The Million Song Graph

Andrea Soto

Project Overview and Goals

Problem	How to store a music recommendation system
Solution	Model the system as a graph to make it easy to transverse relationships
Data	Million Song Dataset and the Last.fm dataset
Project Goals	- Model data as a graph - Implement and automate an ETL process - Store data in graph database
Technologies	AWS EC2 and EBS, Jupyter, Spark, Python (pyspark and py2neo libraries), Neo4j

Motivation

Challenges of implementing a good music recommendation system:

- How to make a good a playlist?
- How to recommend new songs or artists that don't have existing data?
- How to filter songs by things like mood?

➔ **Implement a graph model that exploits music interconnectivity**

30 million songs 2.3 million artists 1.6 billion playlist	Spotify
35 million songs	DEEZER
30 million songs	Google play
42 million songs	Music
1.5 million songs	PANDORA

The Million Song Dataset

Collected in 2011 from the Echo Nest API

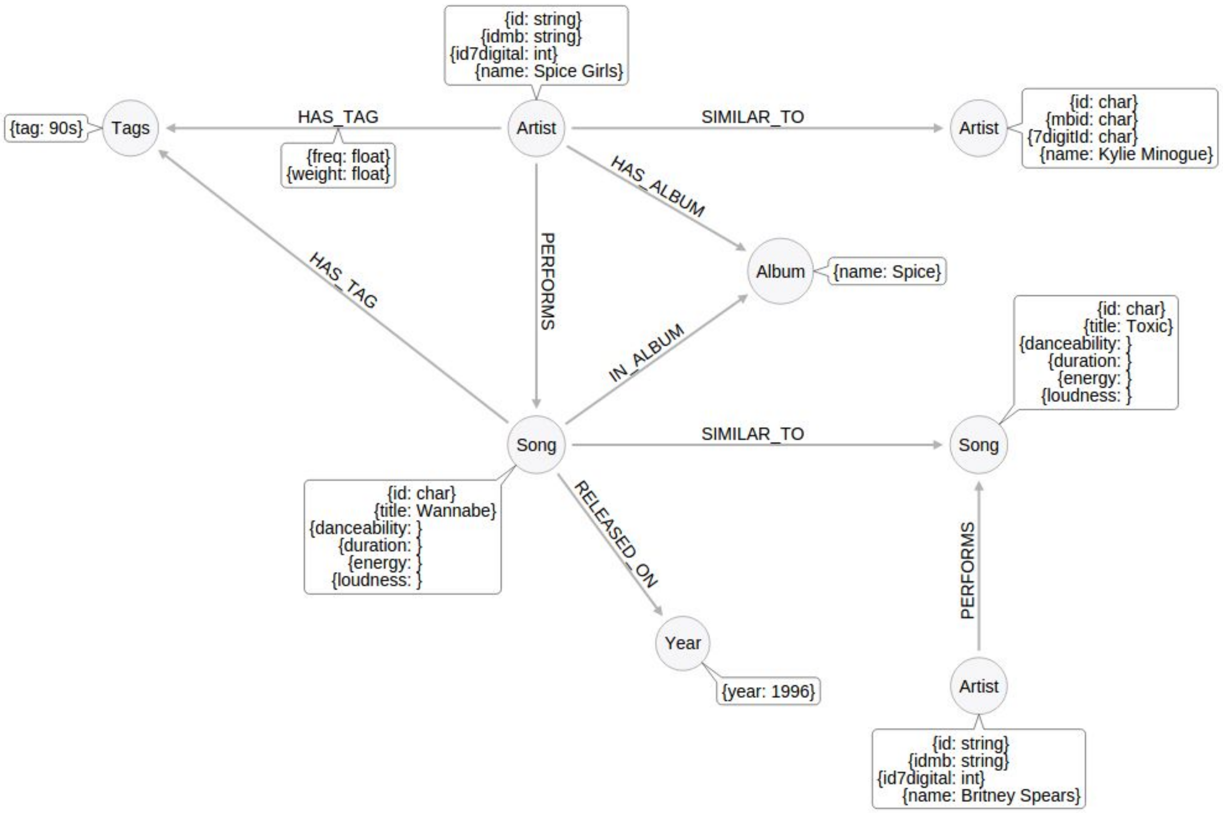
- 280 GB
- 45,000 unique artists
- 2.2 million artist similarity relations
- HDF5 files (one per song)

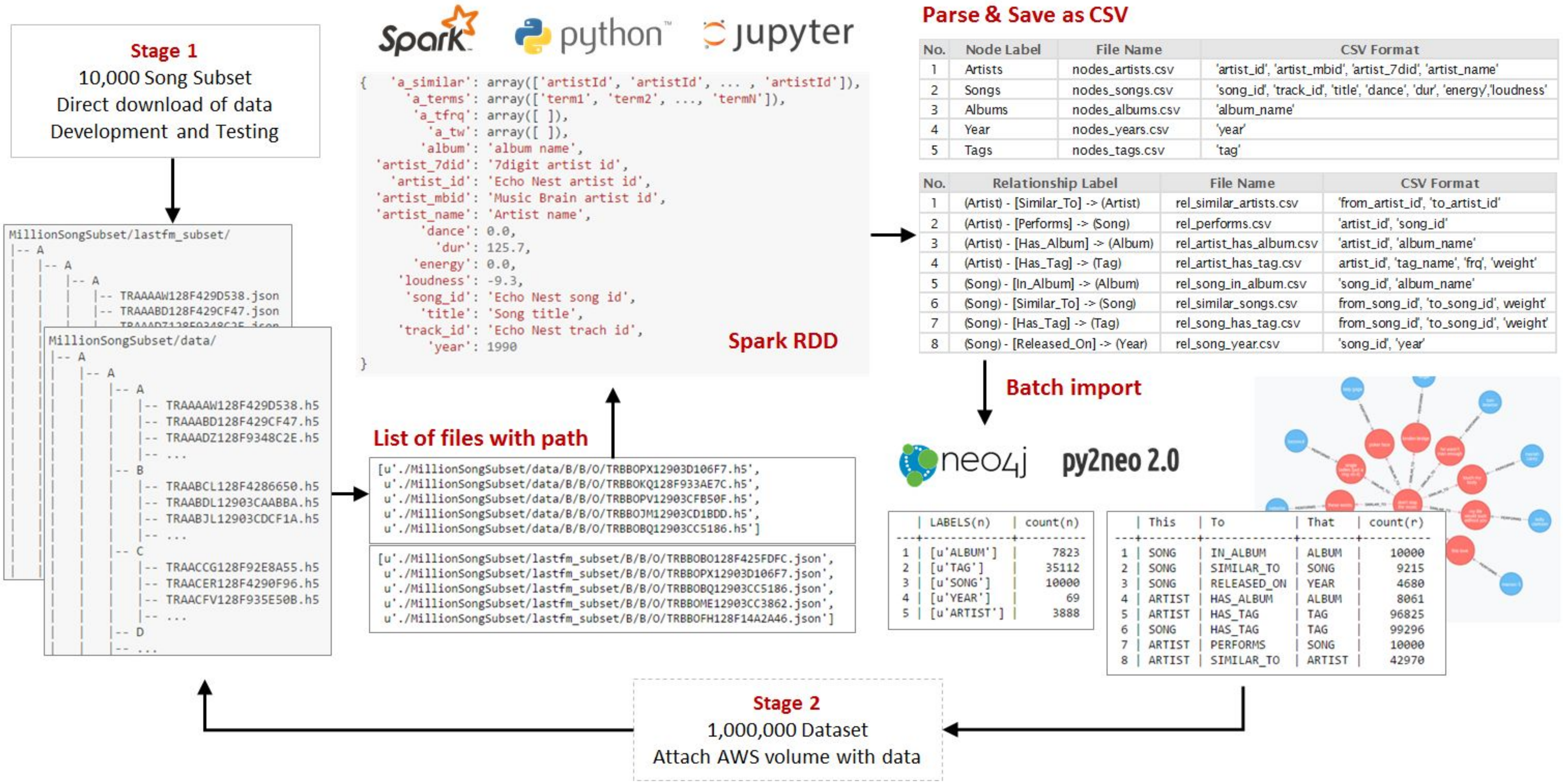
Complemented with lasft.fm dataset

- 56.5 million song similarity relations
- 8.6 million song-tag pairs
- JSON files

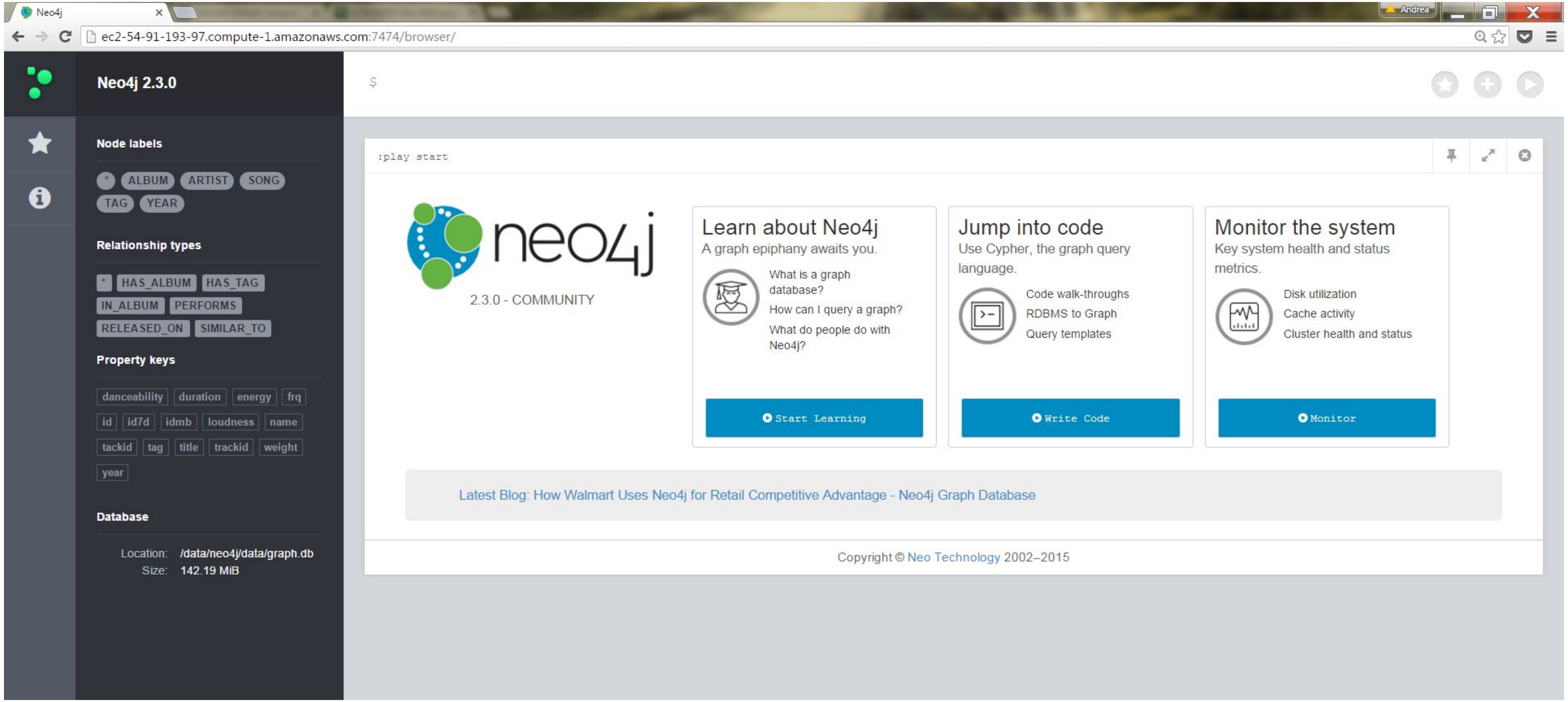
Field Name	Type	Description
artist 7digitalid	int	7digital ID
artist id	string	Echo Nest ID
artist mbid	string	Musicbrainz ID
artist name	string	artist name
artist terms	array string	Echo Nest tags
artist terms freq	array float	Echo Nest tags freqs
artist terms weight	array float	Echo Nest tags weight
danceability	float	algorithmic estimation
duration	float	in seconds
energy	float	from listener point of view
loudness	float	overall loudness in dB
release	string	album name
similar artists	array string	Echo Nest artist IDs
song id	string	Echo Nest song ID
title	string	song title
track id	string	Echo Nest track ID
year	int	MusicBrainz song release

Graph Model


{id: string}
{idmb: string}
{id7digital: int}
{name: Spice Girls}
{tag: 90s}
Tags
HAS_TAG
{freq: float}
{weight: float}
Artist
SIMILAR_TO
Artist
{id: char}
{mbid: char}
{7digitId: char}
{name: Kylie Minogue}
HAS_ALBUM
PERFORMS
HAS_TAG
Album
{name: Spice}
IN_ALBUM
{id: char}
{title: Toxic}
{danceability: }
{duration: }
{energy: }
{loudness: }
Song
SIMILAR_TO
Song
{id: char}
{title: Wannabe}
{danceability: }
{duration: }
{energy: }
{loudness: }
RELEASED_ON
PERFORMS
Year
{year: 1996}
Artist
{id: string}
{idmb: string}
{id7digital: int}
{name: Britney Spears}


Stage 1
10,000 Song Subset
Direct download of data
Development and Testing
Spark
python
jupyter
{ 'a_similar': array(['artistId', 'artistId', ... , 'artistId']),
'a_terms': array(['term1', 'term2', ..., 'termN']),
'a_tfrq': array([ ]),
'a_tw': array([ ]),
'album': 'album name',
'artist_7did': '7digit artist id',
'artist_id': 'Echo Nest artist id',
'artist_mbid': 'Music Brain artist id',
'artist_name': 'Artist name',
'dance': 0.0,
'dur': 125.7,
'energy': 0.0,
'loudness': -9.3,
'song_id': 'Echo Nest song id',
'title': 'Song title',
'track_id': 'Echo Nest trach id',
'year': 1990
}
Spark RDD
Parse & Save as CSV
No. | Node Label | File Name | CSV Format
1 | Artists | nodes_artists.csv | 'artist_id', 'artist_mbid', 'artist_7did', 'artist_name'
2 | Songs | nodes_songs.csv | 'song_id', 'track_id', 'title', 'dance', 'dur', 'energy','loudness'
3 | Albums | nodes_albums.csv | 'album_name'
4 | Year | nodes_years.csv | 'year'
5 | Tags | nodes_tags.csv | 'tag'
No. | Relationship Label | File Name | CSV Format
1 | (Artist) - [Similar_To] -> (Artist) | rel_similar_artists.csv | 'from_artist_id', 'to_artist_id'
2 | (Artist) - [Performs] -> (Song) | rel_performs.csv | 'artist_id', 'song_id'
3 | (Artist) - [Has_Album] -> (Album) | rel_artist_has_album.csv | 'artist_id', 'album_name'
4 | (Artist) - [Has_Tag] -> (Tag) | rel_artist_has_tag.csv | artist_id', 'tag_name', 'frq', 'weight'
5 | (Song) - [In_Album] -> (Album) | rel_song_in_album.csv | 'song_id', 'album_name'
6 | (Song) - [Similar_To] -> (Song) | rel_similar_songs.csv | from_song_id', 'to_song_id', weight'
7 | (Song) - [Has_Tag] -> (Tag) | rel_song_has_tag.csv | from_song_id', 'to_song_id', 'weight'
8 | (Song) - [Released_On] -> (Year) | rel_song_year.csv | 'song_id', 'year'
Batch import
neo4j
py2neo 2.0
MillionSongSubset/lastfm_subset/
|-- A
|-- A
|-- A
|-- TRAAAAW128F429D538.json
|-- TRAAABD128F429CF47.json
MillionSongSubset/data/
|-- A
|-- A
|-- A
|-- TRAAAAW128F429D538.h5
|-- TRAAABD128F429CF47.h5
|-- TRAAADZ128F9348C2E.h5
|-- ...
|-- B
|-- TRAABCL128F4286650.h5
|-- TRAABDL12903CAABBA.h5
|-- TRAABJL12903CDCF1A.h5
|-- ...
|-- C
|-- TRAACCG128F92E8A55.h5
|-- TRAACER128F4290F96.h5
|-- TRAACFV128F935E50B.h5
|-- ...
|-- D
|-- ...
List of files with path
[u'./MillionSongSubset/data/B/B/O/TRBBOPX12903D106F7.h5',
u'./MillionSongSubset/data/B/B/O/TRBBOKQ128F933AE7C.h5',
u'./MillionSongSubset/data/B/B/O/TRBBOPV12903CFB50F.h5',
u'./MillionSongSubset/data/B/B/O/TRBBOJM12903CD1BDD.h5',
u'./MillionSongSubset/data/B/B/O/TRBBOBQ12903CC5186.h5']
[u'./MillionSongSubset/lastfm_subset/B/B/O/TRBBOBO128F425FDFC.json',
u'./MillionSongSubset/lastfm_subset/B/B/O/TRBBOPX12903D106F7.json',
u'./MillionSongSubset/lastfm_subset/B/B/O/TRBBOBQ12903CC5186.json',
u'./MillionSongSubset/lastfm_subset/B/B/O/TRBBOME12903CC3862.json',
u'./MillionSongSubset/lastfm_subset/B/B/O/TRBBOFH128F14A2A46.json']
| LABELS(n) | count(n)
1 | [u'ALBUM'] | 7823
2 | [u'TAG'] | 35112
3 | [u'SONG'] | 10000
4 | [u'YEAR'] | 69
5 | [u'ARTIST'] | 3888
| This | To | That | count(r)
1 | SONG | IN_ALBUM | ALBUM | 10000
2 | SONG | SIMILAR_TO | SONG | 9215
3 | SONG | RELEASED_ON | YEAR | 4680
4 | ARTIST | HAS_ALBUM | ALBUM | 8061
5 | ARTIST | HAS_TAG | TAG | 96825
6 | SONG | HAS_TAG | TAG | 99296
7 | ARTIST | PERFORMS | SONG | 10000
8 | ARTIST | SIMILAR_TO | ARTIST | 42970
Stage 2
1,000,000 Dataset
Attach AWS volume with data

Live Demo of Music Graph in Neo4j



Pros and Cons of a Graph Database

- ➔ Good fit for highly connected data where there tends to be more relationships than nodes
- ➔ Powerful for traversal queries (friends-of-friends)
- ➔ Cypher is an intuitive and easy to use declarative query language
- ➔ Neo4j High Availability enables horizontal scaling of reads, and also supports ACID transactions

- ➔ Scales vertically because graph structure is hard to partition (bigger data, bigger server)
- ➔ Finding all nodes of the same type is more expensive than in a relational database

Future Work

- ➔ Build a sophisticated algorithm in the back-end that determines the similarity edges in the graph
- ➔ Run personalized Page Rank to make recommendations to users
- ➔ Add new songs, node types, or more properties by connecting to the APIs of Echo Nest, Musicbrainz, 7digital, or other music APIs
- ➔ Add nuances like mood tags, artist collaboration, song covers, live versions, etc.

Takeaways

- ➔ Prototype for migrating music data into Neo4j
- ➔ ETL sounds really simple, but can be really complex. Both the data and the technologies used will drive the ETL process
- ➔ When handling a larger dataset, thinking about DAGs and the overall process makes a huge difference
- ➔ It’s also important to understand the limitations of each technology, and know how to tune configurations to improve performance

Questions?