

Hochschule Karlsruhe – Data Engineering WS 2021/2022 DSCB330 – Vorlesung 13 – Apache Spark



Data Integration

- Data Formats (csv, XML, json)
- Extract, Transform, Load
- **Object Relation Mapper (ORM)**
- Staging

Data Processing

- Relationale Datenbanken
- nicht-relationale Datenbanken
- Resource Description Framework (RDF)
- Ontologien
- Data Warehouse
- **MapReduce**
- **Large Scale Data Analytics – Apache Spark**

Data Modelling

- Serialisierung
- OPC UA
- MQTT
- Pub/Sub
- Data pipelines
 - Apache Airflow
 - gRPC

Web-Service Architektur

- Front-End
- Backend for Frontend (BFF)
- Micro Services
- Docker Container

Security

- Security ist wichtig in allen Phasen der Softwareentwicklung und Datenbereitstellung.

Übung

- Erstellung eines Daten-Modells einer prozesstechnischen Anlage
- Statische Daten
- Dynamische Daten
- Auswertung der Daten



MapReduce

- + Programmiermodell zur Auswertung großer Datenmengen (mehrere Petabyte) verteilt auf Clustern
- + Verarbeitung in drei Phasen
 - Map
 - Shuffle
 - Reduce
- + Map und Reduce werden vom Anwender spezifiziert

Quelle: <https://de.wikipedia.org/wiki/MapReduce>

Hochschule Karlsruhe | Data Engineering | DSCB330 | VL 13 | WS 2021/2022 | Dipl.-Phys. Thomas Bierweiler | thomas.bierweiler@h-ka.de

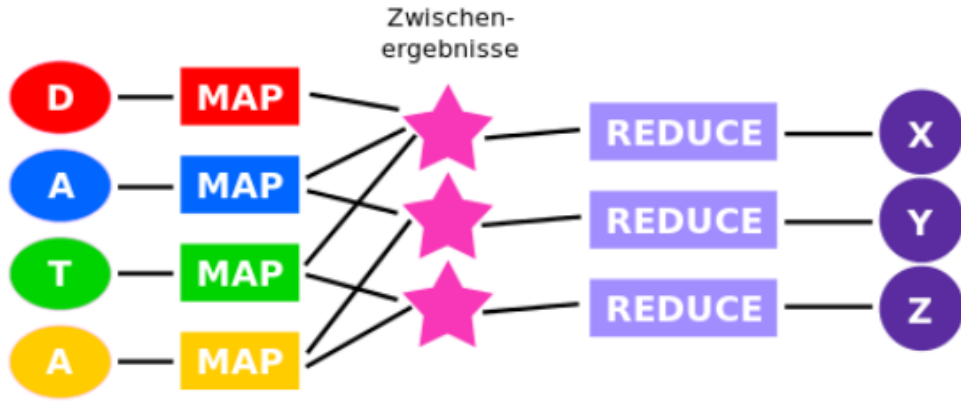
14.01.2022



MapReduce



Arbeitsweise



+ Map-Phase:

- Die Eingabedaten (D, A, T, A) werden auf eine Menge von Map-Prozessen verteilt (illustriert durch bunte Rechtecke), welche jeweils die vom Nutzer bereitgestellte Map-Funktion berechnen.
- Die Map-Prozesse werden idealerweise parallel ausgeführt.
- Jede dieser Map-Instanzen legt Zwischenergebnisse ab (illustriert durch pinkfarbene Sterne).

- Von jeder Map-Instanz fließen Daten in eventuell verschiedene Zwischenergebnisspeicher.

+ Shuffle-Phase:

- Die Zwischenergebnisse werden gemäß den Ausgabeschlüsseln, die von der Map-Funktion produziert wurden, neu verteilt, sodass alle Zwischenergebnisse mit demselben Schlüssel im nächsten Schritt auf demselben Computersystem verarbeitet werden.

+ Reduce-Phase:

- Für jeden Satz an Zwischenergebnissen berechnet jeweils genau ein Reduce-Prozess (illustriert durch violette Rechtecke) die vom Nutzer bereitgestellte Reduce-Funktion und damit die Ausgabedaten (illustriert durch violette Kreise X, Y und Z).
- Die Reduce-Prozesse werden idealerweise ebenfalls parallel ausgeführt.

Quelle: <https://de.wikipedia.org/wiki/MapReduce>

Hochschule Karlsruhe | Data Engineering | DSCB330 | VL 13 | WS 2021/2022 | Dipl.-Phys. Thomas Bierweiler | thomas.bierweiler@h-ka.de

14.01.2022

Map-Phase

- + Map bildet ein Paar, bestehend aus einem Schlüssel k und einem Wert v , auf eine Liste von neuen Paaren (l_r, x_r) ab, welche die Rolle von Zwischenergebnissen spielen. Die Werte x_r sind vom gleichen Typ wie die Endergebnisse w_i .
- + Bei einem neuen Paar (l, x) verweist der von Map vergebene Schlüssel l dabei auf eine Liste T_l von Zwischenergebnissen, in welcher der von Map berechnete x gesammelt wird.
- + Die Bibliothek ruft für jedes Paar in der Eingabeliste die Funktion Map auf.
- + All diese Map-Berechnungen sind voneinander unabhängig, so dass man sie nebenläufig und verteilt auf einem Computercluster ausführen kann.

Shuffle-Phase

- + Bevor die Reduce-Phase starten kann, müssen die Ergebnisse der Map-Phase nach ihrem neuen Schlüssel l in Listen T_l gruppiert werden.
- + Wenn Map- und Reduce-Funktionen nebenläufig und verteilt

ausgeführt werden, wird hierfür ein koordinierter Datenaustausch notwendig.

- + Die Performanz eines Map-Reduce-Systems hängt maßgeblich davon ab, wie effizient die Shuffle-Phase implementiert ist.
- + Der Nutzer wird in der Regel nur über die Gestaltung der Schlüssel l auf die Shuffle-Phase Einfluss nehmen. Daher reicht es, sie einmalig gut zu optimieren, und zahlreiche Anwendungen können hiervon profitieren.

Reduce-Phase

- + Sind alle Map-Aufrufe erfolgt bzw. liegen alle Zwischenergebnisse in T_l vor, so ruft die Bibliothek für jede Zwischenwertliste die Funktion Reduce auf, welche daraus eine Liste von Ergebniswerten w_j berechnet, die von der Bibliothek in der Ausgabeliste als Paare (l, w_j) gesammelt werden.
- + Auch die Aufrufe von Reduce können unabhängig auf verschiedene Prozesse im Computercluster verteilt werden.

Quelle: <https://de.wikipedia.org/wiki/MapReduce>

Hochschule Karlsruhe | Data Engineering | DSCB330 | VL 13 | WS 2021/2022 | Dipl.-Phys. Thomas Bierweiler | thomas.bierweiler@h-ka.de

14.01.2022

MapReduce Beispiel (1/6)



Beispiel

- + Man möchte für umfangreiche Texte herausfinden, wie oft welche Wörter vorkommen.

Angabe der Map- und Reduce-Funktionen

```
map(String name, String document):  
    // name: document name ("key")  
    // document: document contents ("value")  
    for each word w in document:  
        EmitIntermediate(w, 1);  
  
reduce(String word, Iterator partialCounts):  
    // word: a word ("key")  
    // partialCounts: a list of aggregated partial counts ("values") for 'word'  
    int result = 0;  
    for each v in partialCounts:  
        result += v;  
    Emit(word, result);
```

Quelle: <https://de.wikipedia.org/wiki/MapReduce>

Hochschule Karlsruhe | Data Engineering | DSCB330 | VL 13 | WS 2021/2022 | Dipl.-Phys. Thomas Bierweiler | thomas.bierweiler@h-ka.de

14.01.2022



MapReduce

Beispiel (2/6)



Map-Phase

- + Map bekommt jeweils einen Dokumentnamen *name* und ein Dokument *document* als Zeichenkette übergeben.
- + Map durchläuft das Dokument Wort für Wort.
- + Jedes Mal, wenn ein Wort *w* angetroffen wird, wandert eine 1 in die *w*-Zwischenergebnisliste (falls diese noch nicht existiert, wird sie angelegt).
- + Ist man mit allen Wörtern durch und hat der Text insgesamt *n* verschiedene Wörter, so endet die Map-Phase mit *n* Zwischenergebnislisten, jede für ein anderes Wort sammelnd, welche so viele 1-Einträge enthält, wie das entsprechende Wort im Dokument gefunden wurde.
- + Eventuell liefen viele Map-Instanzen gleichzeitig, falls der Bibliothek mehrere Wörter und Dokumente übergeben wurden.

Shuffle-Phase

- + Die Zwischenergebnislisten von mehreren Prozessen / Systemen für das gleiche Wort *w* werden zusammengefasst,

und auf die Systeme für die Reducer verteilt.

Reduce-Phase

- + Reduce wird für das Wort *word* und die Zwischenergebnisliste *partialCounts* aufgerufen.
- + Reduce durchläuft die Zwischenergebnisliste und addiert alle gefundenen Zahlen auf.
- + Die Summe *result* wird an die Bibliothek zurückgegeben, sie enthält, wie oft das Wort *word* in allen Dokumenten gefunden wurde.
- + Die Zwischenergebnisse konnten parallel, durch gleichzeitige Reduce-Aufrufe, berechnet werden.

Quelle: <https://de.wikipedia.org/wiki/MapReduce>

Hochschule Karlsruhe | Data Engineering | DSCB330 | VL 13 | WS 2021/2022 | Dipl.-Phys. Thomas Bierweiler | thomas.bierweiler@h-ka.de

14.01.2022



MapReduce Beispiel (3/6)



```
Text = "Fest gemauert in der Erden Steht die Form, aus Lehm gebrannt. Heute muß die Glocke werden,  
Frisch, Gesellen! seid zur Hand. Von der Stirne heiß Rinnen muß der Schweiß, Soll das Werk  
den Meister loben, Doch der Segen kommt von oben."
```

+ Aufteilung in Sätze, Entfernung von Satzzeichen

```
Eingabeliste = [ (satz_1 = „fest gemauert in der erden steht die form aus lehm gebrannt“)  
                  (satz_2 = „heute muß die glocke werden frisch gesellen seid zur hand“)  
                  (satz_3 = „von der stirne heiß rinnen muß der schweiß soll das werk  
                        den meister loben doch der segen kommt von oben“) ]
```

+ Jeder Satz wird von einem Map-Prozess bearbeitet.

+ Die Map-Aufrufe generieren folgende Zwischenergebnispaare:

```
P1 = [ ("fest", 1), ("gemauert", 1), ("in", 1), ("der", 1), ("erden", 1),  
        ("steht", 1), ("die", 1), ("form", 1), ("aus", 1), ("lehm", 1), ("gebrannt", 1) ]  
P2 = [ ("heute", 1), ("muß", 1), ("die", 1), ("glocke", 1), ("werden", 1),  
        ("frisch", 1), ("gesellen", 1), ("seid", 1), ("zur", 1), ("hand", 1) ]  
P3 = [ ("von", 1), ("der", 1), ("stirne", 1), ("heiß", 1), ("rinnen", 1),  
        ("muß", 1), ("der", 1), ("schweiß", 1), ("soll", 1), ("das", 1), ("werk", 1), ("den", 1),  
        ("meister", 1), ("loben", 1), ("doch", 1), ("der", 1), ("segен", 1), ("kommt", 1),  
        ("von", 1), ("oben", 1) ]
```

Quelle: <https://de.wikipedia.org/wiki/MapReduce>

Hochschule Karlsruhe | Data Engineering | DSCB330 | VL 13 | WS 2021/2022 | Dipl.-Phys. Thomas Bierweiler | thomas.bierweiler@h-ka.de

14.01.2022

8



MapReduce Beispiel (4/6)



- + Die Map-Prozesse liefern ihre Paare an die MapReduce-Bibliothek, welche diese in den Zwischenergebnislisten sammelt. Parallel könnte folgendes geschehen (Die gleiche Taktung der 3 Map-Prozesse ist unrealistisch, tatsächlich überlappen sich die Ausführungen. Die T_wort-Listen sind lokal pro Map-Prozess vorhanden und werden **nicht** zwischen den Schritten synchronisiert):

1. Iteration:

```
P1: T_fest      = [ 1 ]      (neu)
P2: T_heute    = [ 1 ]      (neu)
P3: T_von      = [ 1 ]      (neu)
```

2. Iteration:

```
P1: T_gemauert = [ 1 ]      (neu)
P2: T_muß     = [ 1 ]      (neu)
P3: T_der      = [ 1 ]      (neu)
```

3. Iteration:

```
P1: T_in       = [ 1 ]      (neu)
P2: T_die      = [ 1 ]      (neu)
P3: T_stirne   = [ 1 ]      (neu)
```

- + Im vierten Schritt sieht man, dass Zwischenergebnislisten lokal für jeden Map-Prozess existieren und nicht global wiederverwendet werden können:

4. Iteration:

```
P1: T_der      = [ 1 ]      (neu, der 1. Map-Prozess hat noch kein T_der, nur P3)
P2: T_glocke   = [ 1 ]      (neu)
P3: T_heiss    = [ 1 ]      (neu)
```

5. Iteration

```
P1: T_erden    = [ 1 ]      (neu)
P2: T_werden   = [ 1 ]      (neu)
P3: T_rinnen   = [ 1 ]      (neu)
```

6. Iteration

```
P1: T_steht    = [ 1 ]      (neu)
P2: T_frisch   = [ 1 ]      (neu)
P3: T_muß      = [ 1 ]      (neu, der 3. Map-Prozess hat noch kein T_muß, nur P2)
```

Quelle: <https://de.wikipedia.org/wiki/MapReduce>

Hochschule Karlsruhe | Data Engineering | DSCB330 | VL 13 | WS 2021/2022 | Dipl.-Phys. Thomas Bierweiler | thomas.bierweiler@h-ka.de

14.01.2022

MapReduce

Beispiel (5/6)



- + Im siebten Schritt kommt dann zum ersten Mal vor, dass ein weiteres Vorkommen in einer bereits angelegten Zwischenergebnisliste gesammelt wird:

7. Schritt

```
P1: T_die      = [ 1 ]      (neu, der 1. Map-Prozess hat noch kein T_die)
P2: T_gesellen = [ 1 ]      (neu)
P3: T_der      = [ 1, 1 ]   (beim 3. Map-Prozess seit Iteration 2 vorhandene Liste verwenden)
```

- + usw.

Quelle: <https://de.wikipedia.org/wiki/MapReduce>

Hochschule Karlsruhe | Data Engineering | DSCB330 | VL 13 | WS 2021/2022 | Dipl.-Phys. Thomas Bierweiler | thomas.bierweiler@h-ka.de

14.01.2022

10



MapReduce

Beispiel (6/6)



- + Nach 21 Schritten sind alle drei Map-Prozesse mit ihrer Arbeit fertig, die Map-Phase endet und es beginnt die Reduce-Phase. Die Zwischenergebnislisten, die von verschiedenen Map-Prozessen zu demselben Wort angelegt wurden, werden zusammengefügt. Für jede der entstandenen Zwischenergebnislisten (hier sortiert aufgeführt)

- + können wir parallel einen Reduce-Prozess starten, der jeweils die Elemente aufzählt. Das Ergebnis von MapReduce sieht in etwa so aus:

```
Ausgabeliste = [ ("fest", 1), ("heute", 1), ("von", 2), ("gemauert", 1),  
                  ("muß", 2), ("der", 4), ("in", 1), ("die", 2), .. ]
```

```
                reduce  
T_der      = [ 1 ] ++ [ 1, 1, 1 ] -> [ 4 ]  
T_die      = [ 1 ] ++ [ 1 ]      -> [ 2 ]  
T_fest     = [ 1 ]                -> [ 1 ]  
T_gemauert = [ 1 ]                -> [ 1 ]  
T_glocke   = [ 1 ]                -> [ 1 ]  
T_heiss    = [ 1 ]                -> [ 1 ]  
T_heute    = [ 1 ]                -> [ 1 ]  
T_in       = [ 1 ]                -> [ 1 ]  
T_muß      = [ 1 ] ++ [ 1 ]      -> [ 2 ]  
T_stirne   = [ 1 ]                -> [ 1 ]  
T_von      = [ 1, 1 ]            -> [ 2 ]  
.  
.  
. (für alle verschiedenen T-Listen)
```

Quelle: <https://de.wikipedia.org/wiki/MapReduce>

Hochschule Karlsruhe | Data Engineering | DSCB330 | VL 13 | WS 2021/2022 | Dipl.-Phys. Thomas Bierweiler | thomas.bierweiler@h-ka.de

14.01.2022

11



Large Scale Data Analytics – Apache Spark



Apache Spark

- + Apache Spark™ is a multi-language engine for executing data engineering, data science, and machine learning on single-node machines or clusters.
- + Key features
 - Batch/streaming data
 - SQL analytics
 - Scalability – from your laptop to petabyte-scale
 - Advanced Analytics – Spark not only supports ‘Map’ and ‘reduce’. It also supports SQL queries, Streaming data, Machine learning (ML), and Graph algorithms.
 - Supports multiple languages: Java, Python, Scala
- + Apache Spark is a lightning-fast cluster computing technology, designed for fast computation. It is based on Hadoop MapReduce and it extends the MapReduce model to efficiently use it for more types of computations, which includes interactive queries and stream processing. The main feature of Spark is its in-memory cluster computing that increases the processing speed of an application.
- + Spark is designed to cover a wide range of workloads such as batch applications, iterative algorithms, interactive queries and streaming. Apart from supporting all these workload in a respective system, it reduces the management burden of maintaining separate tools.

Quelle: <https://spark.apache.org/>; https://www.tutorialspoint.com/apache_spark/apache_spark_introduction.htm

Hochschule Karlsruhe | Data Engineering | DSCB330 | VL 13 | WS 2021/2022 | Dipl.-Phys. Thomas Bierweiler | thomas.bierweiler@h-ka.de

14.01.2022

12



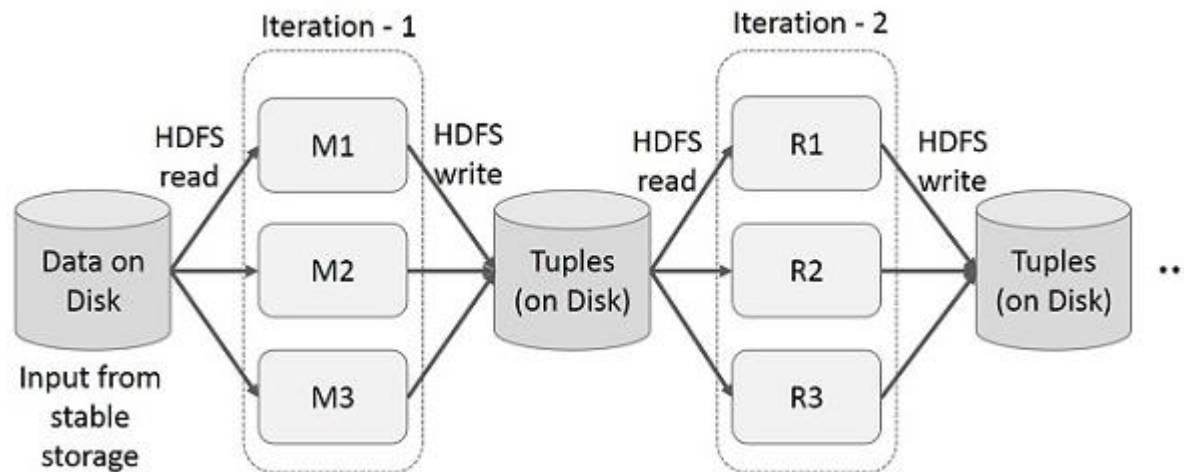
Apache Spark

How does Spark increase the speed of MapReduce (1/2)



Iterative Operations on MapReduce

- + overheads due to data replication, disk I/O, and serialization
- + HDFS: Hadoop Distributed File System



Quelle: https://www.tutorialspoint.com/apache_spark/apache_spark_rdd.htm

Hochschule Karlsruhe | Data Engineering | DSCB330 | VL 13 | WS 2021/2022 | Dipl.-Phys. Thomas Bierweiler | thomas.bierweiler@h-ka.de

14.01.2022

13

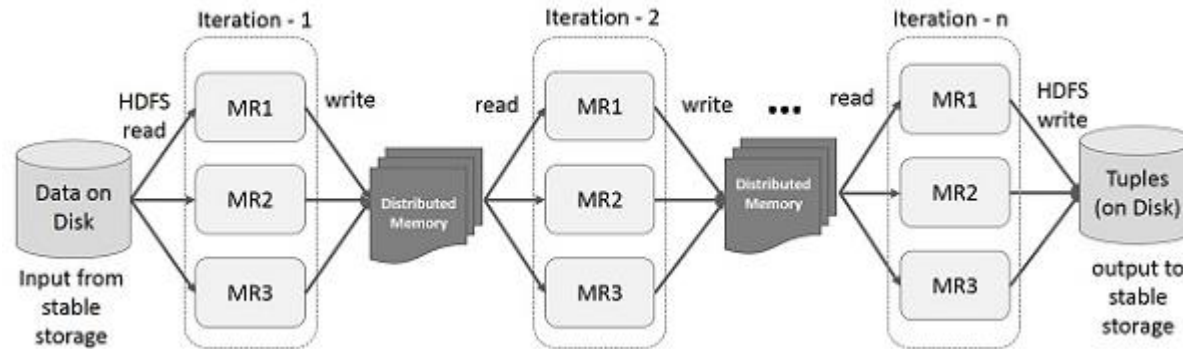
Apache Spark

How does Spark increase the speed of MapReduce (2/2)



Iterative Operations on Spark RDD

- + Resilient Distributed Datasets (RDD)
- + supports in-memory processing
- + intermediate results are stored in a distributed memory instead of stable storage (Disk)



Quelle: https://www.tutorialspoint.com/apache_spark/apache_spark_rdd.htm

Hochschule Karlsruhe | Data Engineering | DSCB330 | VL 13 | WS 2021/2022 | Dipl.-Phys. Thomas Bierweiler | thomas.bierweiler@h-ka.de

14.01.2022

14

Large Scale Data Analytics – Apache Spark



Installation des python-Pakets

```
python -m pip install wheel  
python -m pip install pyspark
```

Installation von Apache Spark unter Windows

- + <https://phoenixnap.com/kb/install-spark-on-windows-10>
- + Aliase für die App-Ausführung ausschalten
- + Im python-Installationspfad die Datei python.exe nach python3.exe kopieren.

Erstes Beispiel

- + Übung 33-Apache-Spark-Intro

Quelle: <https://spark.apache.org/>; <https://phoenixnap.com/kb/install-spark-on-windows-10>

Hochschule Karlsruhe | Data Engineering | DSCB330 | VL 13 | WS 2021/2022 | Dipl.-Phys. Thomas Bierweiler | thomas.bierweiler@h-ka.de




14.01.2022

15

← Einstellungen

🏠 Aliase für die App-Ausführung

Apps können einen Namen deklarieren, der zum Ausführen der App über eine Befehlszeile verwendet wird. Wenn mehrere Apps denselben Namen verwenden, wählen Sie den zu verwendenden Namen aus.

	Windows Package Manager Client winget.exe	<input checked="" type="checkbox"/> Ein
	App-Installer python.exe	<input type="checkbox"/> Aus
	App-Installer python3.exe	<input type="checkbox"/> Aus

Apache Spark Text Mining Projekt



Analyse von Patenttexten

- + Übung 34-Apache-Spark-ML-Claims
- + Erläuterung siehe <https://www.informatik-aktuell.de/entwicklung/methoden/einfuehrung-in-spark-ein-text-mining-projekt.html>
- + Beispieldaten (Datei patent_claims_fulltext.csv.zip) siehe https://bulkdata.uspto.gov/data/patent/claims/economics/2014/patent_claims_fulltext.csv.zip
bzw.
<https://bwsyncandshare.kit.edu/s/nEAyaFNDCozmSja>

Quelle: <https://www.informatik-aktuell.de/entwicklung/methoden/einfuehrung-in-spark-ein-text-mining-projekt.html>;
https://bulkdata.uspto.gov/data/patent/claims/economics/2014/patent_claims_fulltext.csv.zip

Hochschule Karlsruhe | Data Engineering | DSCB330 | VL 13 | WS 2021/2022 | Dipl.-Phys. Thomas Bierweiler | thomas.bierweiler@h-ka.de

14.01.2022

16



Apache Spark

Übungsaufgabe 16



Analyse von Patenttexten

- + Zu Übung 34-Apache-Spark-ML-Claims
- + In der Beispieldatei „patent_claims_excerpt.csv“ wird jeder Claim (Anspruch) eines Patents (pat_no) in einer Zeile dargestellt.

```
pat_no,claim_no,claim_txt,dependencies,ind_flg,appl_id
3930271,1,"1. A golf glove comprising at least an index
3930271,4,4. A golf glove adapted for use on one hand c
3930271,3,"3. A glove comprising an index finger recept
3930271,2,"2. A golf glove in accordance with claim 1 w
3930272,1,"1. In combination with a height adjustable c
3930272,3,3. The lock defined in claim 1 wherein the pi
3930272,2,2. The lock defined in claim 1 wherein the br
```

- + In Zeile 20 der Datei „claims.py“ wird die csv-Datei eingelesen.
- + Filtern Sie die Daten so, dass für jedes Patent nur der erste Claim (also claim_no=1) berücksichtigt wird.

- + Nach der Filterung sehen die Daten folgendermaßen aus:

```
pat_no,claim_no,claim_txt,dependencies,ind_flg,appl_id
3930271,1,"1. A golf glove comprising at least an index
3930272,1,"1. In combination with a height adjustable c
3930273,1,"1. A bed arrangement comprising a bed frame,a
3930274,1,"1. An assembly for use in recreational activ
3930275,1,"1. A method of manufacturing slippers each h
```

- + Führen Sie die Analyse der Patenttexte mit der erweiterten Filterung durch.
- + Erstellen Sie eine kurze Präsentation, in der Sie die Aufgabenstellung, die Vorgehensweise und das Ergebnis darstellen.

Quelle: <https://www.informatik-aktuell.de/entwicklung/methoden/einfuehrung-in-spark-ein-text-mining-projekt.html>;
https://bulkdata.uspto.gov/data/patent/claims/economics/2014/patent_claims_fulltext.csv.zip

Hochschule Karlsruhe | Data Engineering | DSCB330 | VL 13 | WS 2021/2022 | Dipl.-Phys. Thomas Bierweiler | thomas.bierweiler@h-ka.de

14.01.2022

17



