MA-INF 3241 - Lab Distributed Big Data Analytics

18/04/2017

Dr. Hajira Jabeen, Gezim Sejdiu
Smart Data Analytics (SDA)

- Prof. Dr. Jens Lehmann
  - Institute for Computer Science, University of Bonn
  - Fraunhofer Institute for Intelligent Analysis and Information Systems (IAIS)
  - Institute for Applied Computer Science, Leipzig.
- Machine learning techniques ("analytics") for Structured knowledge ("smart data").
- The group aims at covering the full spectrum of research including theoretical foundations, algorithms, prototypes and industrial applications.
Dr. Hajira Jabeen

- Senior Researcher at University of Bonn, 2016-
- PostDoc at AKSW, University of Leipzig, 2015
- Assistant Lecturer, IT University of Copenhagen, 2014
- Assistant Professor, Iqra University 2010-2013

Research Interests:
- Big Data, Data Mining
- Machine Learning and Analytics
- Semantic web and Structured Machine learning
- Evolutionary Computation, Optimization
Gezim Sejdu

- PhD Student & Research Associate at University of Bonn
- Guest Researcher at AKSW group at the University of Leipzig in 2015
- Research Interests:
  - Big Data, Data Mining and Data Analysis,
  - Semantic Web and Semantic Search,
  - Machine Learning,
  - Distributed Computing
Group Members

- Dr. Asja Fischer
- Dr. Henning Petzka
- Dr. Günter Kniesel
- Dr. Alexander Kister
- Dr. Muhammad Sheriff
- Dr. Hamed Shariat
- Dr. Bastian Haarmann
- ...Many others
Group’s Research Interests

- Distributed In-memory Analytics
- Semantic Question Answering
- Machine Learning on Structured Data
- Deep Learning
- Geospatial analysis
- Software Engineering for Data Science
Projects

- Big Data Europe, EU H2020, Big Data
- GEISER, BMWi
- HOBBIT, EU H2020, Big Data
- SLIPO, EU H2020, Big Data
  - Scalable linking and integration
- QROWD, EU H2020, Big Data
  - Big data integration of cities e.g. geographic, transport, meteorological,
- SAKE, BMWi
  - Semantic Analysis of Complex Events
Projects

- Domain Specific Languages (DSLs) for Machine Learning
- Smoothed Analysis of Structured Machine Learning Algorithms from Knowledge Graphs
- Cognitive Robotics
- Experimental Analysis of Class CS Problems
- Tensor Factorisation and Visualization for Knowledge Graphs
Software Projects

- SANSA - Distributed Semantic Analytics Stack
- AskNow - Question Answering Engine
- DL-Learner - Supervised Machine Learning in RDF / OWL
- LinkedGeoData - RDF version of OpenStreetMap
- DBpedia - Wikipedia Extraction Framework
- DeFacto - Fact Validation Framework
Distributed Semantic Analytics

Lead: Dr. Hajira Jabeen
  - Dr. Hamed Shariat Yazdi
  - Dr. Luís Paulo F. Garcia
  - Dr. Henning Petzka
  - Dr. Anisa Rula
  - Gezim Sejdiu
  - Nilesh Chakraborty
  - Heba Allah
  - Theresa Nathan
Semantic Question Answering

- Lead: Dr. Giulio Napolitano
  - Mohnish Dubey
  - Hamid Zafar
  - Debanjan Chaudhuri
  - Konrad Höffner
  - Diego Esteves
  - Gaurav Maheshwari
  - Priyansh Trivedi
Structured Machine Learning

- Lead: Prof. Jens Lehmann
  - Lorenz Bühmann (deputy)
  - Patrick Westphal
  - Simon Bin
  - Hajira Jabeen
  - Mofeed Hassan
Deep Learning

- Lead: Dr. Asja Fischer
  - Dr. Henning Petzka
  - Dr. Alexander Kister
  - Debanjan Chaudhuri
Geospatial Analytics

- Lead: Dr. Gev Ben-Haim
  - Dr. Alexander Kister
  - Dr. Mohamed Sherif
  - Claus Stadler
  - Sinan Shi
Software Engineering for Data Science

- Lead: Günter Kniesel
  - Hamed S. Yazdi
  - Shimaa Khalid
Organisational Matters

Overview

- Distributed Big Data Analysis consists of two modules: lecturers/lab + project
- Mailing list: sign-up sheet
- Lecture notes will be provided at: https://sewiki.iai.uni-bonn.de/teaching/labs/dbda/2017/start
- Changes will be announced at the website and via mailing list
Laboratory Outline

Lecture Timings: Tuesday 10:30 - 14:00

- April 18: Motivation
- **April 25: No Class**
- May 2: Spark Fundamentals I
- May 9: Spark Fundamentals II
- May 16: RDF, Knowledge graphs and SANSA
- May 23: Project Allocation
- May 30 - July 18 Lab Project work
- .
- .
- July 18-25: Project Presentation
Grading

- Final Project.
  - Code,
  - Documentation,
  - Presentation
Big Data

- No Single Definition
- Big data is a term for data sets that are so large or complex that traditional data processing application softwares are inadequate to deal with them.
- Need to be analysed computationally to reveal patterns, trends, and associations, especially relating to human behaviour and interactions.
Big Data

- Every day, there are 2.5 quintillion bytes of data created - so much that 90% of the data in the world today has been created in the last two years alone.
- It is not only about data collection, or data querying, its is about learning from this tremendous data for informed decision making.
Big Data = Transactions + Interactions + Observations

Petabytes → Terabytes → Gigabytes → Megabytes

Increasing Data Variety and Complexity

Source: Contents of above graphic created in partnership with Teradata, Inc.
Harnessing Big Data

- **OLTP**: Online Transaction Processing (DBMSs)
- **OLAP**: Online Analytical Processing (Data Warehousing)
- **RTAP**: Real-Time Analytics Processing (Big Data Architecture & technology)
Why ‘BigData’ is so important?

- It’s relevance have been increased drastically and **Big Data Analytics** is an emerge field to explore.

https://www.google.com/trends/explore?date=all&q=%22big%20data%22
Big data is more real-time in nature than traditional DW applications.

Traditional DW architectures (e.g. Exadata, Teradata) are not well-suited for big data apps.

Shared nothing, massively parallel processing, scale out architectures are well-suited for big data apps.
Zoo of Big Data Animals

Big Data Landscape

Source: Big Data Universe v3. Matt Turck, Jim Hao & FirstMark Capital
## Big Data Ecosystem

### File system
- HDFS, NFS

### Resource manager
- Mesos, Yarn

### Coordination
- Zookeeper

### Data Acquisition
- Apache Flume, Apache Sqoop

### Data Stores
- MongoDB, Cassandra, Hbase, Project Voldemort

### Data Processing
- **Frameworks**
  - Hadoop MapReduce, Apache Spark, Apache Storm, Apache Flink
- **Tools**
  - Apache Pig, Apache Hlve
- **Libraries**
  - SparkR, Apache Mahout, MLib, etc

### Data Integration
- **Message Passing**
  - Apache Kafka
- **Managing data heterogeneity**
  - SemaGrow, Strabon
Big Data Vs

**Volume**
- 40 Zettabytes: 43 trillion gigabytes of data will be created by 2020, an increase of 300 times from 2005.
- 6 billion people have cell phones.
- World population: 7 billion.

**Scale of Data**
- Most companies in the U.S. have at least 100 terabytes (100,000 gigabytes) of data stored.

**Velocity**
- The New York Stock Exchange captures 1 TB of trade information during each trading session.
- Modern cars have close to 100 sensors that monitor items such as fuel level and tire pressure.

**Analysis of Streaming Data**
- By 2016, it is projected there will be 18.9 billion network connections — almost 2.5 connections per person on earth.

**The FOUR V’s of Big Data**
- It’s estimated that 2.5 quintillion bytes (2.5 quadrillion gigabytes) of data are created each day.

**Variety**
- As of 2011, the global size of data in healthcare was estimated to be 150 exabytes (161 billion gigabytes).
- 400 million tweets are sent per day by about 200 million monthly active users.
- 30 billion pieces of content are shared on Facebook every month.

**Different Forms of Data**
- By 2014, it’s anticipated there will be 420 million wearable, wireless health monitors.

**4 Billion+ Hours of Video**
- Are watched on YouTube each month.

**Veracity**
- Poor data quality costs the US economy around $3 trillion a year.

**Uncertainty of Data**
- In one survey, 27% of respondents were unsure of how much of their data was inaccurate.

**Accuracy**
- 1 in 3 business leaders don’t trust the information they use to make decisions.

**Sources:** McKinsey Global Institute, Twitter, Cisco, Gartner, EMC, SAS, IBM, MEPTEC, QAS

**By 2015**
- 4.4 million IT jobs will be created globally to support big data, with 1.9 million in the United States.

**Conclusion**
- Depending on the industry and organization, big data encompasses information from multiple internal and external sources such as transactions, social media, enterprise content, sensors, and mobile devices. Companies can leverage data to adapt their products and services to better meet customer needs, optimize operations and infrastructure, and find new sources of revenue.
Data Flow Models

Data flow vs. traditional network programming

- Message Passing Interface
- Programmer managed data locality and Code
- Hardware failure
- Slow Hardware
- Data Communication is slow over the network
Map reduce

First popular data flow model

- compromises the programmer’s ability to control functionality in order to handle the problems
- In the Map-Reduce model, the user provides two functions (map and reduce),
- Map() must output key-value pairs, and as a result
- reduce() is guaranteed that its input is partitioned by key across machines
Drawbacks of Mapreduce

- Force data analysis workflow into a map and a reduce phase
  - You might need
    - Join
    - Filter
    - Sample
  - Complex workflows that do not fit into map/Reduce

- Mapreduce relies on reading data from disk
  - Performance bottleneck
  - Especially for iterative algorithms
Solution:

- A tool that works in the same environment
- Interactive shell
- Compatible with the existing environment
- No need to replace the stack, but replace map--reduce
The Data flow engines are useful because

- They save time and effort on the part of the programmer by providing abstractions
- They scale well
- Many algorithms have been redesigned to fit into the paradigm
- They have become common on clusters
Spark

- Rich programming interface
- More than 20 efficient, distributed operations
- Easier for data scientists to create custom data analysis pipeline in Spark
- Data can be cached in Memory
- ML Pipelines have gained 10 to 100 times speedup
Spark

- Fast
  - in-memory computations
- General-purpose,
  - Mapreduce jobs, Iterative jobs, queries, Streaming
- easy to use
  - Simple APIS for scala, Python, Java
  - General:
    - Spark runs on Hadoop clusters such as Hadoop YARN or Apache Mesos,
    - As a standalone with its own scheduler.
Spark

Spark provides

- parallel distributed processing,
- fault tolerance on commodity hardware,
- Scalability

Spark adds to the concept by

- low latency,
- Aggressively cached in-memory distributed Computing,
- high level APIs
- stack of high level tools

This saves time and money.
Apache Spark

- **Apache Spark** is an open-source in-memory data processing engine for **distributed processing**. It provides APIs in Scala, Java, Python and R which try to simplify the programming complexity by introducing the abstraction of Resilient Distributed Datasets (RDD), i.e. a logical collection of data partitioned across machines.

- On top of its core, Spark provides 4 libraries:
  - [Spark SQL](#) for SQL and unstructured data processing
  - [Spark Streaming](#) stream processing
  - [MLlib](#) machine learning algorithms
  - [GraphX](#) graph processing.
Apache Flink

- **Apache Flink** is an open-source stream processing framework for distributed, high-performing, always-available, and accurate data streaming applications.

- Flink provides:
  - DataStream API for unbounded streams
  - DataSet API for static data
  - Table API & SQL with a SQL-like expression language.

- Flink also bundles libraries for domain-specific use cases:
  - CEP, a complex event processing library,
  - Machine Learning library, and
  - Gelly, a graph processing API and library.

- Flink embraces the stream as abstraction to implement its dataflow.
Big Data Projects

- Related to this lab
  - Big Data Europe
  - SANSA
Key Observation From BDE

Heterogeneity AKA Variety
A Single View to the Customer
Over the last years, the size of the **Semantic Web** has increased and several large-scale datasets were published.

Now days **hadoop** ecosystem has become a standard for **BigData** applications.

We use this infrastructure for Semantic Web as well.
**SANSA** - Semantic Analytics Stack

- **Knowledge Graphs** become increasingly popular (via graph databases and semantic technologies)

- **But:**
  - Most ML algorithms work on simple feature input (not graphs)
  - Advanced algorithms for knowledge graphs usually do not scale horizontally

- **SANSA** is a suite of APIs for distributed reading, querying, inferencing and analysing of RDF knowledge graphs

Scala

- A functional Programming Language that follows a rich concise syntax
- Multiparadigm (you can also write OO code)
- Interoperates with Java
- Simplifies the concurrent programming, (No threads/Locks)
- Static typing, Immutable objects, Closure, elegant pattern matching
- Strongly Typed language, with functional and concurrency support
Scala

- Conciseness (100 lines in Java = 10-15 lines of code in scala)
- Sometimes harder to understand
  - Currying,
  - Function passing,
  - High order functions
Scala

- Everything is an object
  - Primitive types e.g. numbers, bool
  - Functions
- Numbers are objects:
  - 1+2*3 -> (1).+(2).*3)
- Functions are objects:
  - Pass functions as arguments
  - Store them in variables
  - Return them from other functions
- Function declaration
  - Def functionName ([list of parameters]:[return type])
Scala REPL

- Read evaluate and Print Loop.
- Interactive shell session,
- In interactive mode, the REPL reads expressions at the prompt, wraps them in an executable template, and then compiles and executes the result.
Basics

```
>val msg = "Hello, world!"
msg: java.lang.String = Hello, world!
>println(msg)
msg = "Goodbye cruel world!"
>def max(x: Int, y: Int): Int = if (x < y) y else x
max: (Int,Int)Int
>max(3, 5)
unnamed: Int = 5
```
Anonymous functions:

```scala
val plusOne = (x: Int) => x + 1
(x: Int) => x*x*x
(x: Int, y: Int) => x + y
```

Alternatively

```scala
{def f (x: Int, y: Int) => x + y; f}
```

```scala
println(plusOne(0)) // Prints: 1
anotherFunction(plusOne(20)) // Prints: 21
```
High Order Functions

- First order functions:
  - Acts on simple data types

- High order Function:
  - Act on other functions
  - Sum function in the previous example is
  - High order function
High order Functions and Tail recursion

```python
def sumInts(a: Int, b: Int): Int = 
    if (a > b) 0 else a + sumInts(a + 1, b)
```

```python
def sum(f: Int => Int, a: Int, b: Int) = {
    def loop(a: Int, acc: Int): Int = 
        if (a > b) acc 
        else loop(a + 1, f(a) + acc)
    loop(a, 0)
}

sum(x => x * x, 3, 5)
```

- http://www.artima.com/scalazine/articles/steps.html
- http://www.artima.com/scalazine/articles/steps.html
- http://sda.cs.uni-bonn.de/
- https://github.com/SANSA-Stack
- https://github.com/big-data-europe
- https://github.com/SmartDataAnalytics