Large-scale Publikationsanalyse und Geodaten-Visualisierung

Teilnehmer:
- Richard Khulusi
- Daniel Alexander

Verantwortlicher:
- Dr. Anika Groß [link]

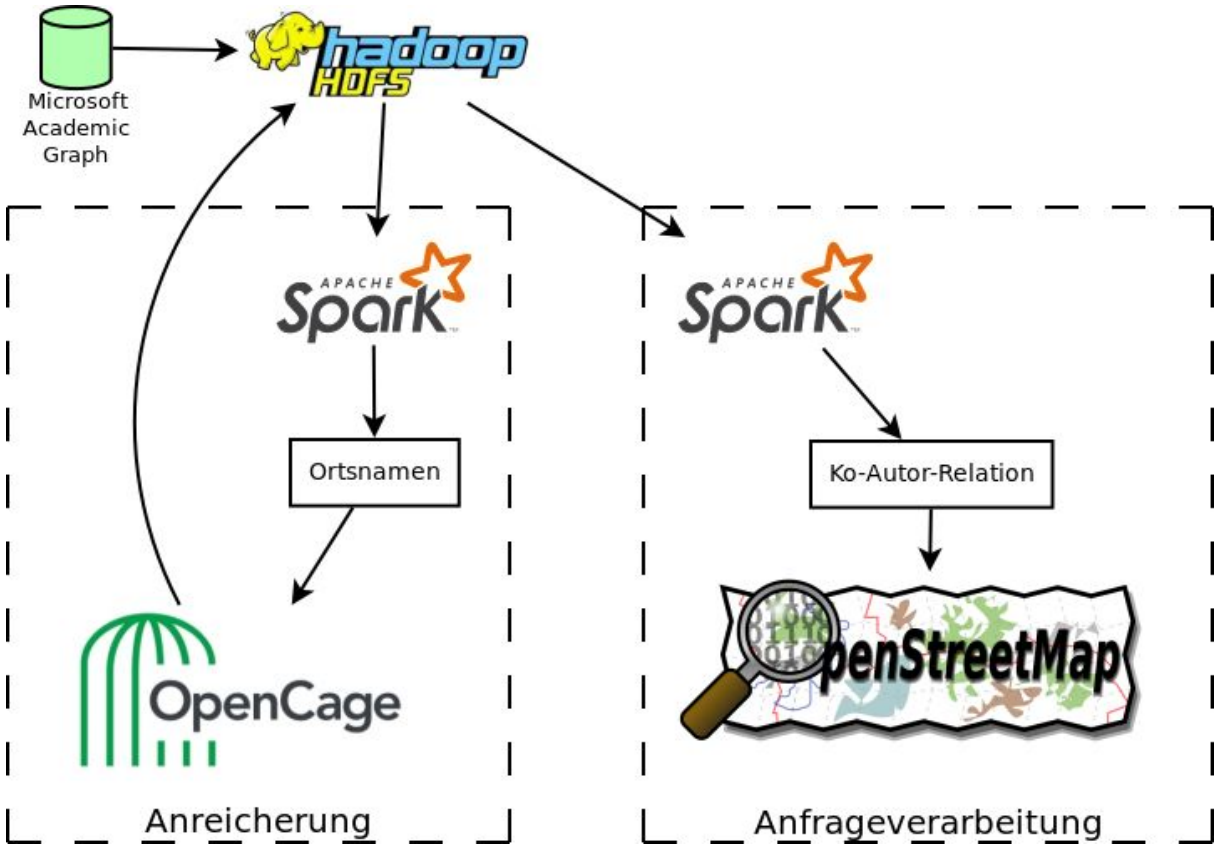
Repository:
- Github [link]

Aufgabenstellung:

Aufgabe ist es, die Beziehungen zwischen Arbeitsgruppen in verschiedenen Orten zu analysieren und zu visualisieren. Der Fokus liegt dabei auf der Analyse von Ko-Autor-Beziehungen. Wir sollen beispielsweise herausfinden, mit welchen anderen Hochschulen die Datenbank-Abteilung der Uni Leipzig bei der Erstellung ihrer Publikationen zusammengearbeitet hat. Als Datenquelle dient der Microsoft Academic Graph (MAG). Zur Visualisierung der Ergebnisse soll eine Landkarte verwendet werden, auf der die Ko-Autor-Beziehungen als Verbindungslinien zwischen verschiedenen Orten dargestellt werden.

Weiterführende Aufgabenstellungen:
Je nach Fortschreiten des Projekts können weitere Fragestellungen behandelt werden. Diese könnten sich mit der Visualisierung von Standorten befassen, deren Publikationen häufig zitiert werden oder von Zusammenhängen zwischen Orten und Forschungsgebieten (FieldOfStudy). Damit könnten wichtige Zentren für die Erforschung bestimmter Themengebiete ausfindig gemacht werden.

Workflow:



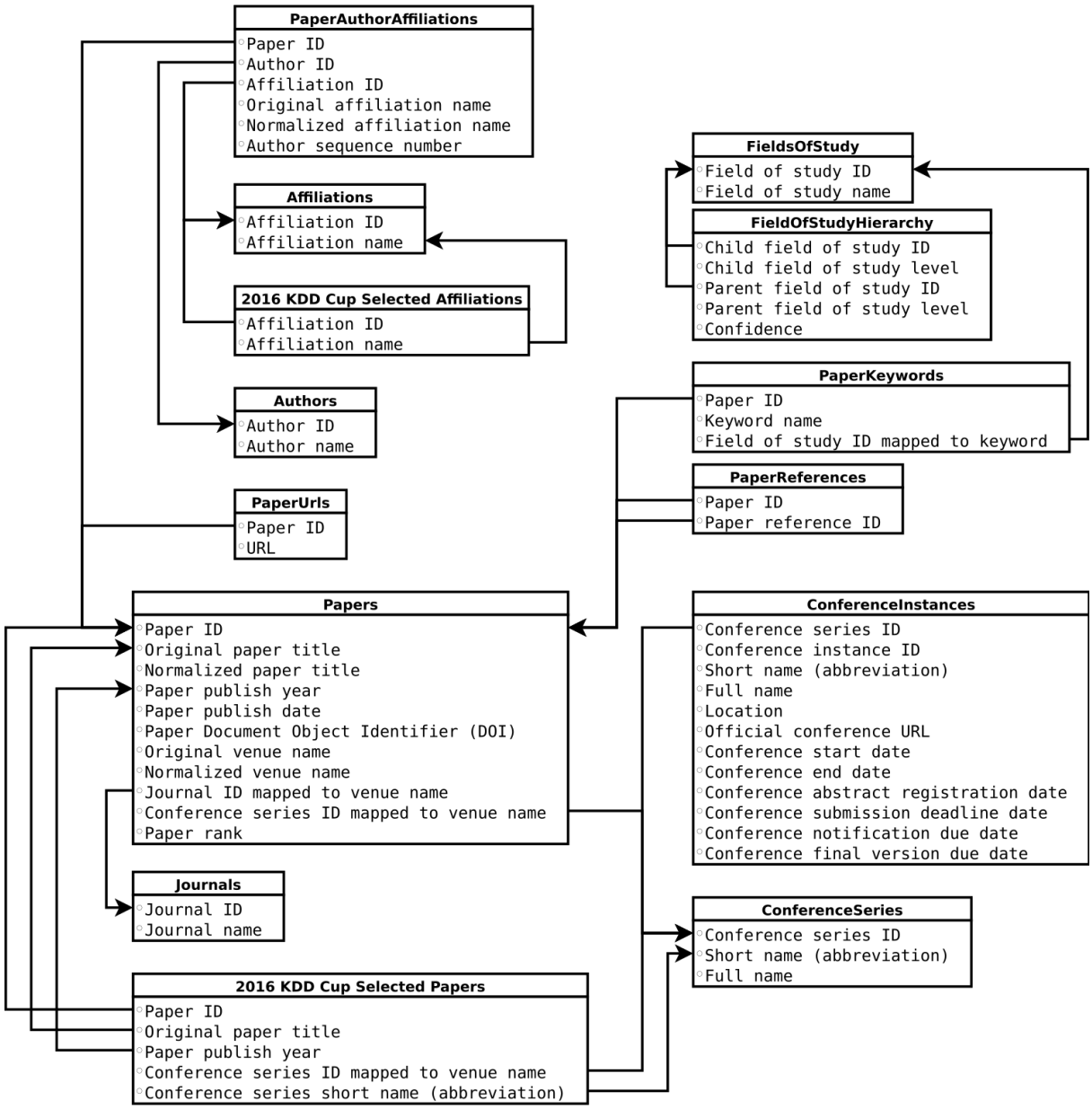
Software:

- Apache Spark [link]
  - Verarbeitung der Daten im Cluster
- Apache Hadoop [link]
  - Datenspeicherung für Cluster mittels HDFS
- OpenStreetMap [link]
  - Visualisierung von Geodaten

Datenimport:

Die Publikationsdaten entstammen dem Microsoft Academic Graph [link]. Es handelt sich dabei nach Angaben der Verantwortlichen um einen heterogenen Graph, der Informationen zu wissenschaftlichen Publikationen beinhaltet. Das betrifft Zitier-Beziehungen zwischenden Publikationen sowie Informationen zu Autoren, Institutionen, Journals und wenigen Konferenzen. Die Daten werden als gezippte Textdateien (.TXT) zur Verfügung gestellt und sind über HTTP verfügbar.

Datenbankschema:



Umfang der Daten: (ein vollständiger Eintrag pro Zeile)

Datei(en)	**Größe**
gezipptes Paket	29 GB
entpackte Textdateien	98 GB

Datei	**Zeilen**
PaperUrls.txt	454.070.767
PaperAuthorAffiliations.txt	337.000.600
FieldOfStudyHierarchy.txt	182.103
Conferences.txt	1.283
2016KDDCupSelectedAffiliations.txt	741
Papers.txt	126.909.021
FieldsOfStudy.txt	53.834
ConferenceInstances.txt	50.202
Journals.txt	23.404
Authors.txt	114.698.044
license.txt	85
readme.txt	90
PaperKeyWords.txt	158.280.968
2016KDDCupSelectedPapers.txt	3.677
Affiliations.txt	19.843

Vorverarbeitung:

Zur Beantwortung der obigen Fragestellung ziehen wir die Tabelle PaperAuthorAffiliations heran. Diese enthält Aussagen über Autoren sowie Affiliations zu Publikationen. Aus dieser Tabelle wollen wir die Affiliations entnehmen, um damit Ortsinformationen zu erhalten (siehe Anreicherung). Weiterhin wollen wir mithilfe dieser Tabelle Ko-Autorschaften identifizieren.

Selektion: Welche Datensätze brauchen wir überhaupt?
Vermutlich können wir alle nötigen Informationen aus der PaperAuthorAffiliations-Relation erhalten.

Projektion: Welche Informationen davon brauchen wir?
Gewisse Informationen der PaperAuthorAffiliations werden nicht benötigt. So existieren beispielsweise für jeden Eintrag zwei Affiliations-Einträge. Einmal der Originaleintrag (in der Form, wie er aus dem Paper extrahiert wurde) und einmal eine überarbeitete und normalisierte Version. Letztere versucht eine Standardschreibweise für die Strings zu erzeugen, welche sich für eine automatische Weiterverarbeitung, wie es bei uns auch geplant ist (siehe Anreicherung), besser eignen sollte. Neben diesen normalisierten Affiliations werden auch die PaperID, AuthorID und AffiliationsID gebraucht. Noch offen ist die Frage nach der Author Sequenz Number. Vermutlich enthält diese Informationen zu den Ko-Authorbeziehungen. Somit wäre diese Information ebenfalls relevant.

Reduktion, gruppieren und verknüpfen:
Um den Umfang des Datensatzes zu minimieren und Bearbeitungszeiten damit zu verringern, werden vor der eigentlichen Verarbeitung nur die relevanten Daten extrahiert. Die Punkte Selektion und Projektion (s.o.) erklären bereits, welche Informationen ausgewählt werden (müssen). Die übrigen Daten entfallen in der Betrachtung, was die zu verarbeitende Datenmenge reduziert. Darüber hinaus können weitere Datenvolumen eingespart werden, in dem mittels Gruppierungen und DISTINCT Anweisungen doppelte Einträge, sofern nicht nützlich, entfernt werden, beziehungsweise zu einem Tupel zusammen gefasst werden. Solch ein Vorgehen wird besonders für den Punkt Anreicherung eine Rolle spielen, wo auf Grund von Dienstlimitierungen möglichst mit wenigen Affiliations gearbeitet werden sollte, ohne qualitative Verluste zu erleiden.

Anreicherung:

Nachdem der Datensatz auf den für uns relevanten Teil reduziert wurde und durch die Vorverarbeitung auch die Informationen in verarbeitbarer Form vorliegen, fehlt uns noch eine Information für die Analyse. Da wir Ko-Authorbeziehungen auf einer Karte darstellen wollen, werden Georeferenzen benötigt, welche im MAG nicht enthalten sind.

Wir werden in diesem Punkt also Informationen in den Datensatz einpflegen. Dafür wird eine neue Relation AffiliationsGeoreferences erzeugt. In dieser wird zu jeder Affiliation ein eindeutiger Ort in Form von Longituden und Latituden gemappt. Zudem werden Informationen über Land und Stadt abgespeichert, um später detaillierte Anfragen bezüglich einzelner Orte formulieren zu können. Mittels Selektionsanfragen anhand der Orte können so bequem Einschränkungen getroffen werden. Neben der Beantwortung gezielter Anfragen können wir so falls nötig auf Performanzprobleme reagieren.
Um an diese Informationen zu gelangen, nutzen wir einen GeoCoding Dienst namens OpenCageData [link]. Dieser basiert auf der OpenStreetMap Datenbank und bekommt den normalisierten Affiliationsstring übergeben. Zu diesem String sucht er selbstständig das passendste Ergebnis (auf die Art und Weise sowie die Korrektheit des Rankings haben wir keinen Einfluss*). Dieses wird uns nun in Form eines JSON-Objektes zurückgegeben, woraus wir uns die nötigen Informationen parsen und in der Relationen einfügen (Materialisierung).
Dieser Vorgang wird separat in einem Java-Programm ablaufen. Er stellt also einen einmaligen Arbeitsschritt im Workflow dar, da sich unsere Daten nicht ändern. Anzumerken sei dabei noch, dass es kritisch ist, dass wir nicht unnötig große Datenmengen in diesem Schritt bearbeiten, da der Dienst gewisse Limitierungen hat, welche die Bearbeitungszeit drastisch erhöhen könnte. So lässt sich nicht mehr als eine Anfrage pro Sekunde und nicht mehr als 2.500 Anfragen am Tag abfragen. Bei einem Umfang von 5.000 Affiliations wären dies schon zwei Tage für den Prozess der Anreicherung.

*) Sollte es zu Problemen mit der Korrektheit kommen, so wäre ein alternativer Weg, dass mittels OpenCageData nur Informationen über Land/Stadt extrahiert werden und der letzte Schritt zur Bestimmung der exakten GeoKoordinaten direkt in der Visualisierung mittels OpenLayers' interner Funktion ermittelt werden.

Analyse der Daten:

Die Verarbeitung der Daten soll mittels Spark auf einem HDFS (Hadoop Distributed Files System) erfolgen. Dazu wurde uns von der Universität Leipzig ein PC (Remote-PC) zur Verfügung gestellt. Wir haben ein HDFS im Pseudo Distributed Mode aufgesetzt und nach erfolgtem Download die Daten ins HDFS geladen. Der Zugriff auf Spark / HDFS erfolgt in Java, wir verwenden also die Spark Java API. Weiterhin bietet Spark mit SparkSQL [link] auch ein Framework, welches es ermöglicht auf geladenen Daten SQL-artig zu arbeiten, welches sich für die Analyse hier anbietet.
Nach einem ersten Plan sollte der Spark-Client des entfernten Uni-PCs vom privaten Arbeitsgerät (Lokal-PC) aus angesteuert werden. Dieser Zugriff erwies sich bereits zu Beginn als problematisch, sodass wir vorerst unsere Programme auf Lokal-PCs mit einem ähnlichen Setup aber reduzierten Daten testen und anschließend das

Programm als JAR-verpackt auf den Remote-PC transferieren und dort auf dem gesamten Datensatz ausführen.
Zur Verarbeitung der Daten mittels Spark werden die TXT-Dateien zeilenweise als n-Tupel entsprechend der jeweiligen Datei verpackt (Referenz: link). Die so aufbereiteten Daten können dann auf ihren relevanten Anteil reduziert und zur Beantwortung von Queries analog zu SQL verwendet werden.
Das Ergebnis soll Aussagen darüber liefern, welche Institutionen miteinander kooperieren, dafür werden entsprechende Kooperationen gezählt und für den Visualisierungsschritt weitergereicht. Je nachdem, was die Daten tatsächlich an Details preisgeben, wird eine unterschiedliche Auflösung in den Ergebnissen angestrebt. So steht bislang nicht fest, ob verschiedene Abteilungen einer einzigen Hochschule einzeln betrachtet werden, oder ob diese zu einem Knoten zusammengefasst werden.

Visualisierung der Ergebnisse:

Die Resultate sollen auf einer Landkarte dargestellt werden. Dabei werden Orte Knoten sein und Ko-Autor-Beziehungen werden zu Kanten zwischen diesen Knoten. Als Software zur Visualisierung soll die Open-Source API OpenLayers verwendet werden. Diese bietet das Erstellen einer Karte via JavaScript an, wie sie auch bei beispielsweise OpenStreetMap zu sehen ist (Referenz: [link]).
Wie im Punkt Anreicherung beschrieben, befinden sich zu diesem Zeitpunkt alle nötigen Daten in Relationen und so sollte es genügen diese mittels der Query in Spark zu laden und an eine Implementation von OpenLayers zu reichen.
In dieser ist zu klären, wie die Knoten angezeigt werden können, was primär vom genauen Umfang der Daten abhängt. So kann es bei zu vielen Daten zu Überdeckungsproblemen kommen, welche es nötig machen, entweder die dargestellten Daten in ihrer Anzahl weiter zu reduzieren oder durch Zusammenfassen die Menge an Daten auf der Karte zu reduzieren (beispielsweise durch einen Glyphansatz).
Weiterhin soll durch die Implementierung der Kanten die Darstellung von möglichst vielen Ko-Autor-Beziehungen gleichzeitig möglich sein, ohne die Darstellung zu überladen. Erste Ideen dazu würden Ansätze beinhalten, die Kanten nur nach Selektion von Knoten anzeigt, was mit einem Glyphenansatz zu bewältigen wäre.
Durch die Nutzung der OpenLayers API werden wir die resultierende Karte in Javascript im WebBrowser anzeigen lassen.