Performance and Analysis of a Graphical Database on Social Media

Rachel Mendiola
Richel Leung
University of California, Riverside

Intro/ Abstract

Graph databases are growing in popularity since many data can be represented more naturally and easier as a graph, increased efficiency of queries (traversals versus joins), and the flexibility that graphs offer over traditional relational databases. In 2016 Facebook introduced Reactions, which allow users to respond to posts with the emotions love, wow, haha, sad, and angry, in addition to the original like. This information is available about each post in the Facebook graph API, and can be used, amongst other things, to perform sentiment analysis on the content of the post. In this work we introduce a dataset of Facebook news posts, containing information on Facebook reactions for each post, that we model in a graph database. We analyze each post to identify keywords and develop a graph data model for this dataset. We identify changes that can be made to this model, and we explain what this model is useful for. We use the model and dataset to report the performance of neo4j graph database manager on different common queries that are relevant in data mining topics today.

Work Contributions

In this work we will contribute the following:

- A new representative dataset of Facebook posts from a large variety of news pages that includes attributes for the number of likes, different reactions, and shares
- Provide methods and suggestions of how to clean the dataset
- Introduce a graph data model that allows for popular data analysis to be performed on the data
- Benchmark the neo4j database manager on several different tests that are relevant in modern data analytics

Method

In this work we used two separate machines, one of which we site for most of the benchmarks because the other was too slow.The specs for the machines are summarized below. Note that computational resources were very limited for us, which might provide useful information for others who are look to do data analytics using neo4j on cheap machines.

- Intel® Celeron(R) CPU N2830 @ 2.16GHz × 2 with 4GB RAM and 78 GB Disk
- Intel® Core™ i5-4690k @ 3.5 GHz with 8GB RAM and 512GB Disk

We created a web crawler mine Facebook News pages based on [9] to collect different information on posts and store it into csv files. After this, we clean this data, according to standard practices. We created a graph data model for this dataset and loaded this data into a neo4j graph database. Following this, we examine different relevant queries in data analytics and report the performance of neo4j on this new dataset.

The platform we used to model our graph database is Neo4j version 3.12. Neo4j is the most widely used graph database management system. It is implemented in java and is accessible from software written in other languages using Cypher Query Language. Cypher Query Language allows for expressive and efficient querying and updating of a property graph. The language was created so that complicated database queries can easily be expressed. We utilized Cypher Shell which is a command-line tool installed along with Neo4j in order to query the data.

Dataset

The dataset contains a total of 1,158,805 posts, from 142 different news sites across Facebook. The data sometimes has missing values, most commonly in the section of ‘status_message’. Depending on the database used, the data may need further cleaning. For instance, different databases may have different constraints on the data format it accepts. The biggest issue we’ve had in this dataset is accounting for stray quotes that cause issues with special characters, and create null values. Any statuses that were published before Facebook introduced the reactions feature (2-24-2016), will have zeros for all the reaction values. The page_name is a unique string identifier for the page that it was posted on. The ‘status_id’ is a unique identifier for the individual post and can be used as a key. Below is a diagram that shows the organization of the data, as well as the specific information collected about each post.

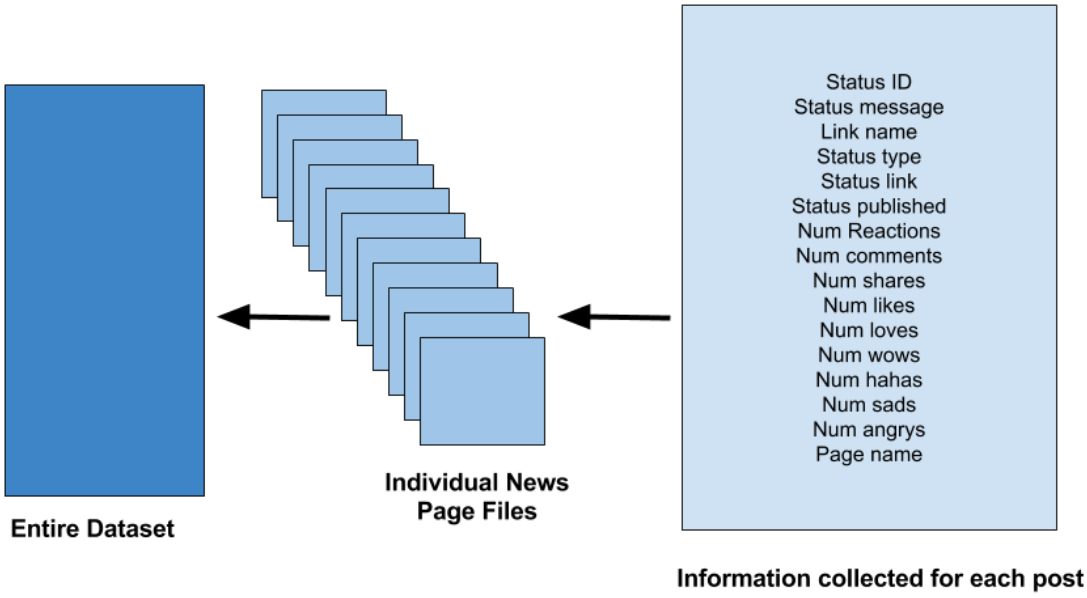


Figure: Information about the content and organization of the dataset. The dataset is split into different csv files based on the different pages. Each file contains around 1000-7000 different status posts. Each status post has several fields listed above. In the section after this we discuss a way to model this data in a graph database.

From this dataset, we have created representative subsets of this data. We are motivated to do this because we want to generate datasets with significantly different structure, such that we can compare how neo4j does on both of these datasets. We are also motivated to do this by our limited processing power, limited ability to clean all the data, and desire to make the dataset more manageable for analysis operations.

As an example of how this data can easily become unmanageable, the NPR post shown below has 26 words in it's 'status_message', 13 of which should be considered 'keywords'. The keyword nodes, and the relationship nodes grow by a large factor with each post, and in the worst case each post may create new keyword nodes for each word in its status message.

```
10643211755_10155341792606756,"With its rich variety of beans and long history of cultivation, Indonesia is building a coffee
culture — and a pride in it — that is truly homegrown.",Indonesia Wakes And Up And Smells Its Own Coffee — Then Drinks It, link,
http://www.npr.org/sections/thesalt/2017/02/26/517035587/indonesia-wakes-and-up-and-smells-its-own-coffee-then-drinks-it?
utm_source=facebook.com&utm_medium=social&utm_campaign=npr&utm_term=nprnews&utm_content=20170226,2017-02-26 16:45:00, 336, 15,
44, 304, 25, 4, 3, 0, 0
```

Figure: An example line from the NPR csv file from our dataset. In the worst case, the query loading the data will create a keyword node and relationship for every word in a status message.

The first subset of the dataset, which we will call dataset 1, is composed of the 10 of the most popular Facebook new sites: BBC, CNN, Fox, Yahoo, nytimes, cbs, abc, the hill,

Huffington post, and Reuters. From each of these pages we use the most recent 1,000 posts. We think this will create an interesting graph because there should be many more connections between keyword nodes of different posts. This is because we sample from the most recent posts, from top sites, which should tend to cover similar topics versus a randomly generated set. Our motivation is to create a more densely connected network to benchmark.

The second subset of the dataset, which we will call dataset 2, is generated by randomly selecting 10,000 posts from the entire dataset. The motivation here is to generate a dataset that will draw from many different pages as sources, but not necessarily be as dense as dataset 1. We think that this will give us three very different sampled data sets that will make for a good comparison in neo4j.

Name	Computer Setup	# Posts	# Properties set	# Nodes	# Relationships
Dataset 1	PC 1, celeron	10,000	134,858	26,380	120,981
Dataset 2	PC 2, i5	10,000	18,256	7,488	11,971

Table. Two data subsets we created from the larger dataset. The first one contains 10,000 of the most recent posts from the most popular 10 datasets, the second is a random sample over the full dataset.

Graphical analysis of the data shows many promising results. These results can be used in sentiment analysis of people in different social groups of different topics. Sentiment trends can be mined from the graph and query into certain keywords that are most likely associated with certain types of reactions. Other models can be inferred from the data. One such inference is the placement of each page node in relation to surrounding nodes. In the overall graph on the data, one can observe page clusters that are either compacted together near the center, or outside separated away from the other pages. Page nodes that are clustered together in the center have more relationships among each other, meaning that they post about similar topics and keywords. Outliers do not post topics that are similar to other pages. There are many possibilities that arise from this relationship. One possibility is that outliers are posting about unique topics that most other pages do not talk about. Another possibility is that the distant relationship from other posts and keywords can be related to fake news. Future analysis of the data and trends can use these relationships as a basis on determining whether or not posts are about fake news.

Data Model

A graph database stores data in the form of a graph, using directed edges to represent relationships between data, where queries and data manipulation are represented by graph operations [1]. Many real-world data can be modelled in a graph, including biological applications and social networks [2]. In this way the relationships between data are the main focus of the model, whereas in the relational model we enforce relationships by means of foreign keys [1]. Typically, relational database management systems only model data as sets of tables and columns. They carry out complex joins and self-joins when the dataset becomes more inter-related. Because of this, there are times when queries become too complex and expensive to run. This also hinders the utilization in real time when performance is key even when the total dataset size increases. Because of this, graph databases are becoming much more compelling as they enable programmers to get a sense of the size of the connected data that exists in their dataset.

Graph databases also have a lot of flexibility, since the schema do not have to be fixed. For instance, if we wanted to gather information about the comments that were posted on these Facebook posts, we could easily add nodes and relationships between posts and comments. These are some of the reasons that compel industry to migrate towards using graph database models for a wider range of data.

We model our data in such a way that we are able to show the relationship among public Facebook pages and the posts that they post. Each public page is represented by a Page node that has a posted relationship with all of it's posts. Each post node contains statistics of the message such as ID, the message itself, and the number of different kinds of reactions, likes, and shares. A visual representation of this information can be found below.

Choosing to model our data this way will allow us to concisely query and understand the relationship between different news pages. Using the keyword nodes we can effectively compare what topics news pages report on, find posts that report about similar keywords, or posts that should report about the same topics via the date nodes. As mentioned above, in combination with this way of modelling the data, we can also analyze how certain posts make people feel and therefore we can infer the sentiment of the post. Visual representations of the two graphs for dataset 1 and dataset 2 using this data model can be found below.

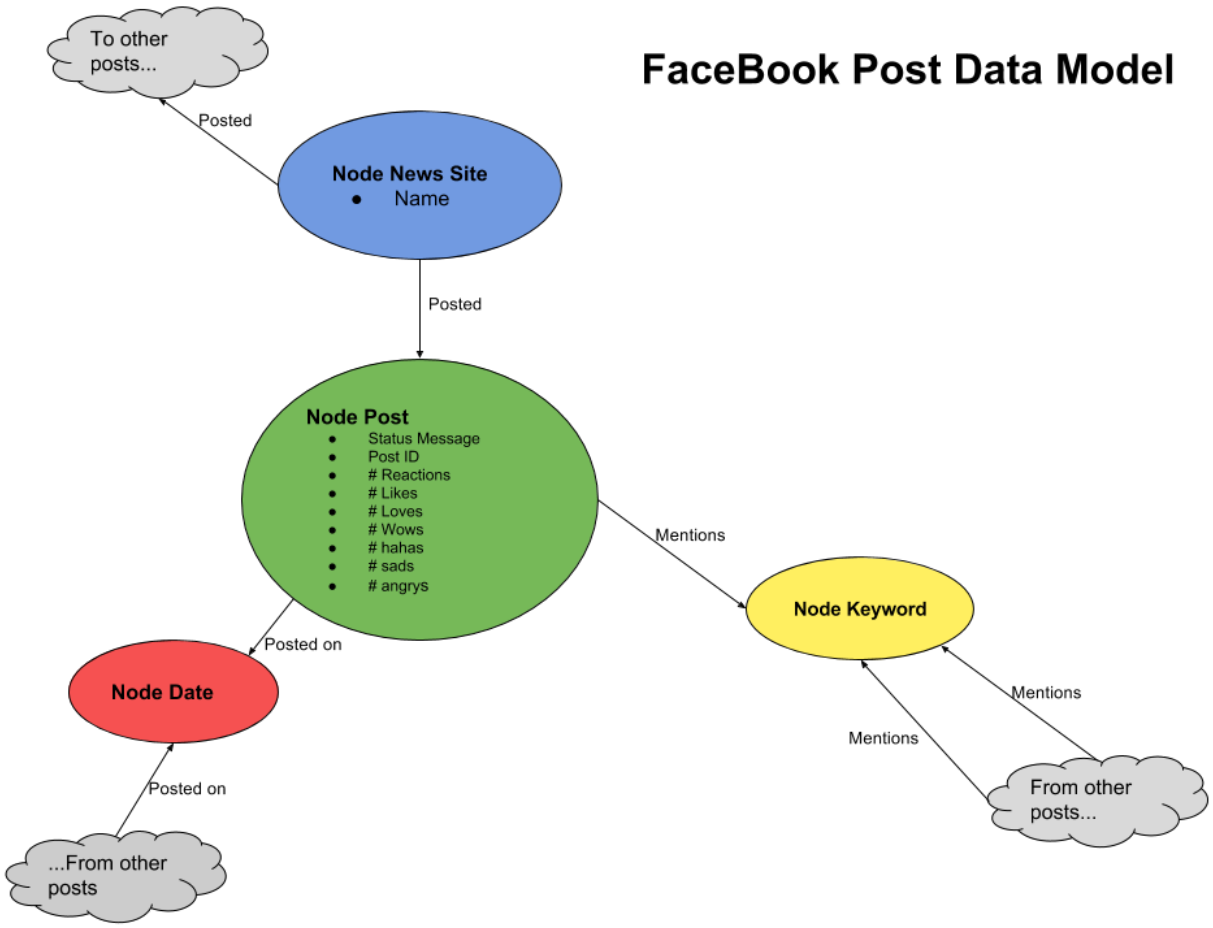


Figure. The graph data model we created for the news posts from Facebook. The design centers around Post Nodes, which are posted by news sites. News sites create different posts. Several keyword nodes are extracted from the post nodes, and many other posts can point to these same keyword nodes. Post nodes have a date, and date nodes can have connections to any other post. We constrain the data by requiring each Post Node to have a unique post ID, and each News Node to have a unique name.

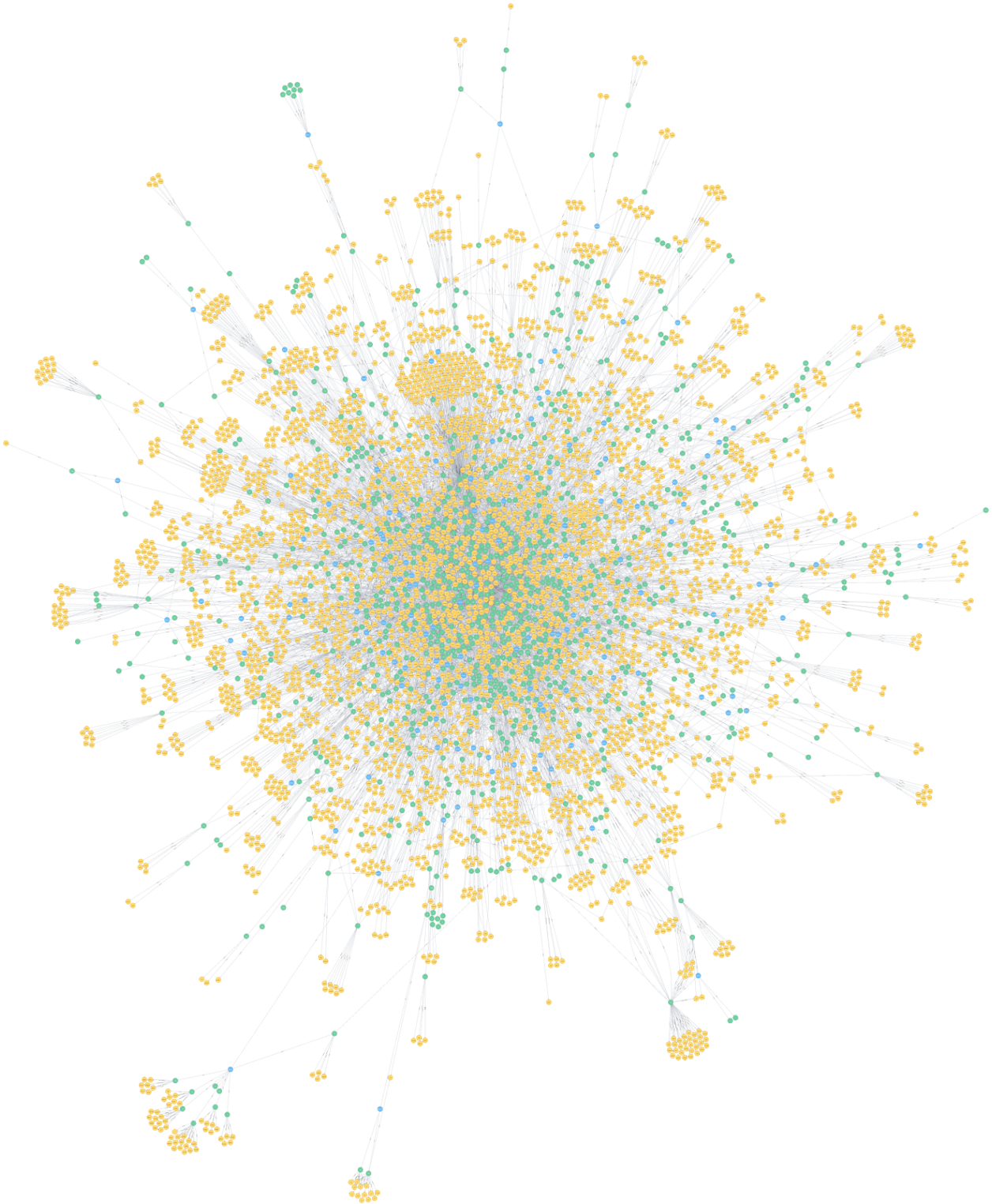


Figure. A visual representation of Dataset 2, the complete graph database of 10kPosts.csv. The green nodes are post nodes, the blue nodes are page nodes, and the yellow nodes are keyword nodes. Note how many posts have bunches of keywords around them, that are not connected to anything else. Because Dataset 2 was a random sample, many of the posts are from different time periods and are likely to not cover the same topics.

Figure: Plotting of 1,000 Page and Keyword nodes and their relationships from dataset 1. In this visual the green nodes are posts and the yellow nodes are keywords. In this dataset many of the post nodes share keywords, which we can see by the many connections between keyword nodes.

Data Cleaning

Before inserting the data into neo4j database we perform different forms of data cleaning and transformation on our dataset. In this section we offer a discussion of the different techniques we did to prepare it for the database, including handling duplications of nodes, transforming titles into keywords, and handling special characters.

In our data model we do not want duplications of nodes in two different cases: equivalent news pages and very similar keywords (think: “trump” versus “Trump’s”). To deal with the first problem, we use a cypher keyword ‘MERGE’ which looks over all the other nodes before creating a new one if it doesn’t exist. To address the second problem, we cleaned the data by removing trailing whitespace and punctuation, and forcing all letters to lowercase. This will allow us to use MERGE on the keyword nodes to make sure articles containing the same keywords point to the same node. The second problem may further be minimized by using a stemming software program, which is not considered in this work.

Part of our data model involves creating nodes for ‘keywords’. When creating ‘keyword’ nodes the goal is to identify and extract important information from the status message that can help us identify what the post is about. We want to be able to extract the most meaningful words from an article, so that we can formulate connections between different posts. Finding keywords in text has been a commonly researched topic in natural language processing. There exist methods for finding keywords that involve statistics, machine learning, and heuristics, and a complete summary can be found in [6]. In this work we take a very simplistic approach, and look at the frequencies of words in the english language to create a list of words to filter out from a post title. The idea of keep of list of words to filter from text is borrowed from natural language processing, and is known as stop words. Stop words are words in a dataset that cause confusion or noise, and can be filtered out of the dataset [2]. Research has been done in identifying and managing lists of stop words [8]. For our project we reuse this concept by identifying a set of stop words, and filter them out of a post title. After this we will be left with only the important words, which we will then consider keywords. In this work form our stop list from merging a high frequency word list [7] with a the stop list [10], to create a dictionary of words that we remove from a status message.

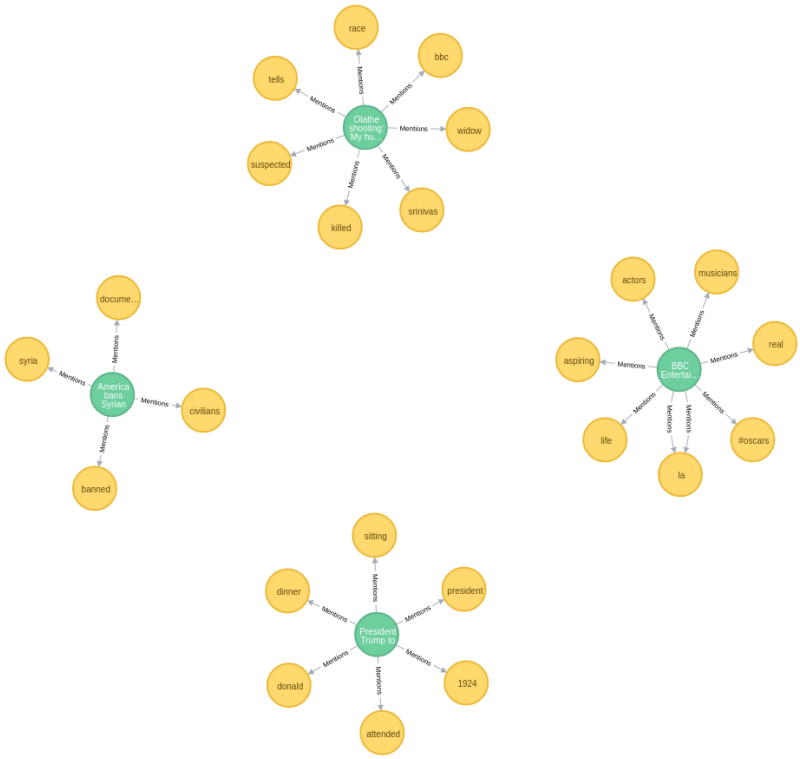


Figure: Example keywords extracted from news posts in dataset 1. We can see that after filtering there is still some noise in the keyword nodes, but for the most part we see only meaningful information extracted from the post title.

Lastly, there were many posts that ended up using special characters that could not be processed or that would end up ultimately modify other entries and prevent the data from being transferred to the database. In this case, manual search for specific special cases were needed in order to delete the section in the csv. Examples of this were posts that contained special characters that made new sections that would cause null values for all attributes in the csv until the file reached the next post.

Tests

We perform four different tests on the database: loading data, basic queries, shortest path finding, and most similar nodes. The parameters for each of these tests is described below, along with any special considerations in our experimental setup. To come up with these tests we drew inspiration from [2] which describes methods to benchmark databases, and [4] in which they benchmark graph databases in different categories including loading of data, insertion of edges, shortest path search, and breadth first search.

Test 1 - Loading the Data

We report the amount of time that it took to load the different datasets into neo4j. The data is committed to the database with a batch size of 100. An important thing to note is that when we loaded our data, we are using two merge statements. When trying to create a new node, a merge statement searches all other nodes to see if it already exists beforehand, if it isn't found the node is created. In the results section we will explore the effect of the MERGE keyword when inserting data, and report its performance on dataset 1.

Test 2 - Queries

A set of basic queries that request various numbers of different types of nodes from the dataset. These queries are design to see how efficient it is to ask the database about different attributes in the dataset. We have also included some queries that might be more useful in data analysis, these include shortest path finding and finding similar nodes.

Returning top k mentioned keywords in post nodes - for summarization

This operation returns the top k keywords based on how many mentions they have. At a high level, this information gives us a summary of the graph database, in that it helps us understand what most articles were written about in the dataset. We can analyze this information by looking at the number of likes, hahas, wows, angrys, etc, to see which topics overall have made Facebook users feel a certain way. Below is a top-10 query on the most used keywords from dataset 1.

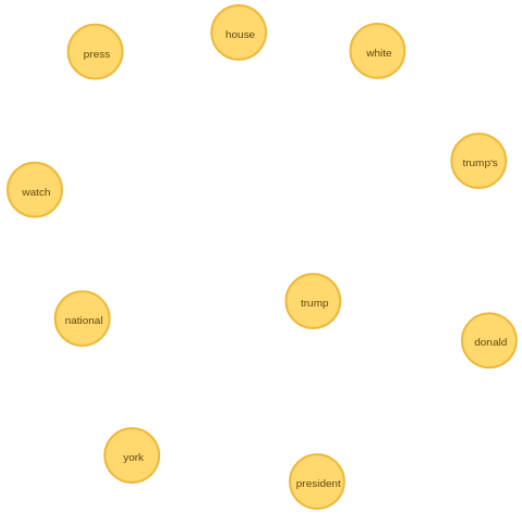


Figure. The top 10 keywords used in dataset 1, based on the relationship 'mentions'.

Return posts with > k likes

This operation returns a set of posts that have more than k likes. This query can also be modified for any other reaction: angrys, comments, hahas, likes, loves, reactions, sads, shares, or wows. This information can be used to query posts that have a high content of a certain reaction and find keywords that all of them have in common. This allows for grouping of reaction posts and also observe the overall demeanor that a page most likely posts about.

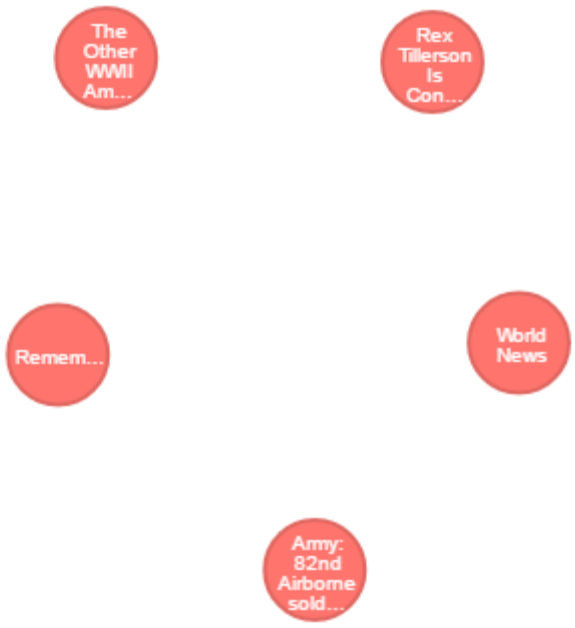


Figure. Finding posts in which the amount of 'sad' reactions is > 1000 in Dataset 2. An example of how we can exploit powerful analytics using basic queries on this data model. We can find the sources of these news sites be exploring their parent relationships, or we could follow the children nodes to the keywords.

Finding the shortest path

A common operation performed on graph databases is to find the shortest path from one node to another. In our case, we may want to find the shortest path from one news page to another, so that we can see the relationship one news page has to another. We can also do path finding to analyze the relationships between pages such as, looking at different paths that two news pages might have in common.

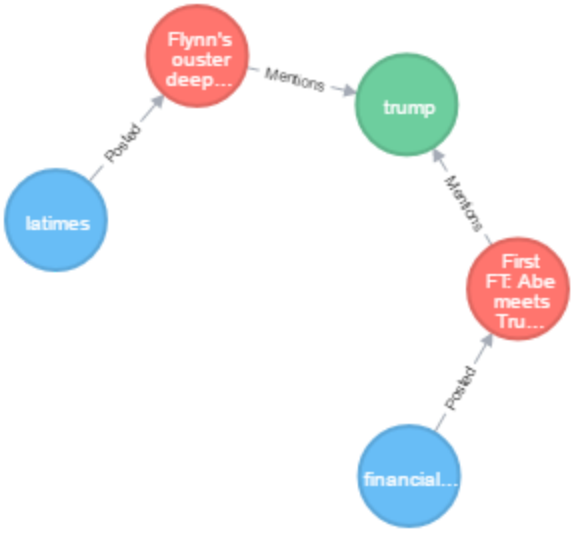


Figure. Finding the shortest path from latimes to financial times. When returned the shortest path, we can see the topics that connect these pages.

Finding Similar Nodes

This operation involves taking an input node, and querying the database to find a set of nodes that is similar to it. In our case this has many different applications. We might be interested in seeing which news pages are similar to each other, or we might be interested to see which keyword nodes are the most similar.

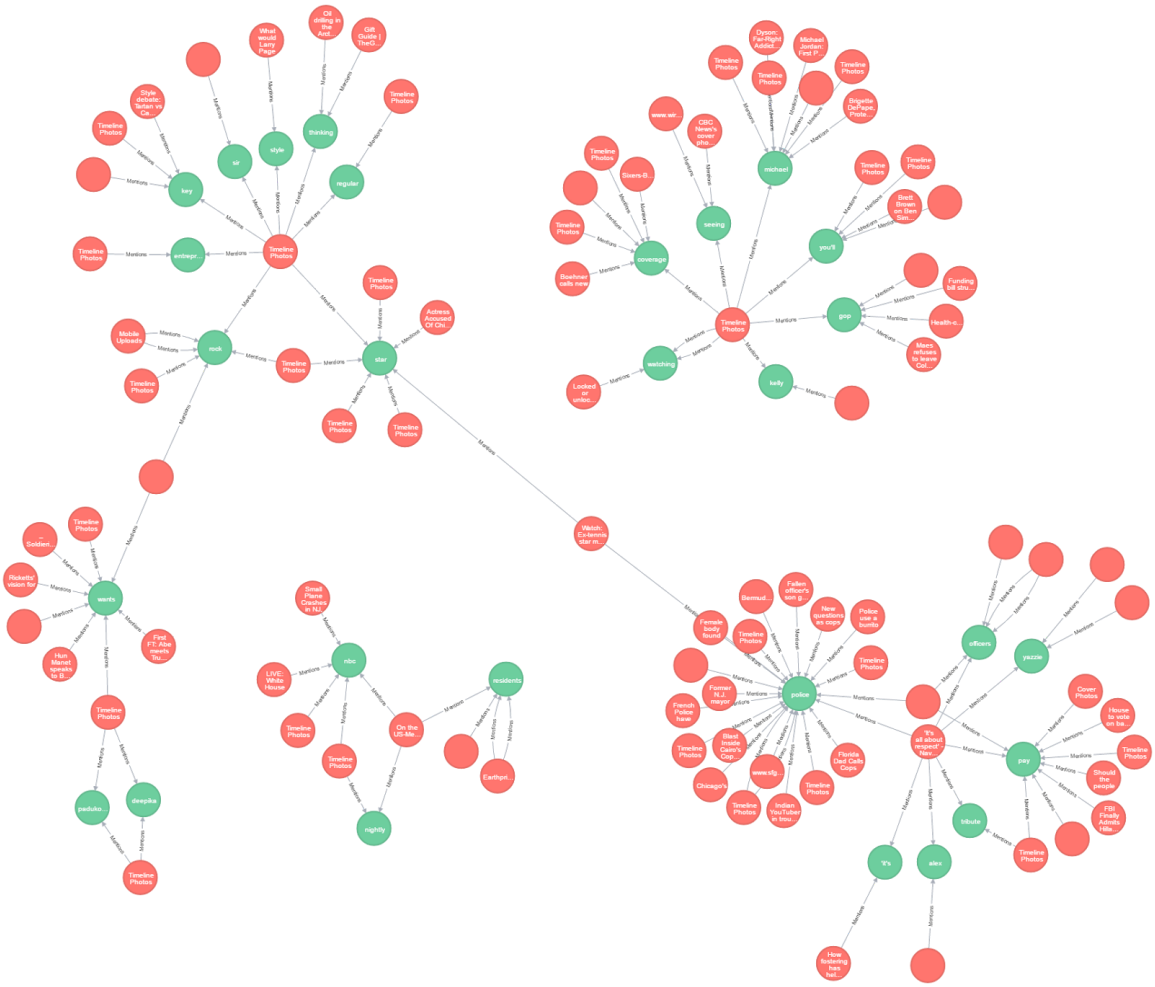
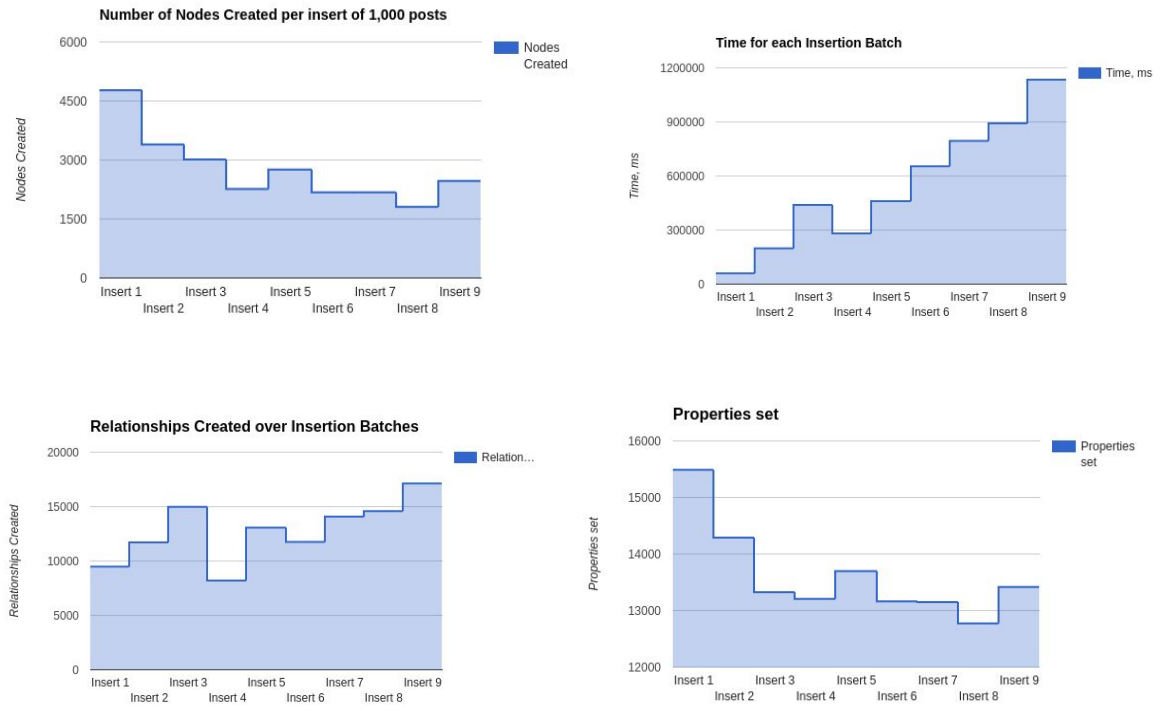


Figure. Finding 100 nodes that have the most similarity based on relationship to given nodes based on keywords on Dataset 2

Results

Test 1. Loading Dataset 1 into neo4j

This dataset was loaded into our first computer, the celeron, in batches of 1,000. In the figures below notice how as we insert more data into the database, we create less nodes, but the database manager takes even longer to insert the posts. Recall that we are using MERGE statements in our insertion query to join keyword nodes and page nodes, which is what causes the database to perform this way. Queries that insert by creating new nodes every time will not have this problem, which is something users of neo4j should take into account before using the MERGE keyword and designing the data model. Because dataset 1 is so densely connected, it takes much longer to load into neo4j than dataset 2, this can also be caused by using a slower processor. Loading dataset 2 into our second computer created 7488 nodes, set 18256 properties, and created 11971 relationships in 29557 ms.



Figures. The number of nodes, relationships, and properties set, over the course of inserting 10,000 posts into computer 1. We can see that using the MERGE operator makes insertions less efficient.

Test 2. Queries

Query	Dataset 1 (on Celeron Processor)	Dataset 2 (on Intel i5)
Return keyword nodes with more than 25 mentions	8862 ms 714 rows	36 ms 1 row
Return keyword nodes with more than 10 mentions	7587 ms 1852 rows	18 ms 47 rows
Return keyword nodes with more than 5 mentions	6304 ms 3270 rows	16 ms 235 rows
Return posts with > 5000 likes	13460 ms 2732 rows	4 ms 35 rows
Return posts with > 500 likes	26981 ms 7043 rows	5 ms 234 rows
Return posts with > 50 likes	29442 ms 9780 rows	4 ms 557 rows
Return the top 25 mentioned keywords	6694 ms	7 ms
Return the top 10 mentioned keywords	5797 ms	6 ms
Return the top 5 mentioned keywords	5849 ms	6 ms
Shortest path finding	(From page FoxNews to cnn) 1 node, 2 relationships 4542 ms	(From page “IFreakingLoveScience” to “HuffPostUk”) 9 nodes, 8 relationships 3 ms

Returning 100 similar nodes to a given node based on keywords	4156 ms	4 ms
Returning 500 similar nodes to a given node based on keywords	5387	8 ms
Returning 1000 similar nodes to a given node based on keywords	6779 ms	21 ms

From the results above we see that returning more nodes doesn't necessarily indicate more time for the database to respond. Depending on which node the database starts on, it could take a different amount of time to return a result. It seems that the tighter the constraint is on the query, the more time it will take. For instance, on both datasets it takes longer to find the keywords with more that 25 mentions, than it does to find keywords with more than 10 mentions.

Conclusions

From our research we were able crawl for public page information and posts and store them in a graphical database to be observed. These datasets crawled contained information for a large variety of news pages and encompassed multiple statistics on the posts themselves. We then cleaned the data such that it was ready to be easily loaded into a graph database. From there, we were able to create a graph data model that allows for popular data analysis to be performed on the data. We have shown some examples of data analytics that can easily be expressed around our data model. Lastly, we tested our model on a number of different tests and queries that are relevant in modern data analytics.

References

[1] Angles, Renzo, and Claudio Gutierrez. "Survey of Graph Database Models" *Universidad De Chile* (n.d.): n. pag. Web.

[2] Boral, Haran, and David DeWitt. "A Methodology for Database System Performance Evaluation." (n.d.): n. pag. Web.

[3] Bruggen, Rik Van. "Demining the “Join Bomb” with Graph Queries." *Neo4j Graph Database*. N.p., 09 Dec. 2015. Web. 21 Mar. 2017.

[4] Jouili, Salim, and Valentin Vansteenberghe. "*An empirical comparison of graph databases*." (n.d.): n. pag. Web.

[5] Kaur, Jasmeen, and Vishal Gupta. "Classical Approaches for Aqueous Extraction." *Extraction Techniques in Analytical Sciences* (n.d.): 37-47. Web.

[6] Klatt, Benjamin, Klaus Krogmann, and Volker Kuttruff. "Developing Stop Word Lists for Natural Language Program Analysis." *Developing Stop Word Lists for Natural Language Program Analysis ** (n.d.): n. pag. Web.

[7] Vgel. "First20hours/google-10000-english." *GitHub*. N.p., n.d. Web. 21 Mar. 2017.

[8] Vicknair, Chad, Michael Macias, Zehndong Zhao, Xiaofei Nan, Yixin Chen, and Dawn Wilkins. “*A Comparison of a Graph Database and a Relational Database” ACM SE 2010*. New York: ACM, 2010. Web.

[9] Woolf, Max. "How to Scrape Data From Facebook Page Posts for Statistical Analysis." *Minimaxir | Max Woolf's Blog*. N.p., n.d. Web. 21 Mar. 2017.

[10] *XPO6*. N.p., n.d. Web. 21 Mar. 2017.