XML-Projekt SoSe 2012

GPSies.org, DBPedia.org, Twitter.com, XML/XSD/XSLT

Gruppe 11

Eike Cochu, Samer El-Safadi, Hannes Geist,
Cenk Gündogan, Michael Pluhatsch

14. Juli 2012

Inhaltsverzeichnis

1 Aufgabenstellung und Organisation

Gruppenleiter: Hannes Geist

Dokumentierer: Eike Cochu

1.1 Aufgabenstellung

Durch den XML-Endpoints von gpsies.com 100.000 Wanderstrecken laden, per XSLT-Schema in ein eigens entwickeltes XSD-Schema transformieren und in einer geeigneten XML-Datenbank speichern. Zu diesen Strecken soll per SPARQL der Endpoint von dbpedia.org abgefragt und alle auf (oder an) der Strecke liegenden Sehenswürdigkeiten abgefragt werden können. Ein HTML-Formular (oder ähnliches) entwickeln, mit dem die XML-Datenbank abgefragt werden kann. Das Ergebnis der Abfrage soll wiederum per XSLT in HTML transformiert und zusammen mit den vorhandenen Sehenswürdigkeiten und den zu diesen Sehenswürdigkeiten vorhandenen Tweets von Twitter.com angezeigt werden.

1.2 Aufgabenverteilung und -bewältigung

- Hannes Geist: XSLT-Transformierungen und Füllen der XML-Datenbank
- Cenk Gündogan: Abfrage der Strecken von gpsies.org
- Eike Cochu: Erstellung des XSD-Schemas, Bereitstellung eines Servers, Dokumentation
- Samer El-Safadi: SPARQL-Abfrage der Sehenswürdigkeiten von dbpedia.org
- Michael Pluhatsch: HTML-Formular mit PHP, Abfrage der XML-Datenbank

1.2.1 Hannes Geist

1.2.2 Cenk Gündogan

Aufgabenbereich: Entwerfen und Implementieren eines Crawlers, der mind. 100.000 Tracks vom gpsies.org Server herunterlädt und lokal in einer gültigen XML-Datei abspeichert.

Bei der Entwicklung des Crawlers wurden die Eigenschaften „Stabilität“ und „Schnelligkeit“ sehr hoch priorisiert. „Stabilität“ bedeutet, dass der Crawler bei Verbindungsabbrüchen weiter arbeiten kann und als Resultat immer noch eine korrekte XML-Datei mit gültigem Inhalt produziert wird. „Schnelligkeit“ des Crawlers wird durch Parallelisierung des Verbindungsaufbaus zum Server gewährleistet. Eine weitere nennenswerte Eigenschaft ist die Tatsache, dass das Resultat des Crawlers in komprimierter Form (mit gzip) erstellt wird. Das Ergebnis ist dann sehr einfach und unproblematisch auf verschiedene Rechner übertragbar.

Die Anfragen an den Server geschehen via Http POST und sehen folgendermaßen aus:

key	der API-Key wurde von gpsies.org bereitgestellt. Ohne diesen ist eine Anfrage an den Server nicht möglich
country	wir beschränken uns nur auf Tracks aus Deutschland
filetype	Als Dateityp der GPS-Daten erwarten wir KML
limit	es sollen **limit** viele Track-Ids übertragen werden
resultpage	pro Query können immer nur 100 Tracks übermittelt werden, mit inkrementierender **resultpage** kann man die nächsten 100 Tracks abfragen

1. URL1: *http://www.gpsies.org/api.do?**key**=<key>&**country**=DE& **limit**=100&**resultpage**=i*
   Es werden die **i.**en 100 Tracks geladen
2. URL2: *http://www.gpsies.org/api.do?**key**=<key>&**limit**=100 &**filetype**=kml&**fileId**=f1&**fileId**=f2...*
   Die 100 TrackIds, die mit URL1 erfragt wurden, werden nun alle an die 2. URL angefügt.
   Mit URL2 erhalten wir detaillierte Informationen zu jedem Track

Zur Realisierung des Crawlers wurde ein Maven-Projekt erstellt und zur leichten Handhabung der REST-Abfragen der HttpClient von org.apache.httpcomponents benutzt. Die Programmiersprache ist Java.

Der Crawler besteht aus zwei Hauptkomponenten, dem Master und die Worker. Zuerst wird vom Master ein komprimierter Output-Stream(gzip), verknüpft mit einer lokalen Datei, geöffnet. Der Master stellt nun in einer Schleife, die von **i=0** bis **i=*iterations*** geht, die Anfrage (URL1) an den Server, um die **i.**en 100 TrackIds zu erhalten. Aus der Response des Servers werden mittels

regular expression die einzelnen TrackIds heraus gefiltert. Mit den erhaltenen TrackIds kann nun die zweite Abfrage an den Server gestellt werden (URL2), um detaillierte Informationen zu jedem Track zu erhalten. Diese zweite Response wird nun wieder mittels regular expression nach Tracks untersucht. Bei jedem Fund eines Tracks, wird ein Worker erzeugt und diesem der gefundene Track übergeben. Der Master wiederholt diesen Prozess bis keine weiteren Tracks in der Response gefunden werden und wartet auf die Terminierung aller Worker. Wenn alle Worker terminiert sind, entsteht ein gültiger XML-Abschnitt, der in den komprimierten Output-Stream geschrieben und geflusht werden kann. Nun beginnt die nächste Iteration des Masters und dieser Prozess führt sich fort, bis **i=iterations** erreicht ist.

Wird ein Worker erstellt und ihm ein Track übergeben, wird der Download-Link der KML-Datei dieses Tracks via regular expressions heraus gefiltert. Die KML-Datei wird mit einem InputStreamReader geöffnet und die enthaltenen Koordinaten ausgelesen. Wir entschieden uns hierbei für eine „nimm jede 10.“-Politik, da es überaus viele, sehr dicht beieinander liegende, Koordinaten zur Verfügung gab. Zum Beenden des Workers wird der Track samt ausgelesener Koordinaten an den Master übergeben.

Das Resultat ist eine komprimierte XML-Datei mit 100.000 Tracks + Koordinaten. Die Größe der Datei liegt bei ungefähr ∼200MB. Entpackt liegt die Größe bei ∼1.1GB. Die ideale Crawl-Dauer liegt bei ∼3.5h-4h

Aufruf des Crawlers: java -jar ./crawler „api-key“ „/path/to/file.xml.gz“ „number of iterations“ Die Dateiendung muss „gz“ sein, da eine komprimierte Datei erzeugt wird. Eine Iteration beinhaltet im besten Fall 100 Tracks, so erhält man bei 1000 Iterationen 100.000 Tracks.

Probleme: Das schwerwiegenste Problem war die Tatsache, dass der Crawler aufgrund von langen Ausfallzeiten seitens des Servers nicht genügend Tracks produzieren konnte und durch viele Timeouts um mehrere Stunden verlangsamt wurde.

1.2.3 Eike Cochu

Aufgabenbereich: Erstellung eines XSD-Schemas für die XML-Datenbank, Bereitstellung des Datenbank- und Applikationsservers sowie Installation der Datenbank, Einfügen der Inhalte in die Datenbank und Installation der Webseite, kleine kosmetische Änderungen an der Webseite und Formatierung.

Zum XSD-Schema: Nachdem wir uns mit einem kurzen Testlauf einige Anschauungsdaten von gpsies.org beschafft hatten, konnten wir basierend darauf grob die Anforderungen an das XSD-Schema entwerfen. Die Idee des Schemas sollte es sein, mehr mit Attributen zu arbeiten und so die Anzahl an Elementen zu verringern, damit die Speichergröße der resultierenden Dateien minimal gehalten wird. Dazu haben wir uns entschlossen, einige weniger wichtige Datenteile der crawl-Daten garnicht erst mit in die Datenbank zu speichern, im Schema waren diese dann auch nicht vorgesehen. Die meisten Elemente sind mittels minOccurs=0 ebenfalls optional gehalten, Attribute sind per default optional.

Wir haben uns für BaseX als Datenbank entschieden, da dieses auf Java basiert und somit systemunabhängig installierbar und auch leicht zu handhaben ist. Die Datenbank wurde auf einem dedizierten Ubuntu 12.04 Server installiert, als Webserver wurde Apache + PHP 5 gewählt.

1.2.4 Samer El-Safadi

1.2.5 Michael Pluhatsch

Organisatorisches: Zur Gruppenkoordinierung hat sich unsere Gruppe jede Woche Sonntags beim Gruppenleiter Hannes Geist getroffen und das weitere Vorgehen besprochen.

2 Datenbank-Info

Verwendete Datenbanksoftware: BaseX XML-Datenbank Anzahl der vorhandenen Datensätze: 100.000