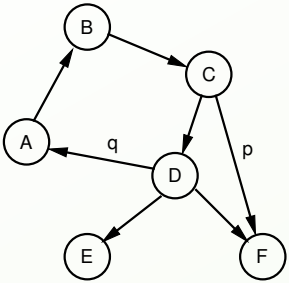
ArangoDB

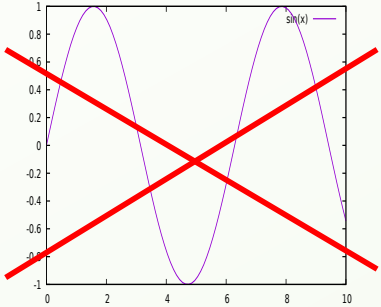
Fishing Graphs in a Hadoop Data Lake

Düsseldorf, 21 February 2018

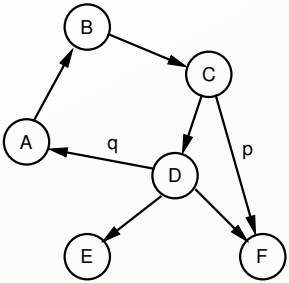
Max Neunhöffer

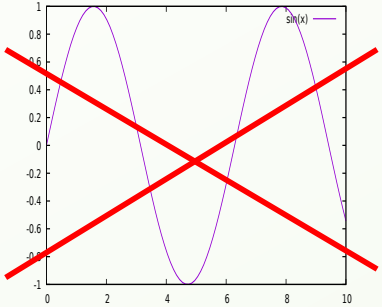
What is a graph?





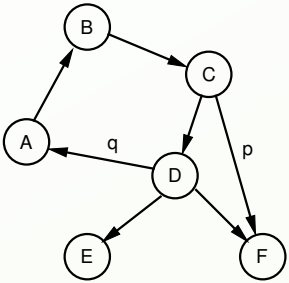
What is a graph?

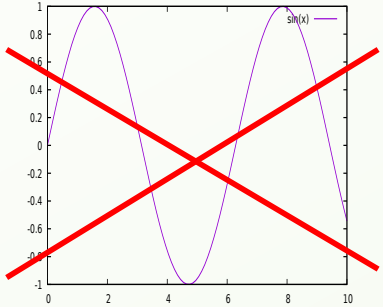




- Social networks (edges are friendship)
- Dependency chains
- Computer networks
- Citations
- Hierarchies

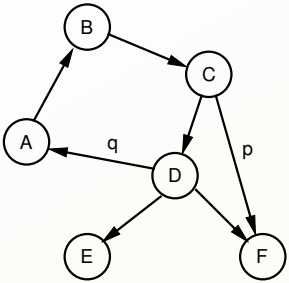
What is a graph?

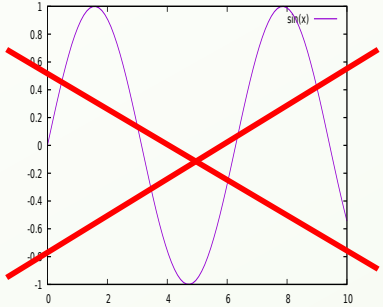




- Social networks (edges are friendship)
- Dependency chains
- Computer networks
- Citations
- Hierarchies
- Indeed any relation

What is a graph?





- Social networks (edges are friendship)
- Dependency chains
- Computer networks
- Citations
- Hierarchies
- Indeed any relation

Sometimes **directed**, sometimes **undirected**.

Usual approach: data in HDFS, use Spark/GraphFrames

```
v = spark.read.option("header",true).csv("hdfs://...")
e = spark.read.option("header",true).csv("hdfs://...")
g = GraphFrame(v,e)

g.inDegrees.show()

g.outDegrees.groupBy("outDegree").count().sort("outDegree").show(1000)

g.vertices.groupBy("GYEAR").count().sort("GYEAR").show()

g.find("(a)-[e]->(b);(b)-[ee]->(c)").filter("a.id = 6009536").count()

results = g.pageRank(resetProbability=0.01, maxIter=3)
```

Limitations/missed opportunities

Ad hoc queries

Often, one would like to perform smallish ad hoc queries on graph data.

Limitations/missed opportunities

Ad hoc queries

Often, one would like to perform smallish ad hoc queries on graph data. Want to bring down latency **from minutes to seconds** or **from seconds to milliseconds**.

Limitations/missed opportunities

Ad hoc queries

Often, one would like to perform smallish ad hoc queries on graph data. Want to bring down latency **from minutes to seconds** or **from seconds to milliseconds**. Usually, we would like to run many of them.

Limitations/missed opportunities

Ad hoc queries

Often, one would like to perform smallish ad hoc queries on graph data. Want to bring down latency **from minutes to seconds** or **from seconds to milliseconds**. Usually, we would like to run many of them.

Examples:

- friends of friends **of one person**
- find all immediate dependencies **of one item**
- find all direct and indirect citations **of one article**
- find all descendants **of one member** of a hierarchy

Limitations/missed opportunities

Ad hoc queries

Often, one would like to perform smallish ad hoc queries on graph data. Want to bring down latency **from minutes to seconds** or **from seconds to milliseconds**. Usually, we would like to run many of them.

Examples:

- friends of friends **of one person**
- find all immediate dependencies **of one item**
- find all direct and indirect citations **of one article**
- find all descendants **of one member** of a hierarchy

IDEA: Use a **Graph Database**

Graph Databases

Graph Databases

Can store and persist graphs.

Graph Databases

Graph Databases

Can store and persist graphs. However, the crucial ingredient of a graph database is their ability to do **graph queries**.

Graph Databases

Graph Databases

Can store and persist graphs. However, the crucial ingredient of a graph database is their ability to do **graph queries**.

Graph queries:

- Find paths in graphs according to **a pattern**.
- Find everything reachable from a vertex.
- Find shortest paths between two given vertices.

Graph Databases

Graph Databases

Can store and persist graphs. However, the crucial ingredient of a graph database is their ability to do **graph queries**.

Graph queries:

- Find paths in graphs according to **a pattern**.
- Find everything reachable from a vertex.
- Find shortest paths between two given vertices.

$\Longrightarrow$ **Graph Traversals**

Graph Databases

Graph Databases

Can store and persist graphs. However, the crucial ingredient of a graph database is their ability to do **graph queries**.

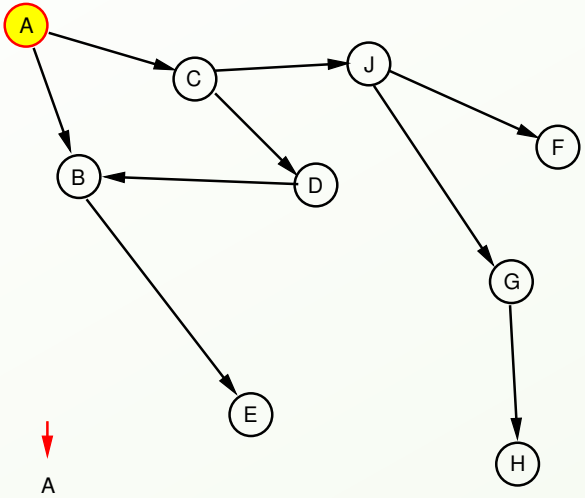
Graph queries:

- Find paths in graphs according to **a pattern**.
- Find everything reachable from a vertex.
- Find shortest paths between two given vertices.

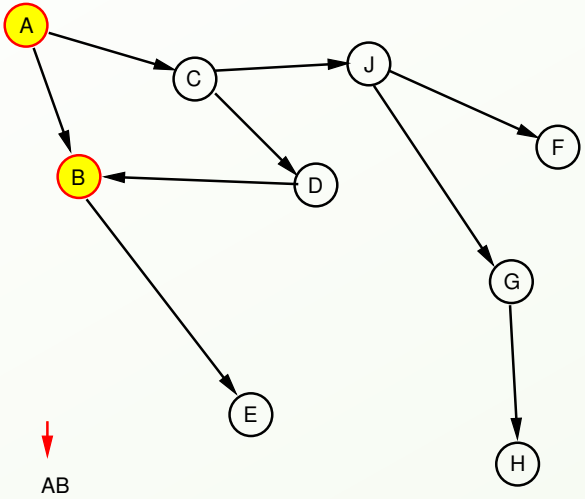
$\Longrightarrow$ **Graph Traversals**

Crucial: Number of steps a priori unknown!

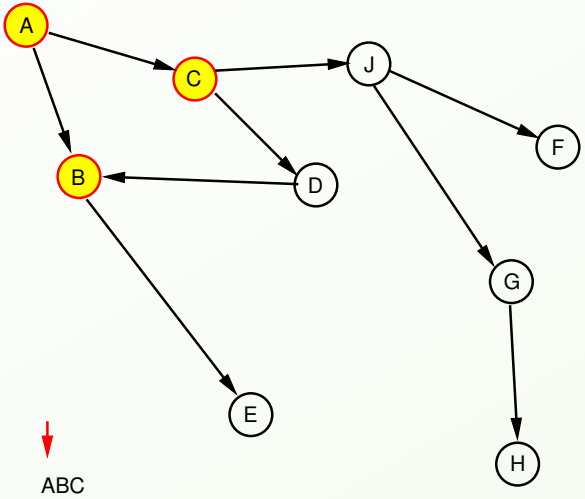
Graph Traversals



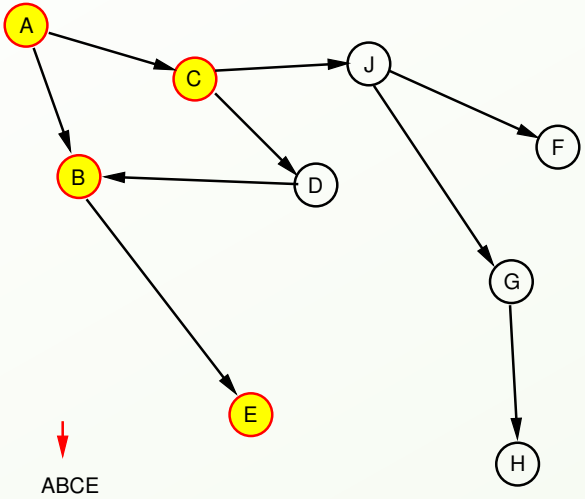
Graph Traversals



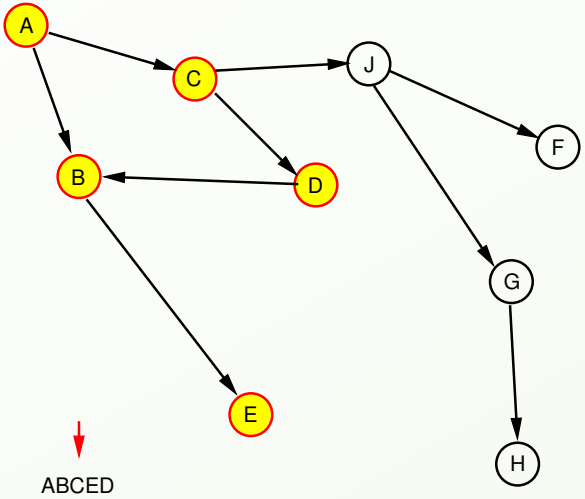
Graph Traversals



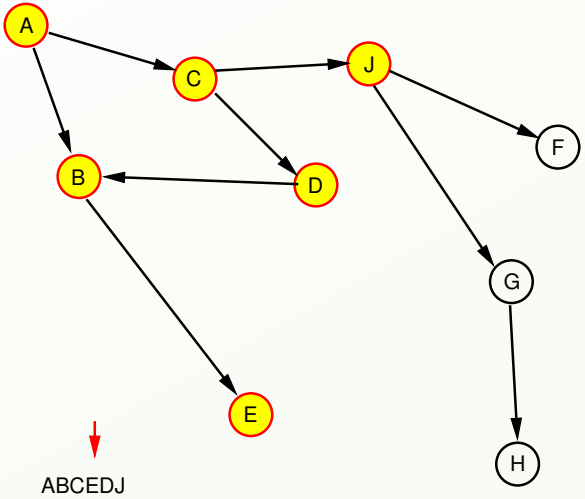
Graph Traversals



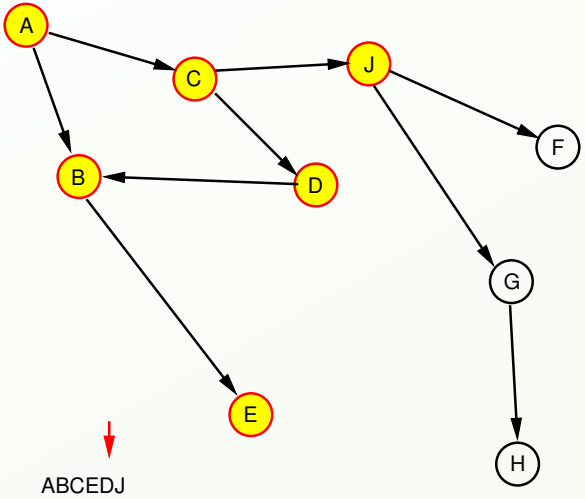
Graph Traversals



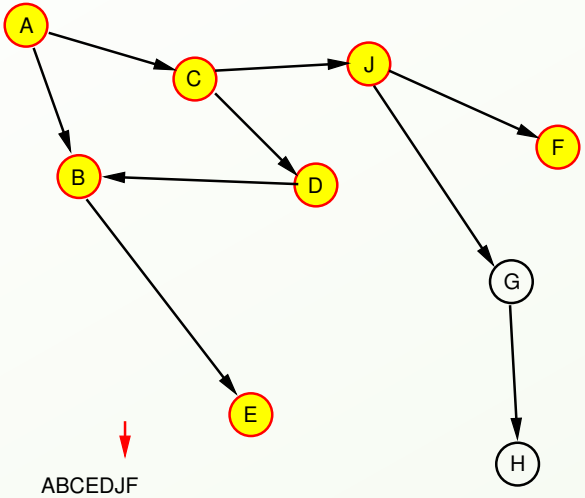
Graph Traversals



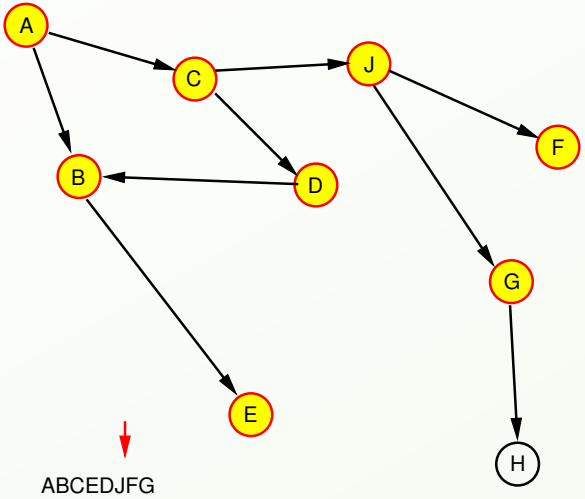
Graph Traversals



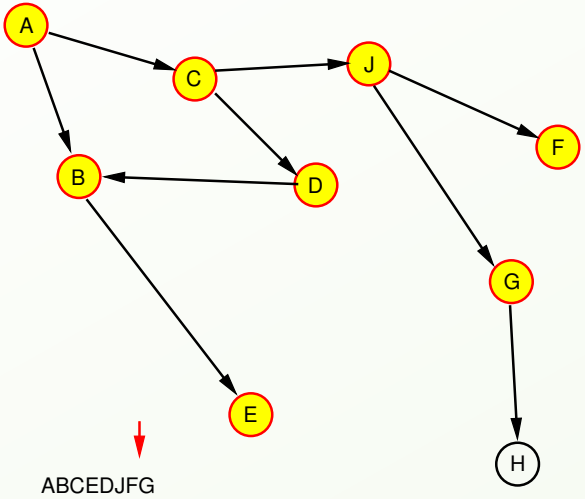
Graph Traversals



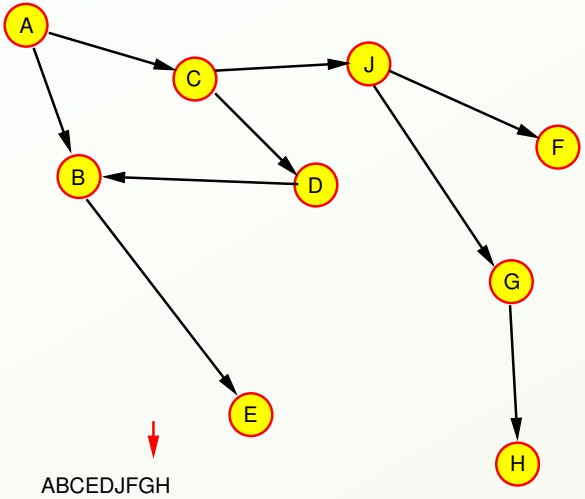
Graph Traversals



Graph Traversals



Graph Traversals



The Multi-Model Approach

Multi-model database

A multi-model database combines a document store with a graph database and is at the same time a key/value store,

The Multi-Model Approach

Multi-model database

A multi-model database combines a document store with a graph database and is at the same time a key/value store,
with a common query language for all three data models.

The Multi-Model Approach

Multi-model database

A multi-model database combines a document store with a graph database and is at the same time a key/value store,
with a common query language for all three data models.

Important:

- Is able to compete with specialised products on their turf.

The Multi-Model Approach

Multi-model database

A multi-model database combines a document store with a graph database and is at the same time a key/value store, with a common query language for all three data models.

Important:

- Is able to compete with specialised products on their turf.
- Allows for **polyglot persistence** using a single database technology.

The Multi-Model Approach

Multi-model database

A multi-model database combines a document store with a graph database and is at the same time a key/value store,
with a common query language for all three data models.

Important:

- Is able to compete with specialised products on their turf.
- Allows for **polyglot persistence** using a single database technology.
- In a microservice architecture, there will be several **different** deployments.

Powerful query language

AQL

The built in Arango Query Language allows

- complex, powerful and convenient queries,

Powerful query language

AQL

The built in Arango Query Language allows

- complex, powerful and convenient queries,
- with transaction semantics,

Powerful query language

AQL

The built in Arango Query Language allows

- complex, powerful and convenient queries,
- with transaction semantics,
- allowing to do joins,

Powerful query language

AQL

The built in Arango Query Language allows

- complex, powerful and convenient queries,
- with transaction semantics,
- allowing to do joins,
- and to do graph queries,

Powerful query language

AQL

The built in Arango Query Language allows

- complex, powerful and convenient queries,
- with transaction semantics,
- allowing to do joins,
- and to do graph queries,
- AQL is independent of the driver used and

Powerful query language

AQL

The built in Arango Query Language allows

- complex, powerful and convenient queries,
- with transaction semantics,
- allowing to do joins,
- and to do graph queries,
- AQL is independent of the driver used and
- offers protection against injections by design.

ArangoDB is a Data Center Operating System App

These days, computing clusters run Data Center Operating Systems.

ArangoDB is a Data Center Operating System App

These days, computing clusters run Data Center Operating Systems.

Idea

Distributed applications can be deployed as easily as one installs a mobile app on a phone.

ArangoDB is a Data Center Operating System App

These days, computing clusters run Data Center Operating Systems.

Idea

Distributed applications can be deployed as easily as one installs a mobile app on a phone.

- Cluster resource management is **automatic**.

ArangoDB is a Data Center Operating System App

These days, computing clusters run Data Center Operating Systems.

Idea

Distributed applications can be deployed as easily as one installs a mobile app on a phone.

- Cluster resource management is **automatic**.
- This leads to significantly better **resource utilization**.

ArangoDB is a Data Center Operating System App

These days, computing clusters run Data Center Operating Systems.

Idea

Distributed applications can be deployed as easily as one installs a mobile app on a phone.

- Cluster resource management is **automatic**.
- This leads to significantly better **resource utilization**.
- Fault tolerance, self-healing and automatic failover is **guaranteed**.

ArangoDB is a Data Center Operating System App

These days, computing clusters run Data Center Operating Systems.

Idea

Distributed applications can be deployed as easily as one installs a mobile app on a phone.

- Cluster resource management is **automatic**.
- This leads to significantly better **resource utilization**.
- Fault tolerance, self-healing and automatic failover is **guaranteed**.

ArangoDB runs on Apache Mesos and kubernetes clusters.

Back to topic: Cloud orchestration as infrastructure

Cloud orchestration is the perfect environment for our needs

It manages for us:

- Software deployment
- Resource management (increased utilization)
- Service discovery

Back to topic: Cloud orchestration as infrastructure

Cloud orchestration is the perfect environment for our needs

It manages for us:

- Software deployment
- Resource management (increased utilization)
- Service discovery
- Allows to plug things together!

Back to topic: Cloud orchestration as infrastructure

Cloud orchestration is the perfect environment for our needs

It manages for us:

- Software deployment
- Resource management (increased utilization)
- Service discovery
- Allows to plug things together!

Consequence: We can easily deploy multiple systems alongside each other.

Back to topic: Cloud orchestration as infrastructure

Cloud orchestration is the perfect environment for our needs

It manages for us:

- Software deployment
- Resource management (increased utilization)
- Service discovery
- Allows to plug things together!

Consequence: We can easily deploy multiple systems alongside each other.

Example: HDFS, Spark and ArangoDB

Deployment on kubernetes (work in progress)

Deploy the ArangoDB operator to k8s

```
kubectl create -f arangodb-operator.yaml
```

Deployment on kubernetes (work in progress)

Deploy the ArangoDB operator to k8s

```
kubectl create -f arangodb-operator.yaml
```

Deploy an ArangoDB cluster instance

```
kubectl create -f simple-cluster.yaml
```

simple-cluster.yaml

```
apiVersion: "database.arangodb.com/v1alpha"
kind: "ArangoDeployment"
metadata:
  name: "example-arangodb-cluster"
spec:
  mode: cluster
```

Import data into ArangoDB

```
hdfs dfs -get hdfs://name-1-node.hdfs.mesos:9001/patents.csv
hdfs dfs -get hdfs://name-1-node.hdfs.mesos:9001/citations.csv

dcos package install arangodb3

arangosh \
    --server.endpoint srv://_arangodb3-coordinator1._tcp.arangodb3.mesos

var g = require("@arangodb/general-graph");
var G = g._create("G",[g._relation("citations",["patents"],["patents"])]);

arangoimp --collection patents --file patents.csv --type csv \
    --server.endpoint srv://_arangodb3-coordinator1._tcp.arangodb3.mesos
arangoimp --collection citations --file citations.csv --type csv \
    --server.endpoint srv://_arangodb3-coordinator1._tcp.arangodb3.mesos
```

Run a graph traversal

This query finds patents cited by `patents/6009503` (depth $\leq$ 3) recursively:

Recursive traversal, 500 results, 317 ms

```
FOR v IN 1..3 OUTBOUND "patents/6009503" GRAPH "G"
  RETURN v
```

Run a graph traversal

This query finds patents cited by `patents/6009503` (depth $\leq$ 3) recursively:

Recursive traversal, 500 results, 317 ms

```
FOR v IN 1..3 OUTBOUND "patents/6009503" GRAPH "G"
  RETURN v
```

This one finds all patents that cite any of those cited by `patents/6009503`:

One step forward and one back, 35 results, 59 ms

```
FOR v IN 1..1 OUTBOUND "patents/6009503" GRAPH "G"
  FOR w IN 1..1 INBOUND v._id GRAPH "G"
    FILTER w._id != v._id
    RETURN w
```

Run a graph traversal

This query finds all patents that cite `patents/3541687` directly or in two steps:

Recursive traversal backwards, 22 results, 15 ms

```
FOR v IN 1..2 INBOUND "patents/3541687" GRAPH "G"
  RETURN v._key
```

Run a graph traversal

This query finds all patents that cite `patents/3541687` directly or in two steps:

Recursive traversal backwards, 22 results, 15 ms

```
FOR v IN 1..2 INBOUND "patents/3541687" GRAPH "G"
  RETURN v._key
```

This one counts all patents that cite `patents/3541687` recursively:

Deep recursion backwards, count 398, 311 ms

```
FOR v IN 1..10 INBOUND "patents/3541687" GRAPH "G"
  COLLECT WITH COUNT INTO c
  RETURN c
```

Yet another approach

If your graph data changes rapidly in a transactional fashion...

Yet another approach

If your graph data changes rapidly in a transactional fashion...

Graph database as primary data store

You can **turn things around**:

- Keep and maintain the graph data in a **graph database**.

Yet another approach

If your graph data changes rapidly in a transactional fashion...

Graph database as primary data store

You can **turn things around**:

- Keep and maintain the graph data in a **graph database**.
- Regularly dump to HDFS and **run larger analysis jobs** there.

Yet another approach

If your graph data changes rapidly in a transactional fashion...

Graph database as primary data store

You can **turn things around**:

- Keep and maintain the graph data in a **graph database**.
- Regularly dump to HDFS and **run larger analysis jobs** there.

Or: Use ArangoDB's **Spark Connector**:

`https://github.com/arangodb/arangodb-spark-connector`

Links

Slides will be at: `https://www.arangodb.com/speakers/max-neunhoeffer/`

`http://hadoop.apache.org/`

`http://spark.apache.org/`

`https://graphframes.github.io/`

`https://www.arangodb.com`

`https://github.com/arangodb/arangodb-spark-connector`

Github: `https://github.com/arangodb/ArangoDB` **(please star us!)**

Twitter: `@arangodb` **(please follow us!)**