Domain Visualization: An evaluation of a domain graph database
Group 19

Sajaval Choudrey sajaval@kth.se
André Danielsson anddani@kth.se
Kim Hiltunen kimhil@kth.se
Sai Man Wong smwong@kth.se

Wednesday 11th May, 2016

School of Computer Science and Communication (CSC)
Royal Institute of Technology KTH, Stockholm, Sweden

Abstract. Web data systems, such as search engines, are heavily used today by a vast number of people, who demand higher scalability and more complex features by the systems. An indexing approach is common for search engines. To ensure efficiency, good algorithms and techniques for storing must be used to process the large amount of data in modern systems. Yahoo! was one of the first who utilized Web indices in form of human-curated "directories". It was proved to be useful, but was not maintainable due to it being difficult to scale in relation to the human power it demanded. Eric Brewer presented a straightforward and automated approach of Web indices in the paper Combining Systems and Databases: A search Engine Retrospective. The approach suggested was a set of Web crawlers that downloads web data and indices which are then used to compute relevant scores. Consecutively, queries are then applied by a front-end service to present the results by its score.

The purpose of this project was to, based on the latter mentioned approach, develop a search engine that shows the relations between domains in a graph. This gives a sense of the structure of links, how domains are related and how frequently links are pointing between domains. The implementation presented in this report is using a python library called "Scrapy" for scraping the web. To store the data, Neo4j will be used as a graph database.

Key words: visualization, domain names, web crawler, graph database, neo4j, scrapy

1 Introduction

Graph databases are great tools to visualize and analyze data with relations in mind. The nature of the graph database allows fast querying of data as well as a visual representation of the result of the query. By combining this type of database with a scraping framework, data relations can be discovered and presented in a clear way.

The aim of this project was to investigate the tools that search engines use to gather data. We wanted to explore the process of gathering data by scraping the web, store this data in a database and interpret the data. We concentrated on presenting the data in graphs instead of using an algorithm to analyze the data.

For the implemented code we chose to not add any special attributes for each of the nodes. The nodes in the graph will only have the registered domain name (its unique ID) as its only attribute. This is the only required data to fulfill our goal of presenting the relations between domains.

2 Method

The literature used for this project was the article ”Combining Systems And Databases: A Search Engine Retrospective” by Eric A. Brewer[1]. In chapter 2 the article delves into crawling, indexing, servers and queries, and gives a basic definition of the terms. We were inspired by the article in particular the section about crawling, thus tempting us to try out the phenomenon on a fundamental level. In order to do this we decided to try to visualize inter-domain relations.

Neo4j uses the Cypher query language to fetch data from the graph database. This language is similar to SQL in respect to the keywords used. One keyword that is used frequently in Cypher query is the `MATCH` keyword. This describes a specific pattern (relationship) we want to find in the query. An evaluation on the performance of Cypher compared to MySQL by F Holzschuher and R Peinl[2] showed that Cypher queries can be more efficient than MySQL if we want to find single relations. If multiple queries must be executed to find the sought after data. Cypher queries are also more readable and maintainable, according to F Holzschuher and R Peinl, in regards to relationship queries.

3 Implementation and Result

Scrapy is a framework for the Python programming language that allows the creation of web crawlers to scrape the data of a set of web pages. This was used to get all the links (HTML a-tags) from each page, extract the url with XPATH query and add the relation to the database. Using a predefined set of rules, the web crawlers can be restricted to specific domains and specific links to follow. The restriction to the same domain, for each web crawler, was important to get the correct data to the database.

Because the main purpose of using a graph database is to store nodes, relations and properties. This project uses a graph database to find relations between domains. Therefore, domain names are represented by a node in the graph and a link between one domain to another is represented as a relationship between the two domains. This way, when queries are made against the database, the relations between the domains can be analyzed. The Py2neo library for Python was used to open the connection to the neo4j database and insert data.

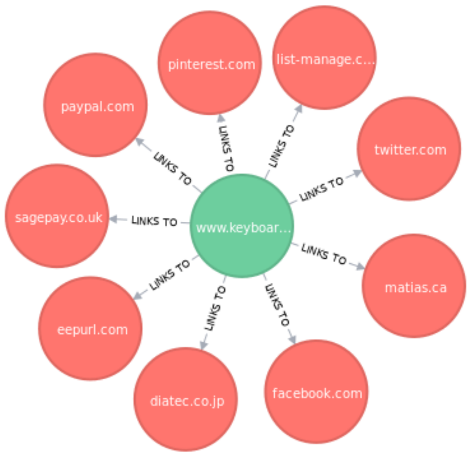

Fig. 1. Domain visualization of related domains for a relatively small website

Another reason neo4j was picked was because graph databases allow easy traversals between entities and relations. Instead of having foreign keys, graph databases gives each node a list of references to its relations. This makes the process of finding a relation between two domains (essentially a graph traversal problem) faster with a graph database compared to a regular relational database.

The graph in Fig. 3 shows the relations of a small website. To get this result, you can use the Cypher query: `MATCH (n) RETURN n`. This will print out all the nodes and relations. The result can be very large and not relevant for larger websites unless you want to find the number of different linked domains or to find if some specific domain has a link on the website. In this project, using two nodes of interest is more relevant to achieve our goal.

Fig. 2 shows a query that can be used to find two nodes and all links both nodes have a relation with. Using modifications of this query, we can find the number of shared links, if a specific link (domain name) is shared or not and domains that are not shared.

```
START a=node(*), b=node(*), c=node(*)
MATCH p1=(a)-->(c), p2=(b)-->(c)
WHERE not((a)-->(b))
AND a.domain = 'URL1'
AND b.domain = 'URL2'
return a, b, c;
```

Fig. 2. Cypher query for finding all common linked domains.

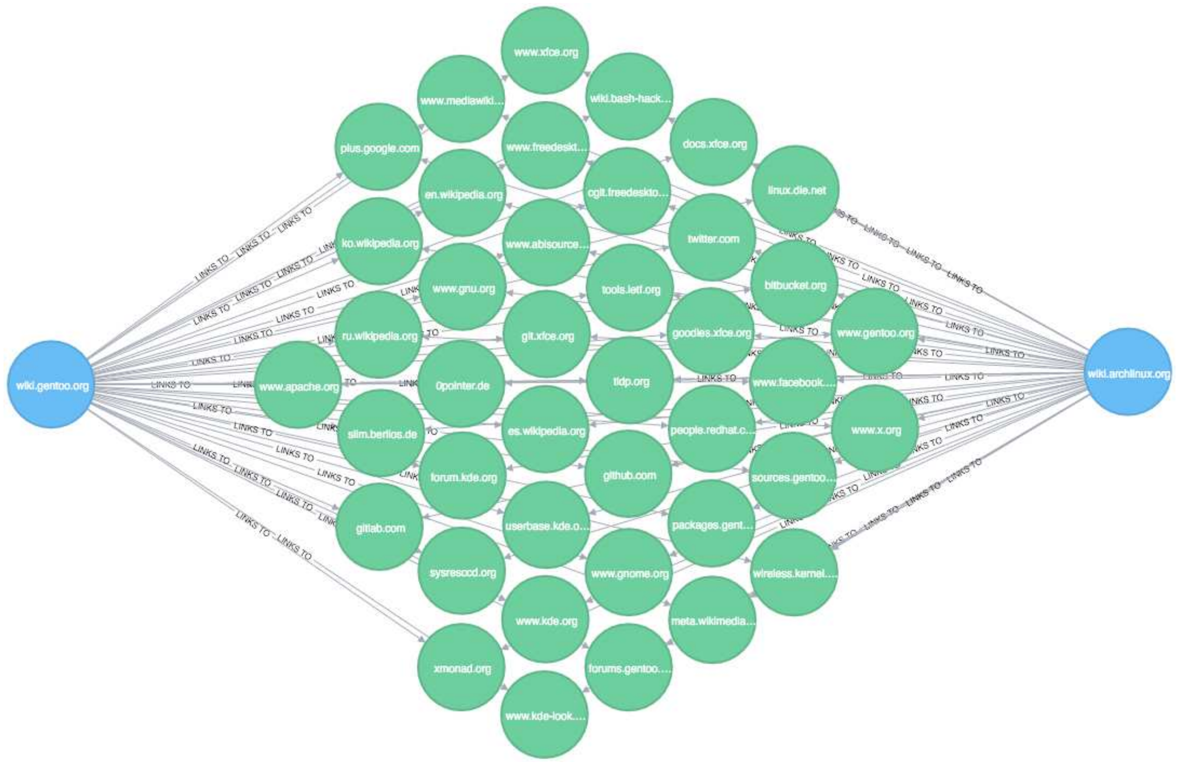

Fig. 3. Graph returned by the query in Fig. 2

4 Discussion

The Scrapy framework allowed for a fast and robust implementation of a web crawler, which enabled extraction of relevant domain-relationships. In order to interpret the extracted data, the Python library Py2neo was utilized to output the data into the graph database neo4j. Consecutively, the Cypher query results was then visualized in form of nodes and edges.

Instead of using a web interface for the queries we used the command line. The advantage of using a web interface would be that all stored data would've been available in one place and that data would be easier to access, for example from a web server. This would also allow a better user experience and usability for the user.

One suggestion for a future expansion of this work is to allow users to specify a larger scrape depth. Instead of being limited to one domain, the web scraper could be specified to be able to continue going outside the chosen domain to get more results and more distant inter-domain relationship. Another task that could be performed in the future for a better experience is the usage of a web interface as described above. The inter-domain relations gathered from the collected data could be analysed in different ways and used for different purposes. An example of how the relations might come in handy is, when used in the field of advertising. Advertising can with the help of the relations be adapted to a certain target group and distributed across domains that are interlinked. There are various similar areas of application where the program can be beneficial.

The method of scraping to receive data (*crawl*, *index*, *serve*)[1] was done by using Scrapy combined with Neo4j. The scrapy spider *crawled* the web, gathering each link and passed the data to the Scrapy pipeline. The purpose of the pipeline was to *index* the data, parsing and inserting the data in the correct format to the database. Lastly, Neo4j was used to *serve* by executing queries to present the data to the user.

The usage of the Neo4j database was overall a very good experience. The database interface was clean and user friendly. The only challenge was to get a grip on how the cypher query language works. More specifically the challenge lied in understanding the MATCH keyword which required a somewhat deeper knowledge of the cypher query language. However, the scope of the project was fairly limited and the challenge was proportional to the scope therefore the overall usage of the database can be seen as a nearly seamless process.

5 Conclusion

Working with this project gave a good fundamental insight into "Scraping" and "Graph databases". As far as finding inter-domain relations goes these two are very fast and efficient and at the same time they are two concepts that are fairly easy to understand. When using "Scraping" and "Graph Databases" together, they form a good tool for creating an overview of inter-domain relations.

References

1. Eric A. Brewer. Combining Systems and Databases: A Search Engine Retrospective. Readings in Database Systems, Fourth Edition, (2005).
2. F Holzschuher, R Peinl. Performance of graph query languages: comparison of cypher, gremlin and native access in Neo4j. Proceedings of the Joint EDBT/ICDT 2013 Workshops. ACM, (2013).
3. Vicknair, Chad, et al. A comparison of a graph database and a relational database: a data provenance perspective. Proceedings of the 48th annual Southeast regional conference. ACM, (2010).