Building Graph Applications with Neo4j

Neo4j Manchester Meetup

Wednesday 1st March
Contents

- Introduction
- The Graph
  - Introduction to the demo
  - Neo4j Primer
  - The graph database
  - Recommendations
- Building a front end around Neo4j
  - Motivations
  - Data management
  - Live demo
Metafused

- Use AI to optimise existing and build net new applications for a broad set of verticals
  - AI is applied to reduce friction, optimise processes, execute action(s) based on observed trigger(s)
- Data driven
  - Being comfortable with being uncomfortable
  - Using the right tool(s) for the job
  - Build, measure, learn
Our Team
Seven Bridges of Königsberg

‘Devise a walk through the city that would cross each of those bridges once and only once.’

Leonhard Euler (1736) proved that, with some problems, you can’t solve them doing the same thing you did yesterday ... and expecting different results.
Thinking Differently

Making the Case for the Bottom Up Approach

‘You can’t do much carpentry with bare hands. Neither can you do much thinking with a bare brain.’ –Daniel Dennett
Or, if you prefer not to have to read . . .

But you will have to watch Brad Pitt. Sorry.
Backend Design
Building Applications with Neo4j

- Presentation of a prototype application based on a well-known problem
  - Demonstrate capability of technology stack
  - Show how Metafused are using Neo4j as a key component of our architecture
  - Demonstrate how Neo4j can be used as part of an AI system
- Building a live recommendation system
**Neo4j Primer**

- **Nodes**
  - Objects representing entities
- **Labels**
  - Assigned to nodes to specify the type of entity
- **Relationships**
  - Directional connections between nodes
- **Properties**
  - Additional information which can be attached to nodes and relationships
- **Cypher**
  - Declarative query language for Neo4j
- **Patterns**
  - Selected combinations of nodes and relationships
- Let's explore these concepts along with the graph...
A Known Graph

To demonstrate the technology, start simple. A great use case for Neo4j is the Movie graph seen in many of the training examples.
A Simple Query (1)

Start with a simple Cypher query, find a node, with label Person and name property equal to “Andrew Stanton”.

MATCH (p :Person) WHERE p.name = "Andrew Stanton" RETURN p;

MATCH (p :Person) WHERE p.name = "Andrew Stanton" RETURN p

Returned 1 record in 74 ms.
A Simple Query (2)

We can look at specific relationships to answer simple questions, for example what movies have Andrew Stanton had a relationship with, or more specifically directed?

MATCH (p :Person)-[r]->(m :Movie) WHERE p.name = "Andrew Stanton" RETURN p, r, m;
MATCH (p :Person)-[d :DIRECTED]->(m :Movie) WHERE p.name = "Andrew Stanton" RETURN p.name, collect(m.title);
The Data

- Data gathered from IMDb using IMDbPY
  - IMDb not for commercial use, OK for this presentation
- Use IMDbPY to query database
- Either
  - Write script to build csv files for nodes and relationships
  - Directly ingest with py2neo ([http://py2neo.org/v3/](http://py2neo.org/v3/))
- This demo
  - 325 movies + full cast, directors, producers, writers
  - Easily extend with any further information e.g. full crew, trivia, keywords etc
  - Initially 10 example users with ratings assigned to 25% of movies following a defined distribution
MATCH (u :User)-[r :RATED]->(m :Movie)
RETURN u.name AS Name,
    u.username AS Username,
    count(r) AS Reviews,
    avg(r.rating) AS 'Average Score'
ORDER BY Reviews DESC
LIMIT 5;

<table>
<thead>
<tr>
<th>Name</th>
<th>Username</th>
<th>Reviews</th>
<th>Average Score</th>
</tr>
</thead>
<tbody>
<tr>
<td>Malcolm Reynolds</td>
<td>TheCaptain</td>
<td>101</td>
<td>3.183168316831683</td>
</tr>
<tr>
<td>Kaylee Frye</td>
<td>Shiny2000</td>
<td>95</td>
<td>3.3578947368421055</td>
</tr>
<tr>
<td>Jayne Cobb</td>
<td>HeroOfCanton11</td>
<td>94</td>
<td>3.255319148936172</td>
</tr>
<tr>
<td>Hoban Washburne</td>
<td>Wash</td>
<td>89</td>
<td>3.202247191011237</td>
</tr>
<tr>
<td>Zoe Washburne</td>
<td>BrownCoatGal</td>
<td>89</td>
<td>3.140449438202248</td>
</tr>
</tbody>
</table>
We can compare two users reviews

MATCH (u1 :User)-[r1 :RATED]->(m :Movie)<-[r2 :RATED]-(u2 :User)
WHERE u1.name = "Malcolm Reynolds" AND u2.name = "Jayne Cobb"
WITH m,
    r1.rating AS score_mal,
    r2.rating AS score_jayne
MATCH (:User)-[r :RATED]->(m)
RETURN m.title AS Movie,
    score_mal AS 'Mal's Rating',
    score_jayne AS 'Jayne's Rating',
    count(r.rating) AS 'Total Reviews',
    avg(r.rating) AS 'Average Rating'
ORDER BY (score_mal + score_jayne)/2 DESC
LIMIT 3;

<table>
<thead>
<tr>
<th>Movie</th>
<th>Mal's Rating</th>
<th>Jayne's Rating</th>
<th>Total Reviews</th>
<th>Average Rating</th>
</tr>
</thead>
<tbody>
<tr>
<td>Terminator 2: Judgment Day</td>
<td>5</td>
<td>3.5</td>
<td>7</td>
<td>3.357142857142857</td>
</tr>
<tr>
<td>The Matrix</td>
<td>4</td>
<td>4.5</td>
<td>2</td>
<td>4.25</td>
</tr>
<tr>
<td>Step Brothers</td>
<td>4</td>
<td>4.5</td>
<td>4</td>
<td>4</td>
</tr>
</tbody>
</table>
**Similarity**

We can measure how similar users are based on their reviews. To do this we will be using the Euclidean distance. The smaller the value the more similar the user.

\[
distance(u_1, u_2) = \sqrt{\sum_{\text{review}=1}^{n} (u_{1_{\text{review}}} - u_{2_{\text{review}}})^2}
\]
**Similarity - Example**

\[
distance(u_1, u_2) = \sqrt{\sum_{\text{review}=1}^{n} (u_{1\text{review}} - u_{2\text{review}})^2}
\]

<table>
<thead>
<tr>
<th>Rating 1</th>
<th>Rating 2</th>
<th>Diff</th>
<th>Diff Squared</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>5</td>
<td>-4</td>
<td>16</td>
</tr>
<tr>
<td>1</td>
<td>4.5</td>
<td>-3.5</td>
<td>12.25</td>
</tr>
<tr>
<td>1</td>
<td>2.5</td>
<td>-1.5</td>
<td>2.25</td>
</tr>
<tr>
<td>1</td>
<td>4</td>
<td>-3</td>
<td>9</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td><strong>Total</strong></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td><strong>39.5</strong></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td><strong>Distance</strong></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td><strong>6.28</strong></td>
</tr>
</tbody>
</table>
**Similarity - Example**

\[
distance(u_1, u_2) = \sqrt{\sum_{\text{review}=1}^{n} \left( u_{1\text{review}} - u_{2\text{review}} \right)^2}
\]

<table>
<thead>
<tr>
<th>Rating 1</th>
<th>Rating 2</th>
<th>Diff</th>
<th>Diff Squared</th>
</tr>
</thead>
<tbody>
<tr>
<td>5</td>
<td>5</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>4.5</td>
<td>4.5</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>2.5</td>
<td>2.5</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>4</td>
<td>4</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td><strong>Total</strong></td>
<td></td>
<td></td>
<td><strong>0</strong></td>
</tr>
<tr>
<td><strong>Distance</strong></td>
<td>0.00</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
**Nearest Neighbours**

We can calculate similarities in Neo4j and then find a user’s nearest neighbours.

```
// Update similarities between users
MATCH (u1:User)-[x:RATED]->(m:Movie)<-[y:RATED]-(u2:User)
WHERE u1 <> u2
WITH SQRT(REDUCE(acc = 0.0, dif IN COLLECT(x.rating - y.rating) | acc + dif^2))/count(m) AS sim, u1, u2, m
MERGE (u1)-[s:SIMILARITY]-(u2)
SET s.similarity = sim;

// Get nearest neighbors (lower the better)
MATCH (u1 :User)-[s :SIMILARITY]-(u2 :User)
WHERE u1.name = "Malcolm Reynolds"
WITH u2.name AS Neighbor,
    s.similarity AS sim
ORDER BY sim
RETURN Neighbor,
    sim AS Similarity
LIMIT 10;
```
Nearest Neighbours

We can calculate similarities in Neo4j and then find a user’s nearest neighbours.

<table>
<thead>
<tr>
<th>Neighbor</th>
<th>Similarity</th>
</tr>
</thead>
<tbody>
<tr>
<td>Simon Tam</td>
<td>0.22480460265528374</td>
</tr>
<tr>
<td>Jayne Cobb</td>
<td>0.2894251362084897</td>
</tr>
<tr>
<td>Derrial Buck</td>
<td>0.29084974261434027</td>
</tr>
<tr>
<td>River Tam</td>
<td>0.2993402609685752</td>
</tr>
<tr>
<td>Kaylee Frye</td>
<td>0.33364137640187447</td>
</tr>
<tr>
<td>Inara Serra</td>
<td>0.344853028275976</td>
</tr>
<tr>
<td>Hoban Washburne</td>
<td>0.3582295386576376</td>
</tr>
<tr>
<td>Zoe Washburne</td>
<td>0.4272001872658765</td>
</tr>
</tbody>
</table>
Recommendations

We can now design our recommendation engine.

1. User logs in and registers in the database
2. User rates a new movie
3. System
   a. Updates database information
   b. Finds user’s new nearest neighbors
   c. Calculate average reviews of movies by x most similar users
   d. Exclude movies already rated by the user
   e. Order movies by average rating
   f. Deliver recommendations
4. User filters results, for example by genre

Let's see how that looks...
Recommendation Query

We can now design our recommendation engine.

// Match user a to users who have rated 2 or more of the same films
MATCH (u2 :User)-[:RATED]->(film: Movie)<-[[:RATED]]-(u1 :User {id: "usr0"})
WITH u1,
    u2,
    COUNT(film) AS film_count
// Reviewed two or more of the same films
WHERE film_count > 1
WITH u1,
    u2
// MATCH Users similarities
MATCH (u2)-[s:SIMILARITY]-(u1)
WITH u1, u2, s.similarity AS similarity
ORDER BY similarity
LIMIT 3
WITH u1, u2, similarity
// Get movies rated by the similar users
MATCH (u2)-[r:RATED]->(m :Movie)
// Only movies user 1 hasn't seen
WHERE NOT((u1)-[:RATED]->(m))
WITH m,
    similarity,
    r.rating AS rating
// Group movies
ORDER BY m.title
WITH m,
    REDUCE(s = 0, i IN COLLECT(rating) | s + i) * 1.0 AS rating_sum,
    SIZE(COLLECT(rating)) AS n_ratings
// Movie must have at least 2 ratings
WHERE n_ratings > 1
// Get the average review
WITH m,
    rating_sum/n_ratings AS reco,
    n_ratings
// Get the genres
MATCH (m)-[:HAS_GENRE]->(g :Genre)
WITH m,
    reco,
    COLLECT(g.name) AS genres,
    n_ratings
// Order by the average recommendation score (not average rating)
// and then n_ratings
ORDER BY reco DESC,
    n_ratings DESC
// Return list
RETURN m.title,
    reco AS score,
    genres
LIMIT 10
## Recommendation Query

<table>
<thead>
<tr>
<th>m.title</th>
<th>score</th>
<th>genres</th>
</tr>
</thead>
<tbody>
<tr>
<td>Back to the Future</td>
<td>4.5</td>
<td>[Comedy, Sci-Fi, Adventure]</td>
</tr>
<tr>
<td>Labyrinth</td>
<td>4.5</td>
<td>[Action, Fantasy, Mystery, Sport]</td>
</tr>
<tr>
<td>Inside Out</td>
<td>4.25</td>
<td>[Family, Drama, Fantasy, Animation, Comedy, Adventure]</td>
</tr>
<tr>
<td>The Jungle Book</td>
<td>4.25</td>
<td>[Adventure, Family, Drama, Fantasy]</td>
</tr>
<tr>
<td>The Dark Knight Rises</td>
<td>3.75</td>
<td>[Thriller, Action]</td>
</tr>
<tr>
<td>Goodfellas</td>
<td>3.75</td>
<td>[Drama, Crime, Biography]</td>
</tr>
<tr>
<td>Intouchables</td>
<td>3.5</td>
<td>[Drama, Comedy, Biography]</td>
</tr>
<tr>
<td>Sin City</td>
<td>3.25</td>
<td>[Crime, Thriller]</td>
</tr>
<tr>
<td>We’re the Millers</td>
<td>3.25</td>
<td>[Thriller, Musical]</td>
</tr>
<tr>
<td>Aliens</td>
<td>3.25</td>
<td>[Adventure, Action, Thriller, Sci-Fi]</td>
</tr>
</tbody>
</table>

Returned 10 records in 30 ms.
Extending the Recommender

The current setup demonstrates the power of a graph database for recommendation, this could be extended in several ways:

- More complex queries incorporating further information e.g. favourite actors
  - Natural use of Neo4j
  - Easily scales
  - Simple to understand
  - Real time

- Scale with more complex Machine Learning
  - Expand on simple nearest neighbour example
  - Include additional contextual information
Google Cloud Platform

We are using Google Cloud Platform (GCP) to help us build and deploy services/applications.

- Google are working towards a ‘no-ops’ environment
- Incorporate virtual machines, clusters, software and APIs
- Allows for rapid prototyping and focus on areas of expertise
Incorporating Machine Learning into the application is easy with GCP.
MOVING TO AN APPLICATION

We have built a simple recommendation engine around data stored in Neo4j.

- How is this delivered to the user?
- How does the front end communicate with Neo4j?
Front End
Introduction

- Working using a build, measure, learn methodology allows us to work through prototypes quickly
- Front end choices motivated by ability to reuse components
- Walk through some design choices for Movie Recommendation Engine
- Discuss future changes and what we have learned
Movie Recommendation Engine Technology Stack

ne04j → Axios → Redux → React → User
**Interacting with Neo4j: Key Points**

- Interacting with Neo4j through transaction end points
  - Move to using node server connected via the Bolt driver
- Able to send multiple queries with one request
- Query design, balance query speed/complexity with number of requests and amount of front end processing
  - For example - one query to return movies with associated genre rather than one query per genre
State Management (1)

- Redux provides a system for state management
  - Immutable
- Data objects stored in a state tree which is updated via Redux’s actions and reducers
- Demonstrate this for part of our application over the next few slides
  - Focus on a small part of state tree
State Management (2)

- Application initialise:
  - Several data objects
  - Focus on movies for this example
State Management (3)

- Application initialises
- Fetch movie data from Neo4j
State Management (4)

- Application initialises
- Fetch movie data from Neo4j
- Movie data processed, broken down into genres
  - Data now available in state
State Management (5)

- Application initialises
- Fetch movie data from Neo4j
- Move data processed, broken down into genres
  - Data available in state
- User selects movie
State Management (5)

- Further actions update the state
  - Movie review
  - Select another movie
  - Log out
  - Log in
Joining the back and front end together helps us build out our application.
Extending the User Experience

- Currently have a very basic application
- Neo4j + GCP give us the ability to quickly iterate through new ideas using a build, measure, learn methodology
  - Add new labels, nodes, relationships and properties to the graph
  - Ingest new data and add more ML
- User could add review comments
  - Sentiment analysis
  - Improve recommendation
- Users could add each other as friends
  - Improve recommendations
  - Push alerts
- Conversational UI/chatbot
- Live voting system
Live Demo
Thanks for listening any questions?