Universität Leipzig

Big Data Praktikum

# Konzeptioneller Entwurf

zum Thema:

# Speicherung und Visualisierung Geotemporaler Daten

Oliver Swoboda
und
Sebastian Thamm

**1. OS: Ubuntu 14.04**

Ubuntu (Linux) ist eine kostenlose, auf Debian basierende Linux-Distribution. Die Motivation der Entwickler von Ubuntu ist es, ein stabiles und leicht zu installierendes Betriebssystem zur Verfügung zu stellen, welches einfach zu bedienen und mit aufeinander abgestimmter Software ausgestattet ist.

Eben diese Eigenschaften haben uns dazu bewogen auf diesem Betriebssystem aufbauend unser Praktikum zu bearbeiten. Wir entscheiden uns Version 14.04 zu verwenden, da diese sich im Zusammenspiel mit den benutzten Versionen der folgenden Programme als stabil erwiesen hat und je nach Bedarf die Auswahl zwischen GUI-/Server- und 32Bit/64Bit-Version bietet.

**2. Java 8 (JDK 8)**

Java ist eine objektorientierte Programmiersprache und eine eingetragene Marke des Unternehmens Sun Microsystems. Im Gegensatz zu anderen Sprachen, wie z.B. C, oder C++, bietet Java Plattformunabhängigkeit und ist deswegen eine der meistgenutzten Programmiersprachen weltweit. Da GeoMesa in Scala, und somit als einziges Framework nicht auf Java (bzw. Java Script) basiert, profitiert auch die entstandene Anwendung nicht von dieser Plattformunabhängigkeit. Das Front-End verbleibt jedoch plattformunabhängig.

Wir haben uns hier für die (zur Zeit) aktuellste (stabile) Version des JDK entschieden.

**3. Apache Hadoop (Version 2.2.0)**

Apache Hadoop ist ein freies, in Java geschriebenes Framework, welches dazu dient verteilt arbeitende und skalierbare Anwendungen zu ermöglichen. Es implementiert den MapReduce-Algorithmus und dient zur Ausführung intensiver Rechenprozesse mit großen Datenmengen auf Computerclustern.

Bestandteil von Apache Hadoop ist das **Hadoop Distributed File System (HDFS)**, welches zur Speicherung sehr großer Datenmengen auf den Dateisystemen mehrerer Knoten dient.

Wir benutzen Apache Hadoop zur Speicherung der geotemporalen Daten, welche wir aus den zur Verfügung gestellten Daten des GDELT-Projektes beziehen. Die Anzahl der Knoten und der Grad der Redundanz bleibt frei knfigurierbar.

Die Wahl der Version folgte auf Grund der Anforderungen der gewählten GeoMesa-Version.

**4. Apache ZooKeeper (Version 3.4.6)**

Apache ZooKeeper ist eine Open-Source-API, welche die Konfiguration und Synchronisation verteilter Prozesse auf großen Systemen ermöglicht.

Wir setzen ZooKeeper ein, um für den/die Clients die Konsistenz der Daten zu bewahren und einen Single Point of Failure zu vermeiden.

Die Wahl viel hier auf den (zu diesem Zeitpunkt) aktuellsten Stable-Release, da dieser problemlos auch mit älteren Versionen von Hadoop und Accumulo zusammenarbeitet.

**5. Apache Accumulo (Version 1.5.4)**

Apache Accumulo, ebenfalls eine Open-Source-API, realisiert einen sortierten, verteilten key/value-Store basierend auf der BigTable-Technology von Google. Es ist in Java geschrieben und erweitert Hadoop und ZooKeeper um cell-based Access Control und serverseitige Programmierungs-mechanismen, welche es an mehreren Stellen erlauben Key/Value-Paare im Datei-Management-Prozess zu bearbeiten.

**6. GeoMesa (Version 1.1.0-rc.6)**

GeoMesa ist eine freies, verteiltes Framework, welches einen raum-zeitlichen Index auf einer BigTable-Datenbank verwirklicht. Es ist in Scala verfasst und implementiert den Geohash-Algorithmus.

GeoMesa ist in der Lage Milliarden geometrischer Features aufzunehmen, zu indexieren und abzufragen, indem es ein parallelisiertes Indexschema benutzt. Es implementiert eine Abbildung von Längengrad, Breitengrad und Datum auf einen eindimensionalen, lexikographischen Key-Space.

Besonders ist hierbei vor allem die flexible Anpassung an verschieden große Zeiträume und geographische Maßstäbe, welche GeoMesa sowohl für die Analyse lokaler, wie auch globaler Gegebenheiten gleichermaßen geeignet macht.

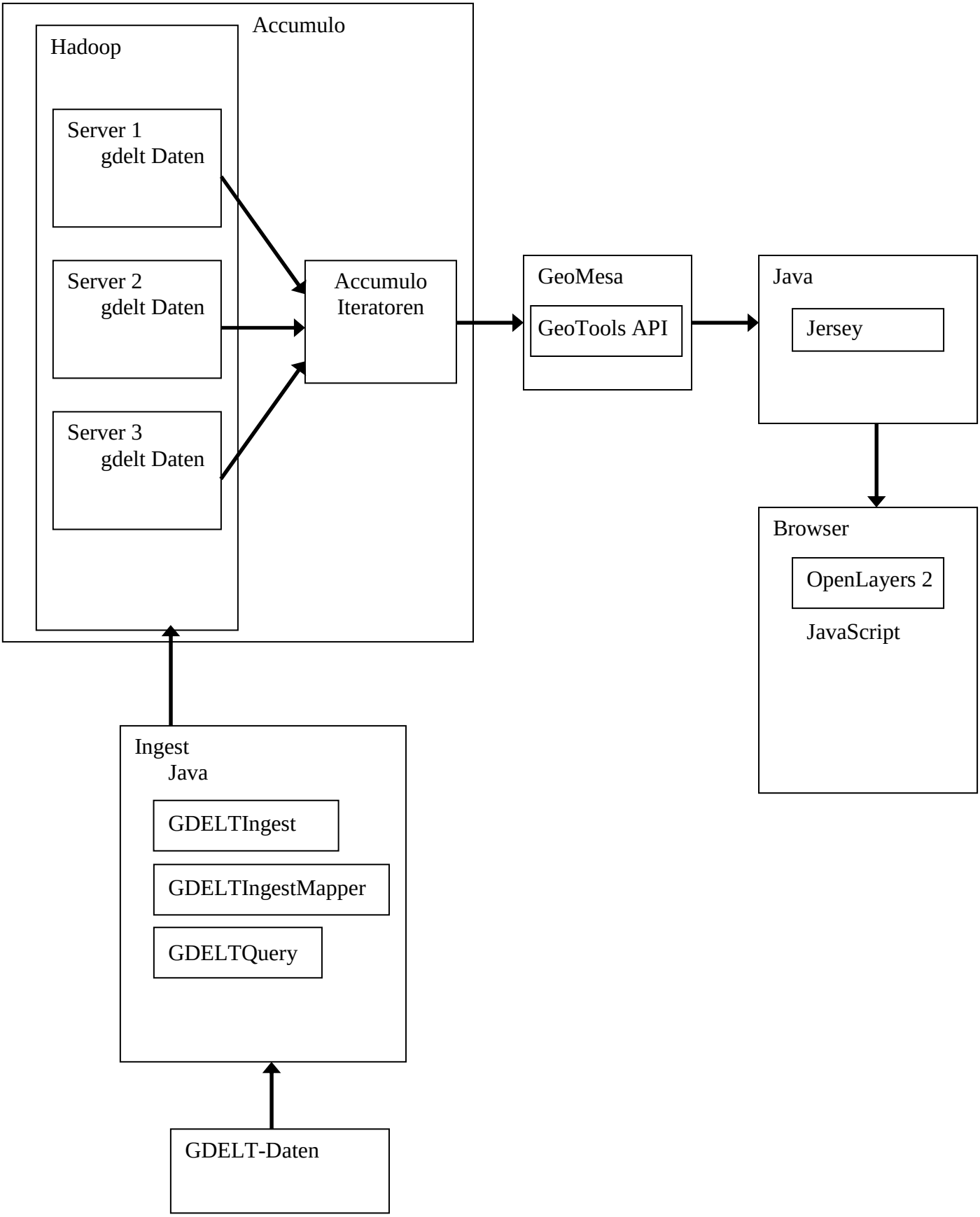
Bei Version 1.1.0-rc.6 handelt es sich um das (zu diesem Zeitpunkt) aktuellste Stable-Release von GeoMesa.

**7. OpenLayers 2**

OpenLayers ist in JavaScript verfasst und dient zur graphischen Darstellung von Geodaten im Webbrowser. Durch die Implementierung mehrerer Schnittstellen wird eine clientseitige Entwicklung unabhängig der genutzten Serversoftware ermöglicht.

Wir haben uns für OpenLayers 2 entschieden, da es im Vergleich zu ähnlichen Frameworks auf Kosten der Komplexität erweiterte Möglichkeiten in Handling und Visualizierung der Daten ermöglicht.

OpenLayers 3 wurde nicht genutzt, da es zu diesem Zeitpunkt zu wenig Stabilität, Dokumentation und Erweiterungen Dritter bietet.

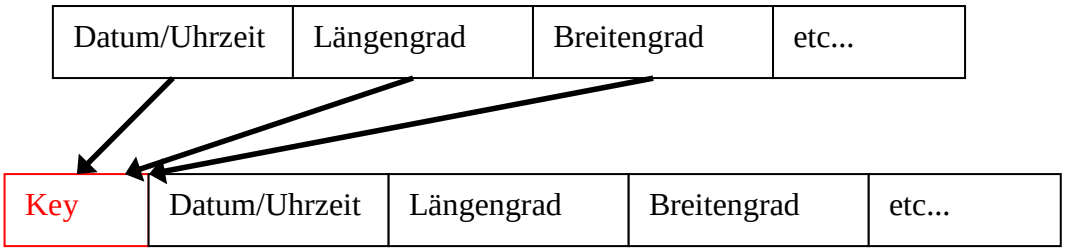

Accumulo
Hadoop
Server 1
gdelt Daten
Server 2
gdelt Daten
Server 3
gdelt Daten
Accumulo
Iteratoren
GeoMesa
GeoTools API
Java
Jersey
Browser
OpenLayers 2
JavaScript
Ingest
Java
GDELTIngest
GDELTIngestMapper
GDELTQuery
GDELT-Daten

Accumulo

Der Key-Value-Store Accumulo speichert und liest jeden Eintrag mittels eines Keys. Diese sind einzigartig und identifizieren einen Eintrag. Accumulo sortiert die Keys und speichert sie verteilt auf einer beliebigen Anzahl von Servern. Anders als bei relationalen Datenbanken ist dieser Key nicht sequentiell generiert, sondern basiert auf einer oder mehreren der wichtigsten Eigenschaften einer Zeile. Somit kann die Datenbank, wenn nach dieser Eigenschaft gesucht wird direkt auf die entsprechende Zeile zugreifen und den entsprechenden Eintrag liefern.

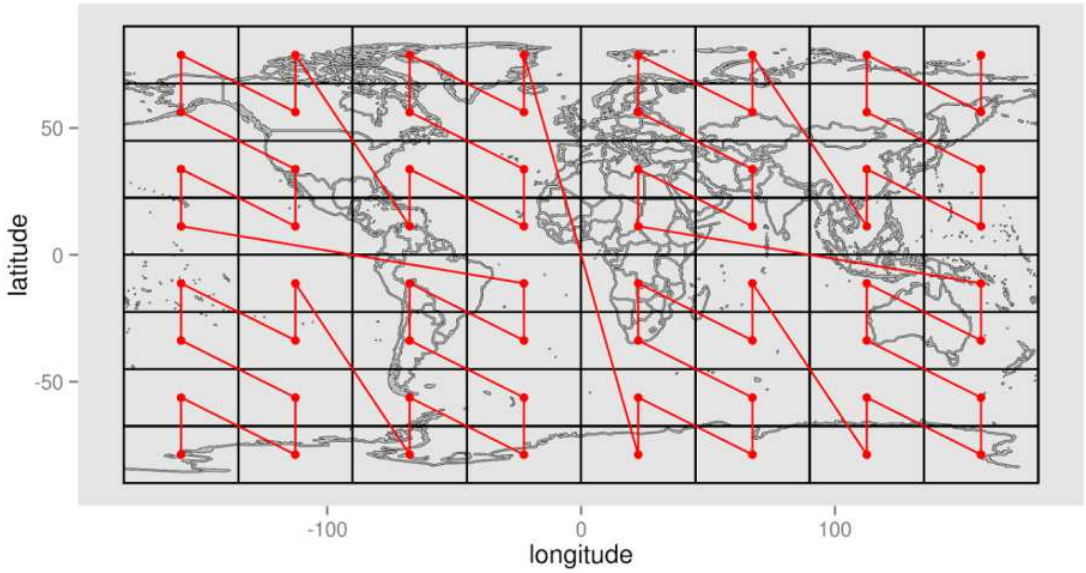
Geomesa

Geomesa baut auf diesem Key-Value-Store auf, indem es Ort und Zeit auf einen Key abbildet. Somit wird das suchen nach Events in einem bestimmten räumlichen und zeitlichen Bereich erheblich beschleunigt.




Andere Anfragen, wie z.B. eine Liste aller Events eines bestimmten Typs provitieren somit nicht von dieser Architektur.

Der von Geomesa genutzte, mathematische Ansatz ist die Abbildung der Weltkarte auf eine Z-Kurve. Dazu wird ein Raster über die Karte gelegt, dessen Zellen einzeln und nacheinander von der Z-Kurve besucht werden (Siehe Grafik).




Durch die sequentielle Nummerierung der einzelnen Punkte (Zellen) werden die x- und die y-Achse effektiv auf eine Dimension reduziert. Geomesa adaptiert dieses Prinzip und erweitert es insofern, dass 3 Dimensionen (Zeit) auf eine Dimension (Key) abgebildet werden.

Die eigentliche Struktur ist noch einmal komplexer, da sie sowohl Informationen zu Zeilen/Spalten, wie auch einen Timestamp und Security Tags enthält. Der zum Key gehörige Inhalt wird als SimpleFeature gespeichert.

<table>
<tr><th colspan="5">KEY</th><th>VALUE</th></tr>
<tr><th colspan="3" rowspan="2">ROW</th><th colspan="2">COLUMN</th><th rowspan="2">TIMESTAMP</th><th rowspan="2">VIZ</th><th rowspan="2">Byte-encoded<br>SimpleFeature</th></tr>
<tr><th>COLUMN FAMILY</th><th>COLUMN QUALIFIER</th></tr>
<tr><td>Epoch Week<br>2 bytes</td><td>Z3(x,y,t)<br>8 bytes</td><td>Unique ID (such as UUID)</td><td>"F"</td><td>-</td><td>-</td><td>Security tags</td><td></td></tr>
</table>

(Dies entspricht der Standardeinstellung von GeoMesa, welche geändert werden kann.)

**Darstellung der Verarbeitung einer Anfrage im Browser:**

Wird im Browser über die aufgerufene Seite *BigData.html* eine Anfrage gestellt, so wird im Front-End (genauer *BigData.js*) eine .json-Datei generiert, welche per Jersey POST an das Java-BackEnd (*Jersey.java*) übermittelt wird. Dort wird anhand der Datei ein Filter und mit diesem eine Query erstellt. Die Einzelergebnisse dieser Query werden in Objekte der Klasse *event* umgewandelt und in einem Objekt der Sammelklasse *events* gespeichert. Diese wird als .json per Jersey POST response zurück an das FrontEnd geliefert, welches anschließend ein Layer in der ausgewählten Darstellungsform generiert.