Milestone 1 Instruction
Milestone 1

- Each team will present on either February 3rd or February 10th
- Prepare about 10-12 mins of presentation (and expect 3-5 mins of Q&A)
- All teams will submit a written Milestone 1 report on February 10th.

- Key elements in Milestone 1 presentation:
  - Task Goal:
    - What do you want to achieve?
    - Why is the research and development important?
    - Is this a new topic that considers challenges of
      - Volume
      - Velocity
      - Variety
    - Does this topic try to incorporate multi-discipline knowledge?
Milestone 1

○ Literature Survey:
  ○ What are the prior arts? What related works were done before?
  ○ Which research publications, tools and products may be utilized to build upon them to achieve the goal?

○ Methodology:
  ○ What types of novel algorithms I shall try to invent and implement?
  ○ Where will you try to gather the data — Existing dataset? Self-collected dataset? Live dataset?
Milestone 1

○ **System:**
  ○ What will your final system look like — potential backend components and interaction with front end?
  ○ How will you create visualization and user interface to help users consume your analysis outcome?

○ **Timeline:**
  ○ What may you achieve in Milestones 2 and 3, and in the Final Project?

Note — it’s good to think about what you want to achieve as complete as possible and study what other people have done. But, like all projects, everything can change when you make progress. Please do not afraid of making bold assumptions and attempt!!
Big Data Foundations
Key Open Source Big Data Foundations

Apache Hadoop

Spark

Lightning-fast unified analytics engine

kafka

A distributed streaming platform
Other Important Foundations (visualization and web servers)
Reference Foundation (AI Platform)
Where to store data?
How to get data in and out?
How to manage access of data?
Hadoop

Key Layers:
• Storage
• Compute Platform
• Shared Services
• Compute Frameworks
Hadoop Storage is widely used
Hadoop Distributed File System (HDFS)

HDFS Architecture

Metadata ops

Namenode

Metadata (Name, replicas, ...): /home/foo/data, 3, ...

Block ops

Datanodes

Replication

Rack 1

Write

Rack 2

Client

Datanodes

Blocks

Read

Client

http://hortonworks.com/hadoop/hdfs/
Basic Data Storage Operations on HDFS

- Hadoop is designed to work best with a modest number of extremely large files.
- Average file sizes ➔ larger than 500MB.

- Write Once, Read Often model.
- Content of individual files cannot be modified, other than appending new data at the end of the file.

- What we can do:
  - Create a new file
  - Append content to the end of a file
  - Delete a file
  - Rename a file
  - Modify file attributes like owner
HDFS blocks

- File is divided into blocks (default: 64MB) and duplicated in multiple places (default: 3)

- Dividing into blocks is normal for a file system. E.g., the default block size in Linux is 4KB. The difference of HDFS is the scale.
- Hadoop was designed to operate at the petabyte scale.
- Every data block stored in HDFS has its own metadata and needs to be tracked by a central server.
There are three principles of systems design in reliability engineering which can help achieve high availability.

1. Elimination of single points of failure. This means adding redundancy to the system so that failure of a component does not mean failure of the entire system.
2. Reliable crossover. In redundant systems, the crossover point itself tends to become a single point of failure. Reliable systems must provide for reliable crossover.
3. Detection of failures as they occur. If the two principles above are observed, then a user may never see a failure – but the maintenance activity must.

### Availability vs. Downtime

<table>
<thead>
<tr>
<th>Availability %</th>
<th>Downtime per year</th>
<th>Downtime per month</th>
<th>Downtime per week</th>
<th>Downtime per day</th>
</tr>
</thead>
<tbody>
<tr>
<td>90% (&quot;one nine&quot;)</td>
<td>36.53 days</td>
<td>73.05 hours</td>
<td>16.80 hours</td>
<td>2.40 hours</td>
</tr>
<tr>
<td>99% (&quot;two nines&quot;)</td>
<td>3.65 days</td>
<td>7.31 hours</td>
<td>1.68 hours</td>
<td>14.40 minutes</td>
</tr>
<tr>
<td>99.9% (&quot;three nines&quot;)</td>
<td>8.77 hours</td>
<td>43.83 minutes</td>
<td>10.08 minutes</td>
<td>1.44 minutes</td>
</tr>
<tr>
<td>99.99% (&quot;four nines&quot;)</td>
<td>52.60 minutes</td>
<td>4.38 minutes</td>
<td>1.01 minutes</td>
<td>8.64 seconds</td>
</tr>
</tbody>
</table>
HDFS blocks

- Replication patterns of data blocks in HDFS.

- When HDFS stores the replicas of the original blocks across the Hadoop cluster, it tries to ensure that the block replicas are stored in different failure points.
Interaction between HDFS components
Several useful commands for HDFS

- All hadoop commands are invoked by the bin/hadoop script.

  ```
  hadoop [-conf confdir] [COMMAND] [GENERIC_OPTIONS] [COMMAND_OPTIONS]
  ```

- `% hadoop fsck / -files -blocks:

  list the blocks that make up each file in HDFS.

- For HDFS, the schema name is hdfs, and for the local file system, the schema name is file.

- A file or director in HDFS can be specified in a fully qualified way, such as:
  
  hdfs://namenodehost/parent/child  or hdfs://namenodehost

- The HDFS file system shell command is similar to Linux file commands, with the following
general syntax:  `hadoop hdfs -file_cmd`

- For instance mkdir runs as:
  
  `$hadoop hdfs dfs -mkdir /user/directory_name`
Several useful commands for HDFS -- II

For example, to create a directory named “joanna”, run this mkdir command:

$ hadoop hdfs dfs -mkdir /user/joanna

Use the Hadoop put command to copy a file from your local file system to HDFS:

$ hadoop hdfs dfs -put file_name /user/login_user_name

For example, to copy a file named data.txt to this new directory, run the following put command:

$ hadoop hdfs dfs -put data.txt /user/joanna

Run the ls command to get an HDFS file listing:

$ hadoop hdfs dfs -ls .
Apache Hive is a data warehouse software project built on top of Apache Hadoop for providing data query and analysis. Hive gives a SQL-like interface to query data stored in various databases and file systems that integrate with Hadoop.

HiveQL Example — Word Count:

1. DROP TABLE IF EXISTS docs;
2. CREATE TABLE docs (line STRING);
3. LOAD DATA INPATH 'input_file' OVERWRITE INTO TABLE docs;
4. CREATE TABLE word_counts AS
5. SELECT word, count(1) AS count FROM
6. (SELECT explode(split(line, '\s')) AS word FROM docs) temp
7. GROUP BY word
8. ORDER BY word;

Key Characteristics: (1) Schema on Read (vs Schema on Write of Traditional DB); (2) ACID — Atomicity, Consistency, Isolation, and Durability.
MapReduce example

The overall MapReduce word count process

Input
- Deer Bear River
- Car Car River
- Deer Car Bear

Splitting
- Deer Bear River
- Car Car River
- Deer Car Bear

Mapping
- Deer, 1
- Bear, 1
- River, 1
- Car, 1
- Car, 1
- Car, 1
- Deer, 1
- Deer, 1
- River, 1
- River, 1

Shuffling
- Bear, 1
- Bear, 1
- Car, 1
- Car, 1
- Car, 1
- Deer, 1
- Deer, 1
- River, 1
- River, 1

Reducing
- Bear, 2
- Car, 3
- Deer, 2

Final result
- Bear, 2
- Car, 3
- Deer, 2
- River, 2

http://www.alex-hanna.com
Data Lake

A data lake provides a way to centrally apply and enforce authentication, authorization, and audit policies across multiple ephemeral workload clusters.

<table>
<thead>
<tr>
<th>Term</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Data lake</td>
<td>Runs Ranger, which is used for configuring authorization policies and is used for audit capture. Runs Hive Metastore, which is used for data schema.</td>
</tr>
<tr>
<td>Workload clusters</td>
<td>The clusters that get attached to the data lake to run workloads. This is where you run workloads such as Hive via JDBC.</td>
</tr>
</tbody>
</table>

“Working with Data Lakes”, Cloudera
### Data Lake — Components

<table>
<thead>
<tr>
<th>Component</th>
<th>Technology</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Schema</td>
<td>Apache Hive</td>
<td>Provides Hive schema (tables, views, and so on). If you have two or more workloads accessing the same Hive data, you need to share schema across these workloads.</td>
</tr>
<tr>
<td>Policy</td>
<td>Apache Ranger</td>
<td>Defines security policies around Hive schema. If you have two or more users accessing the same data, you need security policies to be consistently available and enforced.</td>
</tr>
<tr>
<td>Audit</td>
<td>Apache Ranger</td>
<td>Audits user access and captures data access activity for the workloads.</td>
</tr>
<tr>
<td>Governance</td>
<td>Apache Atlas</td>
<td>Provides metadata management and governance capabilities.</td>
</tr>
<tr>
<td>Directory</td>
<td>LDAP/AD</td>
<td>Provides an authentication source for users and a definition of groups for authorization.</td>
</tr>
<tr>
<td>Gateway</td>
<td>Apache Knox</td>
<td>Supports a single workload endpoint that can be protected with SSL and enabled for authentication to access to resources.</td>
</tr>
</tbody>
</table>

Apache Ambari:
- Provision a Hadoop Cluster
- Manage a Hadoop Cluster
- Monitor a Hadoop Cluster

Apache Zeppelin:
- Web-based notebook for data analytics and collaborative documents
How do I process the data?
How do I execute machine learning from the data?
How do I tell people my analytics results?
Spark

Lightning-fast unified analytics engine

Spark SQL structured data
Spark Streaming real-time
MLib machine learning
GraphX graph processing

Spark Core

Standalone Scheduler
YARN
Mesos
Spark MLlib

Includes:

- **ML Algorithms**: common learning algorithms such as classification, regression, clustering, and collaborative filtering
- **Featurization**: feature extraction, transformation, dimensionality reduction, and selection
- **Pipelines**: tools for constructing, evaluating, and tuning ML Pipelines
- **Persistence**: saving and load algorithms, models, and Pipelines
- **Utilities**: linear algebra, statistics, data handling, etc.

**MLlib: Main Guide**

- Basic statistics
- Data sources
- Pipelines
- Extracting, transforming and selecting features
- Classification and Regression
- Clustering
- Collaborative filtering
- Frequent Pattern Mining
- Model selection and tuning
- Advanced topics
Spark MLlib Basic Statistics

Includes:
- Correlation
- Hypothesis testing
- Summarizer

Example of Calculating Correlation of Time Sequences:

```python
from pyspark.ml.linalg import Vectors
from pyspark.ml.stat import Correlation

data = [(Vectors.sparse(4, [(0, 1.0), (3, -2.0)])),
        (Vectors.dense([4.0, 5.0, 0.0, 3.0]))],
        (Vectors.dense([6.0, 7.0, 0.0, 8.0]))],
        (Vectors.sparse(4, [(0, 9.0), (3, 1.0)]))]
df = spark.createDataFrame(data, ['features'])

r1 = Correlation.corr(df, 'features').head()
print("Pearson correlation matrix:
      ") + str(r1[0])

r2 = Correlation.corr(df, 'features', 'spearman').head()
print("Spearman correlation matrix:
      ") + str(r2[0])
```

MLlib: Main Guide

- Basic statistics
- Data sources
- Pipelines
- Extracting, transforming and selecting features
- Classification and Regression
- Clustering
- Collaborative filtering
- Frequent Pattern Mining
- Model selection and tuning
- Advanced topics
Spark MLlib Features

Includes:

- Extraction: Extracting features from “raw” data
- Transformation: Scaling, converting, or modifying features
- Selection: Selecting a subset from a larger set of features
- Locality Sensitive Hashing (LSH): This class of algorithms combines aspects of feature transformation with other algorithms.

• Feature Extractors
  - TF-IDF
  - Word2Vec
  - CountVectorizer
  - FeatureHasher

• Feature Selectors
  - VectorSlicer
  - RFormula
  - ChiSqSelector

• Feature Transformers
  - Tokenizer
  - StopWordsRemover
  - n-gram
  - Binarizer
  - PCA
  - PolynomialExpansion
  - Discrete Cosine Transform (DCT)
  - StringIndexer
  - IndexToString
  - OneHotEncoder (Deprecated since 2.3.0)
  - OneHotEncoderEstimator
  - VectorIndexer
  - Interaction
  - Normalizer
  - StandardScaler
  - MinMaxScaler
  - MaxAbsScaler
  - Bucketizer
  - ElementwiseProduct
  - SQLTransformer
  - VectorAssembler
  - VectorSizeHint
  - QuantileDiscretizer
  - Imputer
Spark MLlib Supervised Machine Learning Algorithms

Classification
- Logistic regression
  - Binomial logistic regression
  - Multinomial logistic regression
- Decision tree classifier
- Random forest classifier
- Gradient-boosted tree classifier
- Multilayer perceptron classifier
- Linear Support Vector Machine
- One-vs-Rest classifier (a.k.a. One-vs-All)
- Naive Bayes

Regression
- Linear regression
- Generalized linear regression
  - Available families
- Decision tree regression
- Random forest regression
- Gradient-boosted tree regression
- Survival regression
- Isotonic regression
Spark MLlib UnSupervised Machine Learning & Recommendation Algorithms

Clustering:
- K-means
  - Input Columns
  - Output Columns
- Latent Dirichlet allocation (LDA)
- Bisecting k-means
- Gaussian Mixture Model (GMM)
  - Input Columns
  - Output Columns

Collaborative Filtering:
- Explicit vs. implicit feedback
- Scaling of the regularization parameter
- Cold-start strategy
Spark MLlib Model Selection and Tuning

- Model selection (a.k.a. hyperparameter tuning)
- Cross-Validation
- Train-Validation Split
Kafka

Kafka is a distributed streaming platform that is used publish and subscribe to streams of records.
Kafka is used for fault tolerant storage.
Kafka replicates topic log partitions to multiple servers.
Kafka is designed to allow your apps to process records as they occur.
Kafka is fast and uses IO efficiently by batching and compressing records.
Kafka is used for decoupling data streams.
Kafka is used to stream data into data lakes, applications, and real-time stream analytics systems.
Where to use Kafka

Kafka streaming architecture diagram
Zeppelin (early stage)

Data Ingestion; Data Discovery; Data Analytics; Data Visualization & Collaboration
Webservers

Apache (http server) — the oldest and most popular web server exists in every Linux machine, including MacOS machines.

— display webpages of those files reside in its http root directory

```python
from flask import Flask, escape, request

app = Flask(__name__)

@app.route('/

def hello():
    name = request.args.get("name", "World")
    return f'Hello, {escape(name)}!
```
## D3 Visualization (via Javascript) Examples

### Visual Index

<table>
<thead>
<tr>
<th>Box Plots</th>
<th>Bubble Chart</th>
<th>Bullet Charts</th>
<th>Calendar View</th>
</tr>
</thead>
<tbody>
<tr>
<td><img src="image1.png" alt="Box Plots" /></td>
<td><img src="image2.png" alt="Bubble Chart" /></td>
<td><img src="image3.png" alt="Bullet Charts" /></td>
<td><img src="image4.png" alt="Calendar View" /></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Non-contiguous Cartogram</th>
<th>Chord Diagram</th>
<th>Dendrogram</th>
<th>Force-Directed Graph</th>
</tr>
</thead>
<tbody>
<tr>
<td><img src="image5.png" alt="Non-contiguous Cartogram" /></td>
<td><img src="image6.png" alt="Chord Diagram" /></td>
<td><img src="image7.png" alt="Dendrogram" /></td>
<td><img src="image8.png" alt="Force-Directed Graph" /></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Circle Packing</th>
<th>Population Pyramid</th>
<th>Stacked Bars</th>
<th>Streamgraph</th>
</tr>
</thead>
<tbody>
<tr>
<td><img src="image9.png" alt="Circle Packing" /></td>
<td><img src="image10.png" alt="Population Pyramid" /></td>
<td><img src="image11.png" alt="Stacked Bars" /></td>
<td><img src="image12.png" alt="Streamgraph" /></td>
</tr>
</tbody>
</table>
D3 Visualization (via Javascript) Examples

- Sunburst
- Node-Link Tree
- Treemap
- Voronoi Diagram
- Hierarchical Edge Bundling
- Voronoi Diagram
- Bubble Map
- Parallel Coordinates
- Scatterplot Matrix
- Zoomable Pack Layout
- Hierarchical Bars
- Epicyclical Gears
## D3 Visualization (via Javascript) Examples

<table>
<thead>
<tr>
<th>Collision Detection</th>
<th>Collapsible Force Layout</th>
<th>Force-Directed States</th>
<th>Versor Dragging</th>
</tr>
</thead>
<tbody>
<tr>
<td><img src="image1.png" alt="Collision Detection" /></td>
<td><img src="image2.png" alt="Collapsible Force Layout" /></td>
<td><img src="image3.png" alt="Force-Directed States" /></td>
<td><img src="image4.png" alt="Versor Dragging" /></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Choropleth</th>
<th>Collapsible Tree Layout</th>
<th>Zoomable Treemap</th>
<th>Zoomable Icicle</th>
</tr>
</thead>
<tbody>
<tr>
<td><img src="image5.png" alt="Choropleth" /></td>
<td><img src="image6.png" alt="Collapsible Tree Layout" /></td>
<td><img src="image7.png" alt="Zoomable Treemap" /></td>
<td><img src="image8.png" alt="Zoomable Icicle" /></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Zoomable Area Chart</th>
<th>Drag and Drop Collapsible Tree Layout</th>
<th>Radial Cluster Layout</th>
<th>Sankey Diagram</th>
</tr>
</thead>
<tbody>
<tr>
<td><img src="image9.png" alt="Zoomable Area Chart" /></td>
<td><img src="image10.png" alt="Drag and Drop Collapsible Tree Layout" /></td>
<td><img src="image11.png" alt="Radial Cluster Layout" /></td>
<td><img src="image12.png" alt="Sankey Diagram" /></td>
</tr>
</tbody>
</table>
Human brain is a graph of 100B nodes and 700T edges.
AI Platform

Ardi’s 8 Components

- Graph Database
- Relational Database
- Graph Analytics
- Feature Engineering
- Causality Modeling
- Behavior Prediction

Sense
- Deep Language Understanding
- Deep Video Understanding

Learning
- Machine Learning
- Deep Learning
- Autonomous Model Optimization
- Production Workflow

Pipeline
- Action Strategy Simulation

Strategy

Explanation

- Visualization
- ML Explanations

Reasoning

Database

Analytics
Graph computing emerged as THE foundation to solve complex AI issues

**RDF / Property Graph**

**Activity Graph**

**Collective Graph**

**Graph Database**

**Behavior Analysis**

**Relationship Analysis**

<table>
<thead>
<tr>
<th>subject</th>
<th>predicate</th>
<th>object</th>
</tr>
</thead>
<tbody>
<tr>
<td>Charles Flint</td>
<td>born</td>
<td>&quot;1850&quot;</td>
</tr>
<tr>
<td>Charles Flint</td>
<td>died</td>
<td>&quot;1934&quot;</td>
</tr>
<tr>
<td>Charles Flint</td>
<td>founder</td>
<td>IBM</td>
</tr>
<tr>
<td>IBM</td>
<td>HQ</td>
<td>&quot;Armonk&quot;</td>
</tr>
<tr>
<td>IBM</td>
<td>employees</td>
<td>433,362</td>
</tr>
<tr>
<td>IBM</td>
<td>industry</td>
<td>Software</td>
</tr>
<tr>
<td>IBM</td>
<td>industry</td>
<td>Hardware</td>
</tr>
<tr>
<td>IBM</td>
<td>industry</td>
<td>Services</td>
</tr>
</tbody>
</table>

Improvement on accuracy: 2x-3x by context / relation analysis, 10x+ by reasoning / behavior prediction
Information / Transaction Flows
How do kids grow intelligence?

“Airplane”

“Grandma”

“Grandma is in Taiwan”

“Auntie is also in Taiwan”

“I like grandma”

“I like grandpa”

“I like grandma and grandpa”

The boy said:
Why is Graph Computing the core of AI?

How does a Graph Computing mechanism reason and store data?

Graph platform works like the human mind, connecting the dots when comprehending.
Cognitive Recognition

- Recognize an event from sensing -> feature extortion -> object/scene/action detection -> semantics -> cognition inference

Event: Attempting a Board Trick
Example of Multi-Modality Multi-Layer Machine Understanding

- **Sensor Layer**: HR records, Travel records, Badge/Location records, Phone records, Mobile records
- **Feature Layer**: Transmitted images, speech content, video content
- **Concept Layer**: Observations
- **Semantics Layer**: Hidden states
- **Cognition Layer**: Layered understanding

- **Observations**
- **Hidden States**
Building towards “Future AI”

Today’s "AI"
- Processing
- Classification
- Individuals
- Built on Open Sources

Future AI
- Perception
- Reasoning
- Strategy
- Context
- Built on Graph Computing
Graphen’s Ardi Platform makes full-brain AI possible

- **Machine Cognition:**
  - Robot Cognition Tools
  - Feeling
  - Robot-Human Interaction

- **Machine Reasoning:**
  - Bayesian Networks
  - Game Theory Tools

- **Machine Learning:**
  - ML and Deep Learning
  - Autonomous Imperfect Learning

- **Advanced Visualization:**
  - Dynamic and Interactive Viz.
  - Big Data Viz.

- **Graph Analytics:**
  - Network Analysis
  - Flow Prediction

- **Graph Database:**
  - Distributed Native Database

Most of other existing “AI” technology is only one of key fundamental components.
Enterprise-Level AI Platform

• Terabyte-sized native GraphDB, supports trillion of vertices and edges

• ACID-compliant and distributed Graph database and analytics

• Asynchronous job scheduling (both Autonomous ML and GraphDB)

• Scalable, distributed Analytics, modular and expandable through plugins

• Cluster, Replication and High-Availability with disaster recovery

• Error and event Logging, Monitoring, Backup and Recovery
Example — Using AI Platform to Build Finance Products

Graphen Finance AI Solutions
- Core-Banking Monitoring
- Non-Performing Loans Prediction
- Anti Money Laundering
- Fraud Detection
- Regulation Reasoning & Compliance
- Market Intelligence
- AI Trader

Graphen AI Financial Industry Platform

Multimodal Analysis
- Behavioral Analysis
- Network Analysis
- Time-Series Analysis
- Flow Analysis

Risk Modeling
- Anomaly Detection
- Risk Assessment
- Bayesian Inference
- Risk Propagation

Graphen Advanced AI Platform

Computation Engine
- Graph Analytics
- Statistical Computing
- Machine Learning
- Machine Reasoning

Visualization Engine
- Graph Vis
- Statistical Charts
- Machine Learning Vis
- Cognitive Vis

Data Engine
- Graph Database
- Index
- File Storage
- Other Data Storage
- Data Ingestion
- Data Retrieval
Graph Computing (GraphDB & Analytics) + Autonomous Machine Learning (AMOS) + Bayesian Network
An Example Graphen Ardi Data Pipeline

*AMOS: Autonomous Model Optimizer System
Can you see fraud easily here?

<table>
<thead>
<tr>
<th>Company</th>
<th>Balance</th>
<th>Loans</th>
<th>Number of payment cycles</th>
<th>Majority Stakeholder</th>
<th>Phones sold</th>
<th>Buyers</th>
<th>VAT</th>
</tr>
</thead>
<tbody>
<tr>
<td>A</td>
<td>100</td>
<td>11</td>
<td>1</td>
<td>Paul</td>
<td>£10M</td>
<td>B</td>
<td>£1M</td>
</tr>
<tr>
<td>B</td>
<td>123</td>
<td>43</td>
<td>0</td>
<td>Sam</td>
<td>£10M</td>
<td>C</td>
<td>£1M</td>
</tr>
<tr>
<td>E</td>
<td>100</td>
<td>2000000</td>
<td>1</td>
<td>George</td>
<td>£10M</td>
<td>...</td>
<td></td>
</tr>
<tr>
<td>C</td>
<td>345</td>
<td>45</td>
<td>10</td>
<td>Nicole</td>
<td>£10M</td>
<td>D</td>
<td></td>
</tr>
<tr>
<td>D</td>
<td>65</td>
<td>1200</td>
<td>1</td>
<td>Paul</td>
<td>£10M</td>
<td>E</td>
<td>£1M</td>
</tr>
<tr>
<td>...</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

**Q:** In RDBMS, given this data can we say if other companies are as good as E?
How the carousel fraud works in 4 steps

**Step 1**
Company A (US) sells to Company B (Europe) €10M worth of phones.

**Step 2**
Company B sells the phones to company C. It charges €10M + €1M for the VAT.

**Step 3**
Company B sells the phones to company D (US) and claims a VAT refund.

**Step 4**
The directors of A and D disappear with €2M in stolen taxes.
Stakeholders in fraud case

- **Paul (Person)**: Director of Company A, with country: USA, age: 53, criminal_status: unknown.
- **Nicole (Person)**: Director of Company C, with country: Italy, age: 29, criminal_status: unknown.
- **Company A (Company)**: Sells to Company B, type: LLC, creation_date: 08/13/1983.
- **Company B (Company)**: Sells to Company C, type: SRL, creation_date: 04/09/1984, contains item: phones.
- **Company C (Company)**: Collects VAT, creation_date: 19/04/1990, contains item: amount: 1M.

**Relationships:**
- **DIRECTOR_OF** from Paul to Company A, from Nicole to Company C.
- **SELLS_TO** from Company A to Company B, from Company B to Company C.
- **COLLECTS_VAT** from Company B to Company C.
- **PARENT_OF** from Company A to Company B.
Real world graph representation

Vertices are People, Companies, Accounts, Address, Email

Edges : Knows, Works At, Owns, Transfer, Lives At, etc.

Questions:
1) Find accounts/People sharing common address, phone, email, etc.
2) Infer indirect relationships
3) Estimate risk of fraud ring
Graphen Biomedical Knowledge Graph

Large-scale datasets
Uncertainty modeling
Complex relationships….

• 3.5 million biomedical concepts with 7.5 million relationships among them.
Growing List of Graph Analytics

- Shortest Path
- Connected
- Centrality
- Cycles
- Paths
- Cliques
- Egonet
- K-core
- Community
- Link Predictions

- MST - Minimum Spanning Tree
- Page Rank
- Sim Rank
- Strongly Connected Components
- Select Vertices
- Chaining of analytics:
  - Map
  - Reduce
  - Aggregated
  - Store

- Images
  - Image Classification and feature detection (TBD)
Deployments/Interactions

• Either using the local deployments or the cloud version:
  • Local deployments are through docker images.
  • Cloud version can directly use Ardi machine on AWS.
• Many ways to use:
  • Web UI with Graphen Viz through RestAPI
  • Rest client, with an wrapper, programs in Python but communicates through Rest
Local version

- tar zxf graphdb-client-master.tar.gz
- cd graphdb-client
- Run docker-run.sh to start the GraphDB server, it creates ~/graphdb-rest/csv and ~/graphdb-rest/data and mapped to inside docker, you may change the host directories through the docker run command.
- graphdb-client/tutorial has this Jupyter Notebook tutorial, run jupyter-notebook from the tutorial directory.
- The input csv files to be loaded must be placed in ~/graphdb-rest/csv/ directory
- The ~/graphdb-rest/data/ will contain the graph files
- tests/ contains some unit tests that can also be viewed for examples of analytics
- projects/ has some examples of more complicated dataset and analytics, feel free to explore
Open tutorial.ipynb Notebook

Make sure to append path where client.py is

*client.py is Python2-based, Python3 version will be available early February.

In this example, graphDB host and Jupyter Notebook is ran from the same host.
Ingest vertex CSV file into graphDB

Vertex csv file content, located in ~/graphdb-rest/csv

```python
vertex_file_path = "vertex_2_files.csv"
has_header = 0
column_delimiter = ','
default_vertex_label = "2"
vertex_content_type = [
    {"position": [2, "STRING"]},
    {"age": [3, "DOUBLE"]},
]
column_number_map = {"vertex_id": 1,
    "properties": vertex_content_type}

g.load_table_vertex(file_path=vertex_file_path,
    has_header=has_header,
    column_delimiter=column_delimiter,
    default_vertex_label=default_vertex_label,
    column_number_map=column_number_map,
    column_header_map={},
    content_type=vertex_content_type,
    data_row_start=0,
    data_row_end=0,
    batch_size=10000)

Out[12]: '{"status": "success", "message": "5 vertices are added"}'
```
Ingest edge CSV file into graphDB

```python
edge_file_path = "edge_files.csv"  # name of csv file
has_header = 0  # whether has header or not
column_delimiter = ','  # sep of csv
default_source_label = "1"  # select the column for src_vertex Label
default_target_label = "1"  # select the column for tgt_vertex Label
default_edge_label = "owes"  # set default edge label

d['art_of_money'] = ['S', "DOUBLE"],  # set properties, list of dicts;
    # dict is of format {'prop_name': [column, dtype]}

edge_column_number_map = {
    "source_id": 1,  # select column for src_vertex
    "target_id": 2,  # select column for tgt_vertex
    "source_label": 3,  # select column for src_vertex_label
    "target_label": 4,  # select column for tgt_vertex_label
    "edge_label": 5,  # select column for edge label
    "properties": edge_column_number_map
}

g.load_table_edge(file_path=edge_file_path,
    has_header=has_header,
    column_delimiter=column_delimiter,
    default_source_label=default_source_label,
    default_target_label=default_target_label,
    default_edge_label=default_edge_label,
    column_header_map=(),  # csv header mapping
    column_number_map=edge_column_number_map,
    data_row_start=0,  # skip rows
    data_row_end=0,  # end by rows
    batch_size=1000)
```

Edge csv file content, located in ~/graphdb-rest/csv

```
roy,1,gabi,2,pays,1000
roy,1,andy,1,pays,50
andy,1,gabi,2,pays,25
daniel,2,bill,1,pays,100
daniel,2,jun,2,pays,200
big_danny,2,gabi,2,owes,50
roy,1,big_danny,2,owes,1
big_danny,2,tim,2,owes,1000
bill,1,tim,2,owes,10
gabi,2,daniel,2,owes,50
gabi,2,bill,1,owes,25
```
Read the graph schema and content as json

In [13]: g.get_schema(graph_name="test_graph") # check the schema

Out[13]: 

```
"status": "success", "schema": {
  "vertex_props": [{"prop_name": "position", "prop_type": "STRING"},
  {"prop_name": "age", "prop_type": "DOUBLE"}]
  },
  "edge_labels": [{"pays", "owes"}],
  "vertex_labels": [{"2", "1"}],
  "edge_props": [{"prop_name": "amt_of_money", "prop_type": "DOUBLE"}]
```

In [14]: g.print_graph(graph_name="test_graph") # print the graph as json string

Out[14]: 

```
"status": "success", "graph_type": 0, "statistics": {
  "num_edges": 12,
  "num_vertices": 9
},
"data": [{
  "source_label": "2",
  "target_label": "2",
  "target_id": "gabi",
  "label": "owes",
  "eid": "13835058055282163712",
  "source_id": "big_danny",
  "properties": {
    "amt_of_money": 50.0
  }
},
  {"source_label": "2",
  "target_label": "2",
  "target_id": "tim",
  "label": "owes",
  "eid": "13835058055282163714",
  "source_id": "big_danny",
  "properties": {
    "amt_of_money": 100.0
  }
},
  {"source_label": "1",
  "target_label": "bill",
  "label": "pays",
  "eid": "13835339530258874371",
  "source_id": "daniel",
  "properties": {
    "amt_of_money": 100.0
  }
},
  {"source_label": "2",
  "target_label": "2",
  "target_id": "jun",
  "label": "pays",
  "eid": "13835339530258874372",
  "source_id": "daniel",
  "properties": {
    "amt_of_money": 100.0
  }
},
  {"source_label": "2",
  "target_label": "2",
  "target_id": "danie1",
  "label": "owes",
  "eid": "13835058055282163716",
  "source_id": "gabi",
  "properties": {
    "amt_of_money": 50.0
  }
},
  {"source_label": "1",
  "target_label": "1",
  "target_id": "bill",
  "label": "owes",
  "eid": "13835058055282163717",
  "source_id": "gabi",
  "properties": {
    "amt_of_money": 25.0
  }
},
  {"source_label": "2",
  "target_label": "2",
  "target_id": "tim",
  "label": "owes",
  "eid": "13835058055282163718",
  "source_id": "gabi",
  "properties": {
    "amt_of_money": 50.0
  }
},
  {"source_label": "1",
  "target_label": "2",
  "target_id": "gabi",
  "label": "pays",
  "eid": "13835339530258874370",
  "source_id": "andy",
  "properties": {
    "amt_of_money": 25.0
  }
},
  {"source_label": "1",
  "target_label": "2",
  "target_id": "tim",
  "label": "owes",
  "eid": "13835058055282163715",
  "source_id": "bi
```
Visualize

```python
In [87]: # Here we have a simple code on how to visualize the graph
    # we need to make this into a function
    import networkx as nx
    import matplotlib.pyplot as plt

    G = nx.DiGraph()
    labels={}

    verts = graph_to_dict("data")["vertices"]
    for v in verts:
        #print v
        G.add_node(v["id"])
        for p in v["properties"]: 
            pname=p.keys()[0]
            pval=p.values()[0]
            #print pval,"::", pname
            print(pname)
            print(pval)
            print(v["id"])
            G.nodes[ v["id"] ][pname] = pval
            labels[v["id"]] = v["label"]

    edges = graph_to_dict("data")["edges"]
    for e in edges:
        G.add_edge(e["source_id"],e["target_id"])

    pos=nx.circular_layout(G) # positions for all nodes
    nx.draw_networkx_nodes(G, pos, cmap=plt.get_cmap('jet'),
                           node_size = 1000)
    nx.draw_networkx_labels(G, pos)
    nx.draw_networkx_edges(G, pos, edge_color='r', arrows=True)
    #nx.draw_networkx_edges(G, pos, edgelist=black_edges, arrows=False)
    plt.show()
```
Query edges and neighbors

In [25]:
```
g.get_edge(source_id="roy", source_label="1", target_id="gabi", target_label="2")
```

Out[25]:
```
{"status": "success", "statistics": {"num_edges": 1}, "data": {"edges": [{"source_label": "1", "target_label": "2", "target_id": "gabi", "label": "pays", "eid": "13835339350258874368", "source_id": "roy", "properties": [{"amt_of_money": 1000.0}]}]}}
```

In [26]:
```
g.get_edge(source_id="gabi", source_label="2", target_id="roy", target_label="1")
```

Out[26]:
```
{"status": "success", "statistics": {"num_edges": 0}, "data": {"edges": []}}
```

In [27]:
```
# find the neighbors

g.get_neighbor_out(vertex_id="roy", vertex_label="1")
```

Out[27]:
```
{"status": "success", "statistics": {"num_vertices": 3}, "data": {"vertices": [{"properties": [{"position": "boss", "age": 40.0}], "id": "gabi", "label": "2"}, {"properties": [{"position": "data_scientist", "age": 27.0}], "id": "andy", "label": "1"}, {"properties": [{"position": "old_man", "age": 90.0}], "id": "big_danny", "label": "2"}]}"
```
Example of Running analytics (Breadth-First Search)

```python
In [33]:
json_str = '"
   "vertices": [{"id": "roy", "label": "1"}]
   "depth": 1,
   "with_vertex_props": "true"
"

request_body = dict()
request_body["param"] = json_str

g.run_analytics(analytic_name="bfs", request_body=request_body)
```

```
Out[33]:
{"status": "success", "analytics": {}, "statistics": {"num_edges": 3, "num_vertices": 4}, "data": {"edges": [{"source_label": "1", "target_label": "2", "target_id": "gabi", "label": "pays", "eid": 1383539530258874368, "source_id": "roy", "properties": []}, {"source_label": "1", "target_label": "1", "target_id": "andy", "label": "pays", "eid": 1383539530258874369, "source_id": "roy", "properties": []}, {"source_label": "1", "target_label": "2", "target_id": "big_danny", "label": "owes", "eid": 1383580805582163713, "source_id": "roy", "properties": []}], "vertices": [{"properties": {"position": "old_man", "age": 90.0}, "id": "big_danny", "label": "2"}, {"properties": {"position": "data_scientist", "age": 27.0}, "id": "andy", "label": "1"}, {"properties": {"position": "boss", "age": 40.0}, "id": "gabi", "label": "2"}, {"properties": {"position": "engineer", "age": 26.0}, "id": "roy", "label": "1"}]}
```
Reference

Graph Databases

Ian Robinson,
Jim Webber & Emil Eifrem

© 2022 CY Lin, Columbia University
A usual example

![Diagram showing a relational database schema](image)

**Figure 2-1. Semantic relationships are hidden in a relational database**
Figure 2-2. Modeling friends and friends-of-friends in a relational database

Asking “who are Bob’s friends?” is easy, as shown in Example 2-1.

Example 2-1. Bob’s friends

```sql
SELECT p1.Person
FROM Person p1 JOIN PersonFriend
    ON PersonFriend.FriendID = p1.ID
JOIN Person p2
    ON PersonFriend.PersonID = p2.ID
WHERE p2.Person = 'Bob'
```
Example 2-2. Who is friends with Bob?

SELECT p1.Person
FROM Person p1 JOIN PersonFriend
    ON PersonFriend.PersonID = p1.ID
JOIN Person p2
    ON PersonFriend.FriendID = p2.ID
WHERE p2.Person = 'Bob'

Example 2-3. Alice’s friends-of-friends

SELECT p1.Person AS PERSON, p2.Person AS FRIEND_OF_FRIEND
FROM PersonFriend pf1 JOIN Person p1
    ON pf1.PersonID = p1.ID
JOIN PersonFriend pf2
    ON pf2.PersonID = pf1.FriendID
JOIN Person p2
    ON pf2.FriendID = p2.ID
WHERE p1.Person = 'Alice' AND pf2.FriendID <> p1.ID
Figure 2-5. Easily modeling friends, colleagues, workers, and (unrequited) lovers in a graph
Execution Time in the example of finding extended friends (by Neo4j)

Partner and Vukotic’s experiment seeks to find friends-of-friends in a social network, to a maximum depth of five. Given any two persons chosen at random, is there a path that connects them that is at most five relationships long? For a social network containing 1,000,000 people, each with approximately 50 friends, the results strongly suggest that graph databases are the best choice for connected data, as we see in Table 2-1.

Table 2-1. Finding extended friends in a relational database versus efficient finding in Neo4j

<table>
<thead>
<tr>
<th>Depth</th>
<th>RDBMS execution time (s)</th>
<th>Neo4j execution time (s)</th>
<th>Records returned</th>
</tr>
</thead>
<tbody>
<tr>
<td>2</td>
<td>0.016</td>
<td>0.01</td>
<td>~2500</td>
</tr>
<tr>
<td>3</td>
<td>30.267</td>
<td>0.168</td>
<td>~110,000</td>
</tr>
<tr>
<td>4</td>
<td>1543.505</td>
<td>1.359</td>
<td>~600,000</td>
</tr>
<tr>
<td>5</td>
<td>Unfinished</td>
<td>2.132</td>
<td>~800,000</td>
</tr>
</tbody>
</table>
Modeling Order History as a Graph

Figure 2-6. Modeling a user’s order history in a graph
Figure 3-1. A simple graph pattern, expressed using a diagram

This pattern describes three mutual friends. Here’s the equivalent ASCII art representation in Cypher:

\[(a)-[:KNOWS]->(b)-[:KNOWS]->(c), (a)-[:KNOWS]->(c)\]
Cypher Example

Like most query languages, Cypher is composed of clauses. The simplest queries consist of a START clause followed by a MATCH and a RETURN clause (we’ll describe the other clauses you can use in a Cypher query later in this chapter). Here’s an example of a Cypher query that uses these three clauses to find the mutual friends of user named Michael:

```cypher
START a=node:user(name='Michael')
MATCH (a)-[:KNOWS]->(b)-[:KNOWS]->(c), (a)-[:KNOWS]->(c)
RETURN b, c
```
Other Cypher Clauses

WHERE
Provides criteria for filtering pattern matching results.

CREATE and CREATE UNIQUE
Create nodes and relationships.

DELETE
Removes nodes, relationships, and properties.

SET
Sets property values.

FOREACH
Performs an updating action for each element in a list.

UNION
Merges results from two or more queries (introduced in Neo4j 2.0).

WITH
Chains subsequent query parts and forward results from one to the next. Similar to piping commands in Unix.
Property Graph Example – Shakespeare

Figure 3-6. Three domains in one graph
Creating the Shakespeare Graph

```sql
CREATE (shakespeare { firstname: 'William', lastname: 'Shakespeare' }),
  (juliusCaesar { title: 'Julius Caesar' }),
  (shakespeare)::WROTE_PLAY { year: 1599 }->(juliusCaesar),
  (theTempest { title: 'The Tempest' }),
  (shakespeare)::WROTE_PLAY { year: 1610 }->(theTempest),
  (rsc { name: 'RSC' }),
  (production1 { name: 'Julius Caesar' }),
  (rsc)::PRODUCED->(production1),
  (production1)::PRODUCTION_OF->(juliusCaesar),
  (performance1 { date: 20120729 }),
  (performance1)::PERFORMANCE_OF->(production1),
  (production2 { name: 'The Tempest' }),
  (rsc)::PRODUCED->(production2),
  (production2)::PRODUCTION_OF->(theTempest),
  (performance2 { date: 20061121 }),
  (performance2)::PERFORMANCE_OF->(production2),
  (performance3 { date: 20120730 }),
  (performance3)::PERFORMANCE_OF->(production1),
  (billy { name: 'Billy' }),
  (review { rating: 5, review: 'This was awesome!' }),
  (billy)::WROTE_REVIEW->(review),
  (review)::RATED->(performance1),
  (theatreRoyal { name: 'Theatre Royal' }),
  (performance1)::VENUE->(theatreRoyal),
  (performance2)::VENUE->(theatreRoyal),
  (performance3)::VENUE->(theatreRoyal),
  (greyStreet { name: 'Grey Street' }),
  (theatreRoyal)::STREET->(greyStreet),
  (newcastle { name: 'Newcastle' }),
  (greyStreet)::CITY->(newcastle),
  (tyneAndWear { name: 'Tyne and Wear' }),
  (newcastle)::COUNTY->(tyneAndWear),
  (england { name: 'England' }),
  (tyneAndWear)::COUNTRY->(england),
  (stratford { name: 'Stratford upon Avon' }),
  (stratford)::COUNTRY->(england),
  (rsc)::BASED_IN->(stratford),
  (shakespeare)::BORN_IN->stratford
```
START theater=node:venue(name='Theatre Royal'),
    newcastle=node:city(name='Newcastle'),
    bard=node:author(lastname='Shakespeare')
MATCH (newcastle)<-[[:STREET|CITY*1.2]]-(theater)
    <[-[:VENUE]-()]-[:PERFORMANCE_OF]->()[:PRODUCTION_OF]->
    (play)<-[[:WROTE_PLAY]]-(bard)
WHERE w.year > 1608
RETURN DISTINCT play.title AS play

Adding this WHERE clause means that for each successful match, the Cypher execution engine checks that the WROTE_PLAY relationship between the Shakespeare node and the matched play has a year property with a value greater than 1608. Matches with a WROTE_PLAY relationship whose year value is greater than 1608 will pass the test; these plays will then be included in the results. Matches that fail the test will not be included in the results. By adding this clause, we ensure that only plays from Shakespeare’s late period are returned:

```
+-----------------+
| play            |
+-----------------+
| "The Tempest"   |
+-----------------+
1 row
```
Another Query on the Shakespeare Graph

START theater=node:venue(name='Theatre Royal'),
      newcastle=node:city(name='Newcastle'),
      bard=node:author(lastname='Shakespeare')
MATCH (newcastle)-[:STREET|CITY*1..2]-(theater)
      <-[:VENUE]-(::PERFORMANCE_OF)-->()-[:PRODUCTION_OF]>
      (play)<-[:WROTE_PLAY]-(bard)
RETURN play.title AS play, count(p) AS performance_count
ORDER BY performance_count DESC

The RETURN clause here counts the number of PERFORMANCE_OF relationships using the identifier p (which is bound to the PERFORMANCE_OF relationships in the MATCH clause) and aliases the result as performance_count. It then orders the results based on performance_count, with the most frequently performed play listed first:

<table>
<thead>
<tr>
<th>play</th>
<th>performance_count</th>
</tr>
</thead>
<tbody>
<tr>
<td>&quot;Julius Caesar&quot;</td>
<td>2</td>
</tr>
<tr>
<td>&quot;The Tempest&quot;</td>
<td>1</td>
</tr>
</tbody>
</table>

2 rows
Building Application Example – Collaborative Filtering

Figure 4-1. Data model for the book reviews user story

Because this data model directly encodes the question presented by the user story, it lends itself to being queried in a way that similarly reflects the structure of the question we want to ask of the data:

```
START reader=node:users(name={readerName})
MATCH reader-[:LIKES]->book<-[:LIKES]-other_readers-[:LIKES]->books
RETURN books.title
```
Chaining on the Query

START bard=node:author(lastname='Shakespeare')
MATCH (bard)-[w:WROTE_PLAY]->(play)
WITH play
ORDER BY w.year DESC
RETURN collect(play.title) AS plays

Executing this query against our sample graph produces the following result:

<table>
<thead>
<tr>
<th>plays</th>
</tr>
</thead>
<tbody>
<tr>
<td>&quot;The Tempest&quot;, &quot;Julius Caesar&quot;</td>
</tr>
</tbody>
</table>
1 row
What's this query for?

START bob=node:user(username='Bob')
MATCH (bob)-[:SENT]->(email)-[:CC]->(alias),
(alias)-[:ALIAS_OF]->(bob)
RETURN email
How to make graph database fast?

*Figure 6-1. Nonnative graph processing engines use indexing to traverse between nodes*
Use Relationships, not indexes, for fast traversal
Ardi Autonomous Model Optimizing System (AMOS)
Machine Learning Module

**Model Training** – Provides convenient functions such as features importing and preprocessing to model developer, user can choose machine learning model and algorithm and tune parameters flexibly.

**Model Deployment** – Support user to set the frequency of model execution, model deployment and execution.

**Model Optimization** – Support users to optimize model flexibly.

**Model Evaluation** – Support general evaluation criteria to regression and classification like accuracy and recall.

**Model Management** – Integrated supports of importing various features data, choosing model type, saving model, setting access right and deployment. Support importation of models trained on outer platforms. Support automatically generate version of models.
Algorithms Support

Classification
- Support Vector Machine
- XGBoost
- LightGBM
- Random Forest
- Decision Tree

Regression
- Ordinary Linear Regression
- Ridge Regression
- Logistic Regression

- Clustering
  - K-means
  - Birch

- Deep Learning (via TensorFlow)
  - Insert/delete layers
  - Recurrent Neural Network (RNN)
  - Deep neural network (DNN)
  - Convolutional neural network (CNN)
Modeling

• **Model**
  - Random seed
  - Choose parameters automatically
  - Supporting method of parameters choosing: grid search, Bayesian optimization, random search, gradient boosted tree optimization, sequential optimization
  - Model training
    - Support manually tune parameters
  - Show the report of model training timeline and parameter tuning

• **Model Evaluation Criteria**
  - Accuracy
  - Recall
  - precision
  - log_loss
  - jaccard similarity
  - confusion matrix

• **Prediction**
  - Predict the data without label
  - Save as CSV
  - Generate a final report

• **Validation Method**
  - Hold out
  - Cross validation
  - Time series split
Ardi Autonomous Learning System

- Environment Addition
- Data Description
- Model Generation
- Structure Optimization
- Analytics
- Model Update
- Management
Bayesian Network Exploration
AI Market Intelligence Analysis
Chain together probabilities
Conditional probabilities represents parent-child relationship

\[ P(\text{child} \mid \text{parent}_1, \text{parent}_2, \ldots) \]

\[ P(\text{Wet Ground} \mid \text{Car Wash, Rain}), P(\text{Slip} \mid \text{Wet Ground}) \]

\[ P(R, W, S, C) = P(R) \ P(C) \ P(W \mid C, R) \ P(S \mid W) \]

https://www.youtube.com/watch?v=TuGDMj43ehw&feature=youtu.be
Example — diagnosis

e.g. person is smoker and has tuberculosis (click on checkmark to choose true or false) increases chance of seeing anomalies in xray result and higher chance of having dyspnea compared to the baseline

https://www.bayesserver.com/examples/networks/asia
Creating a Bayesian Network

- We want to generate the network, but we may have only sets of observations for each condition or nodes.
- But it’s a NP-Hard problem since we have to go through all different permutations of nodes and directed edges. The time complexity grows the more nodes you have. \(O(2^n(n-1))\)
- We could develop certain heuristics to reduce this search space
- We use K2 scoring as a metric to see how well the sample data fits well with a generated network. We iterate through several permutations of generated networks and pick the one with the best score.