MDDs am günstigsten ist.

Es wird für jedes Objekt in der Datenbank ein räumlicher Index erzeugt, der Informationen zu den partitionierten Teilen des MDDs und deren räumliche Daten enthält. Abfragen werden analysiert, optimiert und dann im *Rasdaman*-Server ausgeführt. Hierfür werden algebraische Semantiken genutzt, die auf AFATL Image Algebra (Ritter et al., 1990) basieren und unterschiedliche Bildtransformationen ausführen können.

Der Zugang zur *Rasdaman*-Datenbank ist mit der Abfrage-Sprache *RasQL* gegeben, die in *RasDL* (*Raster Definition Language*) und *RasML* (*Raster Manipulation Language*) unterteilt ist. Die *Rasdaman*-Entwickler haben sich zum Ziel gesetzt, wohldefinierte Standards einzuhalten; so auch mit *RasQL*, die im objektorientierten DBMS-Standard *ODMG* (*Object Data Management Group*) eingebettet ist.

Die Raster werden in der *Rasdaman*-Datenbank in sog. Collections gruppiert. Diese sind vergleichbar mit Tabellen eines relationalen DBMS. Durch MDD-Typ-Definition ist jedes MDD nur einer Collection zugeordnet (Baumann et al., 1997).

Zusammenfassende Betrachtung der DBMS

In Tabelle 1 sind die beiden DBMS hinsichtlich der für das Projekt relevanten Spezifikationen vergleichend dargestellt.

Beide Systeme können durch die Betriebssysteme *Linux* und *Windows* (*Rasdaman Community* ermöglicht durch den *Petascope-Client* einen Zugriff von *Windows* auf die Datenbank) ausgeführt werden. *RasQL* ist eine eigens für *Rasdaman* entwickelte Sprache, die sich stark an *SQL* orientiert und somit in der Lage ist, Daten in der Datenbank abzufragen, zu manipulieren und zu definieren (Java, 2016).

Tabelle 1: DBMS im Vergleich

Eigenschaften	PostgreSQL	Rasdaman
Version	9.6.2 (09.02.2017)	9.2.1 (17.02.2016)
System	Linux und Windows	Linux (Windows-Clients)
Sprache	SQL	RasQL
Speicherung von Vektordaten	PostGIS	Direkt
Speicherung von Bilddaten	PostGIS Raster	Direkt

Vektordaten können in beide Systeme eingespeist werden, allerdings wird bei *PostgreSQL* eine Erweiterung benötigt, die eine Persistierung von Geometrien ermöglicht. Wichtig für das Projekt *RiverView* ist die Speicherung und Verwaltung von georeferenzierten und großvolumigen Bilddatenbeständen. Diese können mithilfe von *PostGIS Raster* in *PostgreSQL* und in *Rasdaman* direkt importiert werden.

Die *PostgreSQL*-Datenbank kann unbeschränkt groß werden, hat jedoch mit 32 TB eine Beschränkung in der Tabellengröße sowie mit 1,6 TB eine Beschränkung in der Zeilengröße. Ein Feld in *PostgreSQL* kann eine Größe von maximal 1 TB haben. In Yu et al. (2016) wird dargelegt, dass keine Begrenzung der Größe von *Rasdaman* existiert, wenn es als zentrales DBMS für Rasterdaten eingesetzt wird.

2.2 Evaluierung der DBMS für RiverView

Für die Evaluierung der beiden Systeme wurden zwei Eigenschaften in den Mittelpunkt gerückt: Daten, unabhängig von der Art, sollen konsistent und effizient in der Datenbank persistiert werden. Effizienz bedeutet in diesem Zusammenhang, dass das DBMS eine hohe Performanz aufweist, wenn große Datenmengen in der Datenbank abgelegt bzw. abgerufen werden (Web-Anwendungen). Damit wird eine gute Indizierung der Daten vorausgesetzt.

Eigenschaften von PostgreSQL und Rasdaman

Dieser Abschnitt beschäftigt sich mit den Eigenschaften der zwei genannten DBMS *PostgreSQL* und *Rasdaman* im Hinblick auf die Verwendung im Rahmen des Projekts *RiverView*. Aufgrund der Spezifizierung von *RiverView* muss im Unterschied zum Abfragen der Daten, der Import prinzipiell nicht performant sein. Vorgesehen als Client für *RiverView* ist eine Webanwendung, die hohe Latenzzeiten bei der Datenabfrage vermeidet und später durch eine Augmented- bzw. Virtual-Reality-Anwendung ergänzt wird. Als weitere Möglichkeit der Datenbereitstellung wird ein Desktop- oder Web-GIS angebunden. Wenn das DBMS bereits

diesbezüglich Schnittstellen implementiert hat, besteht die Möglichkeit, die über das GIS abgefragten Geodaten aus der Datenbank zu laden und darzustellen.

PostgreSQL

Wird die Erweiterung *PostGIS* und deren Ergänzung *PostGIS Raster* in die Betrachtungen einbezogen, so kann für das objektrelationale DBMS *PostgreSQL* festgehalten werden, dass sowohl skalare und vektorielle Sensordaten als auch komplexe Bilddaten (Rasterdaten mit räumlichem Bezug) gespeichert werden können. Der zeitliche Bezug kann entweder durch eine Komponente *Zeit* oder durch einen benutzerdefinierten spatiotemporalen Datentyp (Raza, 2012) in das DBMS einfließen. *PostGIS* setzt zudem die Simple-Feature-Access-Spezifikation des OGC um. Zudem ist es in der Lage, u. a. die Formate *GeoTIFF*, *PNG*, *JPEG*, und *NetCDF* mit der Datenbanksprache *SQL* zu verwalten. Vektordaten können mithilfe von *KML*, *GML*, *GeoJSON*, *GeoHash* und *WKT* sowie durch *SQL*-Befehle importiert werden (Feat, 2016).

Rasdaman

Rasdaman wurde speziell für die Anforderungen großvolumiger spatiotemporaler Rasterdaten entwickelt. Das System ist OGC-konform und unterstützt alle oben genannten Web Services. *Rasdaman* nutzt die objektrelationale Datenbank *PostgreSQL* in der Version 9.4.9 zur Verwaltung der Geometriedaten (OGC-Service-Schnittstelle) sowie für den Einsatz von Petascope. Die Daten werden allerdings im Dateisystem abgelegt. Die Metadaten werden in die relationalen *SQLite*-Datenbank der Version 3.8.7.1 importiert (Store, 2016).

Testsystem und Methoden

Bei den Testsystemen handelt es sich jeweils um ein *Linux* mit *Debian*-Distribution in der Version 8.6 (*jessie*). Die beiden DBMS (*PostgreSQL* und *Rasdaman*) wurden auf getrennten Server-Instanzen installiert und getestet. Geprüft werden sollte vor allem, wie sich die beiden ausgewählten Managementsysteme verhalten, wenn großvolumige spatiotemporale Bilddaten gespeichert werden.

PostgreSQL ist in Verbindung mit *PostGIS Raster* in der Lage, spatiotemporale Bilddaten persistent zu speichern. Zusätzlich existiert die Möglichkeit, Bilddaten als sogenannte BLOBs (Binary Large Objects) in der Datenbank abzulegen. Darüber hinaus ist es wie in *Rasdaman* möglich, die Bilddaten im Dateisystem abzulegen und deren Metadaten in *PostgreSQL* zu importieren. Tabelle 2 stellt die einzelnen Methoden und ihre Vor- bzw. Nachteile zusammen, unabhängig von der noch zu untersuchenden Effizienz.

Tabelle 2: Vor- und Nachteile der Methoden für das *PostgreSQL*-DBMS

Methode	Vorteile	Nachteile
PostGIS Raster	Rasterdatenspeicherung mit Geometrie	–
BLOB	Speicherung von Rasterdaten mit Geometrie-Metadaten möglich	Speicherung von Geometrien nur mithilfe von Metadaten (World-Datei) möglich
Dateisystem	Bilddatenspeicherung mit Geometrie-Metadaten	Datensicherheit und Integrität; Speicherung von Geometrien nur mithilfe von Metadaten möglich (World-Datei)

Aufgrund der in der Tabelle genannten Nachteile, werden die Methoden ‚BLOB' und ‚Dateisystem' nicht weiter betrachtet.

Methoden

Da nicht bei allen der oben genannten Methoden eine Georeferenzierung möglich ist oder nur über Umwege realisiert werden kann, werden in dieser Arbeit die Im- und Exportperformanz von *PostGIS Raster* und *Rasdaman* untersucht. Da in beiden Fällen eine Client-Server-Architektur vorliegt, wurden die vorliegenden, weiter unten detailliert beschriebenen Bilddaten mithilfe eines in der Entwicklungsumgebung Eclipse in der Programmiersprache Java eigens implementierten Konsolenprogramms in das jeweilige DBMS im- und exportiert. Die Zeitmessung erfolgte jeweils nur für das ‚reine' Im- bzw. Exportieren mithilfe der Systemzeit. Für die *Rasdaman*-Datenbank muss als Besonderheit erwähnt werden, dass der Import über *cURL* realisiert wurde. Somit konnte nur mithilfe von *Petascope* eine Verbindung zu *Rasdaman* hergestellt und demzufolge dann mittels *cURL* die Bilddatei zum Server übertragen werden. Der Export der Bilddaten aus dem *Rasdaman*-DBMS ist ebenfalls mit *Petascope* durchgeführt worden. Die Versuche wurden in mehreren Durchläufen durchgeführt. Dabei wurden auf die gleiche Art und Weise Referenzmessungen vorgenommen, welche die Wiederholbarkeit der Messergebnisse verifizieren sollen. Bei den Bilddaten, die in beide DBMS importiert wurden, handelt es sich um ein nicht-georeferenziertes JPG-komprimiertes Panoramabild aus den Aufzeichnungen der *LadyBug*-Kamera des *RiverView*-Systems, das eine Auflösung von 8.000 × 4.000 Pixel bei einer Bittiefe von 24 aufweist. Mithilfe des *LadyBug SDK* wurden die sechs Einzelbildaufnahmen zu einem Panorama zusammengefügt, wobei die Dateigröße 4,76 MB umfasste. Des Weiteren wurden zwei georeferenzierte GeoTIFF-Bilder in die Managementsysteme mit einer Auflösung von 2.510 × 1.253 Pixel bzw. 2.216 × 1.419 Pixel eingelesen. Der Speicherbedarf der beiden GeoTIFF-Bilddateien betrug 4,1 MB bzw. 2,5 MB.

Untersuchung 1: In dieser Untersuchung wurde die Effizienz des Imports in das DBMS ermittelt. Für jedes Bild wurde ein einzelner Versuch unternommen, der jeweils 100 Durchläufe enthält. In jedem Durchlauf wurde das Bild ein weiteres Mal in die Datenbank importiert. Die gleiche Vorgehensweise wurde bei der Referenzmessung vorgenommen. Zu jedem Import einer Bilddatei wurde eine Zeitmessung vorgenommen. Caching wurde in dieser Versuchsreihe nicht berücksichtigt, da nach jedem der Durchläufe die Verbindung zum PostgreSQL-Server abgebaut und erst mit dem nächsten Durchlauf wieder aufgebaut wurde. Der Import erfolgte durch folgendes Kommando: *raster2pgsql.exe -I -C -x -e -s 4326 -t 128x128 filepath tableName | psql.exe -U user -d base -h host -p port*. Am Ende sind dementsprechend drei Versuche zu je 100 Durchläufen und einer zusätzlichen Referenzmessung allein für das DBMS *PostgreSQL* durchgeführt worden. Auch für das Managementsystem *Rasdaman* wurden insgesamt 600 Durchläufe (drei Bilddatei und Referenzmessungen) ausgeführt.

Untersuchung 2: Eine weitere Untersuchung ermittelt Effizienz des Exports aus den Managementsystemen. Die SELECT-Abfrage wurde direkt nach jedem Import in die Datenbank vorgenommen. Für jeden Export wurde die Zeit gemessen. Auch hier wurde Caching nicht berücksichtigt. Durch folgende Abfrage wurden die Bilddaten exportiert:
SELECT ST_AsPNG(rast) as rastpng FROM tableName.

Ergebnisse

Die Abbildungen 1a bis 1c dokumentieren die Geschwindigkeitsmessungen beim Import in die DBMS *PostgreSQL* und *Rasdaman* grafisch. Auf der Ordinate ist die Zeit *t* in Minuten und auf der Abszisse die Nummer des Durchlaufs aufgetragen. In Abbildung 2 sind die entsprechenden Messdaten für die Exportvorgänge aus der Datenbank *PostgreSQL* und *Rasdaman* gegenübergestellt.

Abbildung 1a zeigt die Importgeschwindigkeit des Panoramabildes in das *Rasdaman*- bzw. *PostgreSQL*-System. Bei dem Versuch zu *PostgreSQL PostGIS Raster* sind zwischen den Durchläufen 69 und 74 zwei Ausreißer zu beobachten, die in der Referenzmessung nicht zu sehen sind. Die Schwankungsgrenzen zwischen Referenz- und Originalmessung liegen bei Durchlauf 100 zwischen ca. 67 und 76 Minuten. Die etwas unter der Erstmessung liegende Referenz deutet auf eine geringere Serverauslastung zum Zeitpunkt der Messung hin. Der Vergleich zwischen der *Rasdaman*- und der *PostGIS*-Messung zeigt eine große Diskrepanz. Während die Importzeit für *PostGIS Raster* sehr schnell in den zweistelligen Minutenbereich steigt, liegt die Zeit für den Import für *Rasdaman* konstant zwischen 4.996 und 6.771 Millisekunden (ms). Am Verlauf des Graphen von *PostGIS Raster* wird zudem deutlich, dass die Importzeit in die Datenbank überproportional größer wird, je mehr Bilder in das DBMS geladen werden.

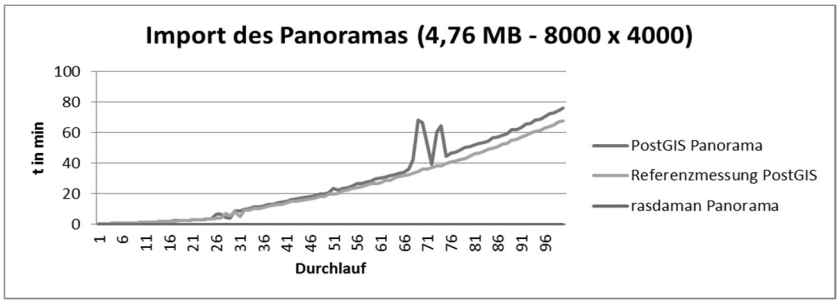

Abb. 1a: Import Panoramabild

Abbildung 1b zeigt das Zeitverhalten für *Rasdaman* und *PostGIS Raster*, wenn ein mittelgroßes georeferenziertes Bild (4,1 MB) in die Datenbank eingespeist wird. Es zeigt sich ein ähnliches Bild wie in der Abbildung zuvor. Die Referenzmessung zu *PostGIS Raster* bewegt sich in einem vergleichbaren Rahmen wie die Erstmessung. Auffällig sind dabei die Ausschläge ab Durchlauf 86, die – belegt durch die Referenzmessung – wiederholbar sind. Die Zeiten für den Import in die *Rasdaman*-Datenbank liegen zwischen 650 und 2.199 ms. Somit ist *Rasdaman* um ein Vielfaches schneller als der Import mit *PostGIS Raster*, der im Durchschnitt 3,7 Minuten dauert.

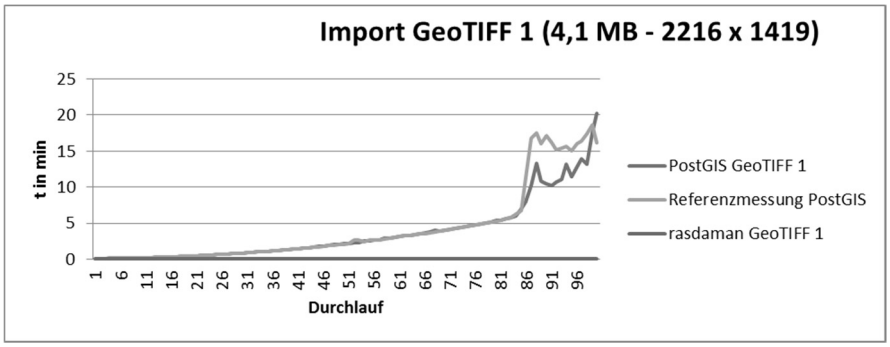

Abb. 1b: Import GeoTIFF-Bild 1

Ein ähnliches Bild zeigt sich in Abbildung 1c, in der die Importzeit von *PostGIS Raster* und *Rasdaman* für die kleinere GeoTIFF-Datei (2,5 MB) dargestellt wird. Referenz- und Erstmessung liegen hier sehr nahe beieinander. Die *Rasdaman*-Importzeit liegt zwischen 473 und 2.604 ms und unterliegt somit – relativ gesehen – teilweise starken Schwankungen, was aber auf die Serverauslastung zurückzuführen ist. Mit maximal 2.604 ms für den Import einer Bilddatei ist der *Rasdaman*-Import im Vergleich zu *PostGIS Raster* mit 2,8 min deutlich schneller.

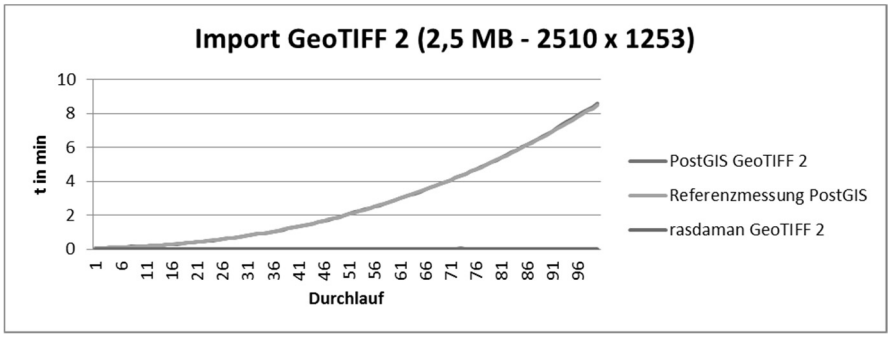

Abb. 1c: Import GeoTIFF-Bild 2

Abbildung 2 zeigt die Messdaten für die Exportvorgänge aus den Datenbankmanagementsystemen *Rasdaman* und *PostgreSQL* mit *PostGIS Raster*. Die gestrichelten Linien zeigen jeweils die Exportzeiten des Panoramabilds. Es wird demnach deutlich mehr Zeit benötigt, um Bilddaten aus der *PostgreSQL*-Datenbank mit *PostGIS Raster* zu exportieren als mit *Rasdaman*. Das gleiche Verhalten ist bei den anderen Bilddaten zu beobachten: *Rasdaman* ist um ein Vielfaches schneller als *PostGIS Raster*. Die gepunkteten Linien zeigen die Exportzeit der GeoTIFF-Bilddatei 1, wohingegen die Werte für das kleinere GeoTIFF-Bild 2 in einer durchgezogenen Linie zu sehen sind.

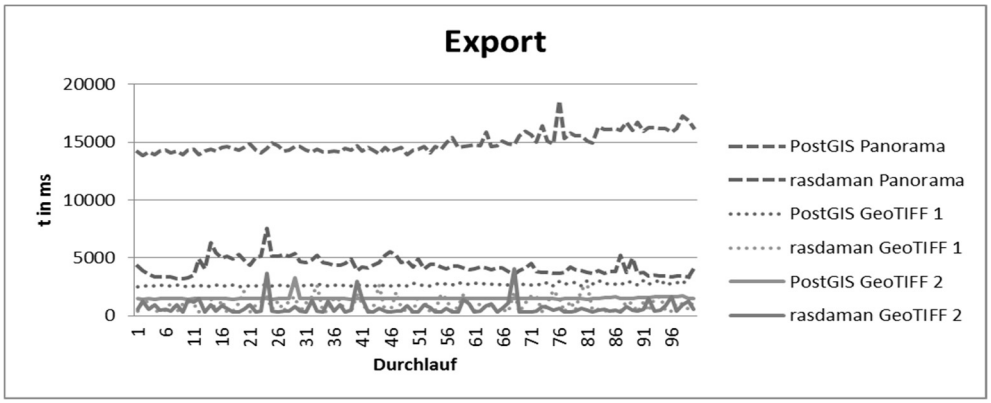

Abb. 2: Export

Anzumerken ist, dass sich die Managementsysteme annähernd konstant bei den Exportzeiten verhalten. Die Schwankungen, die in den Abbildungen zu sehen sind, sind einerseits auf die höhere Auflösung im Diagramm (Angabe der Zeit in Millisekunden) und andererseits auf die Serverbelastung zurückzuführen.

2.3 Zusammenfassung

In beiden untersuchten Datenbankmanagementsystemen werden die Daten korrekt und wiederabrufbar abgelegt. Beide Systeme sind in sich ebenso effizient. Effizienz bedeutet, dass die Daten richtig und wiederabrufbar in das System abgelegt werden können und ebenso, dass diese Daten schnell im- bzw. vor allem exportiert werden können.

Aufgrund der Untersuchungen und deren Ergebnissen kann festgestellt werden, dass das Datenbankmanagementsystem *Rasdaman* erheblich effizienter ist als *PostgreSQL PostGIS Raster*, um große Bilddatenmengen zu im- bzw. exportieren.

3 Fazit und Ausblick

Das Geodatenbankmanagementsystem von *RiverView* muss in der Lage sein, heterogene Daten, d. h. skalare, vektorielle und Rasterdaten, konsistent und effizient zu speichern. Auf dieser Grundlage wurden die beiden DBMS *PostgreSQL* und *Rasdaman* untersucht, die georeferenzierte Daten speichern können und hierbei ggf. auch Big-Data-Anforderungen gerecht werden.

Die beiden untersuchten Datenbankmanagementsysteme unterscheiden sich grundlegend in der Verbreitung und der Intensität ihrer Nutzung in der Praxis. Während *PostgreSQL* eine der meistgenutzten Geodatenbanken ist, ist *Rasdaman* bis vor wenigen Jahren noch fast unbekannt gewesen. *Rasdaman* wird vor allem in der Medizin mit Sensor- und statistischen Daten eingesetzt. Auch im Bereich der Geowissenschaften mit großen Simulations- und Bilddatenbeständen bietet sich das DBMS an, zumal *Rasdaman* OGC-konform ist. Die Untersuchungen in dieser Arbeit zeigen, dass *Rasdaman* eine sehr effiziente und somit sehr schnelle

Datenbank aufbietet, die den Datenverwaltungsanforderungen von *RiverView* gerecht wird. Vor allem die allgemeine Performanz, die Flexibilität, die Skalierbarkeit und der schnelle Zugriff auf Daten sind überzeugend. *PostgreSQL* bietet mit der Erweiterung *PostGIS* und *PostGIS Raster* eine Lösung für Rasterdaten zur georeferenzierten Speicherung. Jedoch ist diese Lösung – zumal in Hinblick auf Big-Data-Anwendungen – womöglich nicht mehr ausreichend, da die Performanz für viele Anwendungen, z. B. Web-Anwendungen, unbefriedigend ist.

Im Projekt *RiverView* werden neben großen Bilddatenbeständen auch weitere skalare und vektorielle Sensordaten verwaltet. Für das Management derartig heterogener Datentypen können zukünftig zusätzlich multimodale Datenbankmodelle genutzt werden. Die auf der Effizienz *Rasdamans* beruhenden Vorteile können so in Verbindung mit anderen DBMS genutzt werden. Hierbei sollen zukünftig auch Lösungen auf der Grundlage von NoSQL in die Betrachtungen einbezogen werden.

Danksagung

Das Projekt *RiverView* wird vom Bundesministerium für Bildung und Forschung (BMBF) mit dem Kennzeichen 033W022 im Rahmen der Fördermaßnahme *ReWaM* im Förderschwerpunkt *NaWaM* gefördert.

Literatur

Baumann, P., Furtado, P., Ritsch, R., & Widmann, N. (1997). *The RasDaMan Approach to Multidimensional Database Management*. München, Germany: Bavarian Research Centre for Knowledge-Based Systems (FORWISS).

Effkemann, C., Schwermann, R., & Blankenbach, J. (2017). Kalibrierung und Navigation eines Überwasser-Mapping-Systems für die Erfassung von bildhaften und sensorischen Gewässerparametern. In: W. Lienhart (Ed.), *Ingenieurvermessung 17. Beiträge zum 18. Internationalen Ingeniervermessungskurs, Graz, 2017* (pp. 113–130). Berlin/Offenbach: Wichmann.

Engels, R., Schüttrumpf, H., Blankenbach, J., & Schwermann, R. (2016). RiverView – Die Digitalisierung der Wasserwirtschaft durch hochaufgelöste raumzeitliche Informationsmodellierung. In: *Tagungsband zur 49. Essener Tagung für Wasser- und Abfallwirtschaft*, 02.-04.03.2016 in Essen.

Feat (2016). *PostGIS – Spatial and Geographic objects for PostgreSQL*. Retrieved Dec 17, 2017, from http://www.postgis.net/features.

Java (2016). *Rasdaman Java Developers Guide, rasdaman Version 9.2*. Retrieved Jan 12, 2017, from http://rasdaman.org/browser/manuals_and_examples/manuals/doc-guides/dev-guide-java.pdf.

LANUV (2012). *LANUV-Arbeitsblatt 18 – Gewässerstruktur in Nordrhein-Westfalen. Kartieranleitung für die kleinen bis großen Fließgewässer*. Recklinghausen: Landesamt für Natur, Umwelt und Verbraucherschutz NRW.

PostGIS (2016). *PostGIS – Spatial and Geographic objects for PostgreSQL*. Retrieved Jan 17, 2017, from http://www.postgis.net.

PostGIS9 (2016). *PostGIS Manual – Chapter 9. Raster Reference*. Retrieved Jan 19, 2017, from http://postgis.net/docs/manual-2.1/RT_reference.html.

PostgreSQL (2016). *PostgreSQL – The world's most advanced open source database*. Retrieved Jan 17, 2017, from http://www.postgresql.org.

Rasdaman (2016). *Rasdaman – Welcome to rasdaman – the world's most flexible and scalable Array Engine*. Retrieved Jan 17, 2017, from http://www.rasdaman.

Raster (2016). *PostGIS Raster Home Page*. Retrieved Jan 19, 2017, from https://trac.osgeo.org/postgis/wiki/WKTRaster.

Raza, A. (2012). Working with Spatio-Temporal Data Type. *International Archives of the Photogrammetry, Remote Sensing and Spatial Information Science, Vol. XXXIX-B2*. XXII ISPRS Congress, Australia.

Ritter, G., Wilson, J., & Davidson, J. (1990). Image Algebra: An Overview. *Computer Vision, Graphics and Image Processing, 49*(3), 297–331.

Store (2016). *Rasdaman Storage Backend*. Retrieved Dec 05, 2016, from http://www.rasdaman.org/wiki/RasdamanStorageBackend.

Yu, S., & Guo, S. (Eds.) (2016). *Big Data Concepts, Theories, and Applications*. Cham, Schweiz: Springer International Publishing. doi:10.1007/978-3-319-27763-9.