POINT EVIDENCE EXPLAIN LINK

Qu: An overview of the application of NoSQL databases to recommendation systems The benefits of using it in these areas Focus on providing details of actual examples of the application of NoSQL databases to Netflix's recommendation system which type of NoSQL applies and why What problems will arise

TODO:

- make sure each benefit is linked to an aspect of recommendation system

- link in benefits of neo4j into implementation

- re look at implementation in consideration to the overview points

- write intro and conclusion

- add in facts and statistics

- CUT CUT CUT

- REF REF REF

Introduction

In the world of big data, - companies need to stay competitive, the brand loyalty is becoming less and less - in a way that benefits both them and and their customers - to keep them interested and wanting more we should use these profiles that the companies get out of the big data - to show that we know those customers, we know what you like, don't waste your time and our time or money in continuely searching for what you 'think' you want and instead, here is what you want based on what we know about you.

When you look at the companies that use graph databases as the core of their various products, you start to see some of the most successful companies in the world right now. Walmart for their shopping, PayPal for their peer to peer payment, Google ads and page ranking. Common themes throughout these companies is real-time, connected, big data, and Netflix is no exception. This again is where graph databases come into their own.

Why graph against other NoSQL

When thinking about implementing or changing a part of a system that is oriented around data, one has to ask a multitude of questions to understand which direction is best to take. What is the data? What it does represent? What do you want out of it? What is the rate of change to the data? Is it structured? If we were to apply these questions to a recommendation system of a company

the size of Netflix, we would soon realise that the amount of data is huge, offering 4,335 movies and 1,609 TV shows (Lovely, 2016), it is ever-changing, growing from 29 million subscribers worldwide in 2012 to 86 million in 2016 (Statista, 2016), it's largely unstructured and it is highly connected. Graph databases is the only system out of NoSQL or relational models that is able to deal with the outcomes of the questions asked and not only does it deal with those outcomes effectively, it specialises in them. ADD SOMETHING SUGGESTING NEO4J OUT OF GRAPH DATABASES, it being market leader, already used by top companies bla bla

For the sake of comparison, the way in which it would be implemented in a relational model would first consist of all the tables for each of the element effecting the recommendation. This would include: users, movies and ratings where ratings would act as a join table. The problems arise however when you consider the questions presented about the data that we're using. From 2015 to 2016 third quarter's Netflix has seen a rise in 17.5 million subscribers worldwide (Statista, 2016) all of which are rating movies constantly. You can see how much the data grows and effects the growth of other data within that relational model to the point of saturation. To search these tables in real-time and to update them, to refresh the join tables constantly to reflect the new or modified data would be so computationally expensive it wouldn't make sense. Facebook discovered this first hand when working on there recommendation system for suggesting new friends, finding that when working with connected data, graph databases perform 1,000 times faster than relational database models (Rathle and Eifrem, 2013). Relational-models are known to be good at data that is well-structured and understood data, that doesn't require a lot of change and has minimal connectivity.

In the 2014 Cassandra summit, Netflix admitted to using Cassandra for 95% of their database usages, this includes everything from storing meta data, customer data and even the recommendation system (Kalantzis, 2014). Along with them being able to create their own specialised adaptation of Cassandra due to their enormous development power, this is because other types of NoSQL approaches do help deal with some of these constraints. Largely, NoSQL databases by nature are able to deal with unstructured or semi-structured data very well. However, I believe this is the least important of the constraints when working with recommender system. The edge that graph databases has over any NoSQL database model is the fact it specialises in connected and correlated data. This due to it being based on graph theory. Graphs represent how the data is correlated and connected. The meaning of data comes from relationships between it. In relation to Netflix recommendation system, graph databases give a 360 view of the user, movie, rating or description and all the data that is linked to it specifically, with the ability to exclude other data of the same subset, so we can start to understand everything about these aspects from all angles (De Marzi and Neo4j, 2015). These connections are allowed to grow and shrink accordingly and the more we learn about the user and the movies, the greater the relevance and relationships are to one another and how best this can be exploited for the

gain of the company and user alike.

When dealing with big data, it is all too easy to be overwhelmed and to not be able to make sense of the data that is there due to the sheer amount of it. When such problems arise, Neo4J has the abstract benefit that it is very intuitive to design and implement. The schema that can easily be drawn out can be directly implemented and transformed into the actual graph database (Lyon and Neo4j, 2016). This opposed to other NoSQL and relational models that still require the mentality of data to be organised into tables, rows, columns and for each to be carefully indexed. Not having to do this allows for greater freedom and flexibility in the model of the database and ultimately benefits the system manipulating such data. It also helps the details not to be overlooked and the cognitive unload of not having to worry about implementation of the schema into strict rules allows the focus to be on what the data now represents.

This freedom and flexibility of accessing and manipulating data in Neo4J is reflected in the speed in which these operations can be performed. Being able to process the data that is in the graph database quickly is by far one of the biggest benefits when offering recommendations in real-time. This is possible due to the feature 'index-free adjacency' that Neo4J offers. That is, it doesn't need an index to be able to travel between nodes that are connected by a relationship (Merkl Sasaki and Neo4j, 2015). This saves expensive index look-ups that you have to use when working with relational-models and other NoSQL models. What is unique to Neo4j and graph databases is that no matter the growth in data, the index-free adjacency allows 'constant-time traversal', meaning that the performance of the traversal of nodes will not change (Merkl Sasaki and Neo4j, 2015). This is especially important when you consider the rate of which the data held by Netflix grows. eBay found this to be true when they migrated their delivery system from MySql to Neo4J, their 'solution is literally thousands of times faster than the prior MySQL solution, with queries that require 10-100 times less code.' (Neo4j, 2016).

Neo4J realised when dealing with data that is so interconnected that older SQL syntax based language wouldn't suffice, they instead built their own query language, Cypher. Having an own language specifically for this own database type is extremely powerful. SQL is built and optimised for relational database models, Cypher is built and optimised for Neo4J. Cassandra also has its own version of CQL, but it is not to be confused with Neo4Js CQL as they completely opposite in syntax and purpose. Cassandra's CQL is almost identical to SQL in syntax and purpose, it's only difference to SQL is that it is optimised for querying Cassandra databases. However Neo4J Cypher language contrasts that of the traditional SQL syntax and purpose on almost all fronts. Cypher main focuse is pattern matching which is an important concept when working with big data as you want to be able to find patterns to make sense of what is there. This concept can directly be applied to recommendations as users build up patterns of movie watching habits and from those patterns one can draw out conclusions about what other movies to watch. The syntax of the language is

designed to be close to natural language as to help with the intuitiveness, as described in previously, it shifts the focus to what you actually want from the data, not how to do it. It also means that the queries are a lot more concise. The Neo4J website gives examples of Cypher queries that are half or a quarter size in line count to their SQL equivalents (Hunger, Boyd, and Lyon, 2016). They are small scale examples but as data and their connections grow exponentially, so do the SQL statements, unlike Cypher queries that stay concise. The fact that they're so concise also means that they're easy and fast to implement, they're manageable and maintainable. When working with data that represents users and their movie preferences in relation to ratings and other users, one could see how large an SQL statement would start to become due to the amount of joins and clause conditions, so it is easy to realise the benefits of having a language built specifically to solve these issues.

As aforementioned, Netflix currently relies on Cassandra for almost all of its databases needs. This was chosen because Cassandra was built up from the ground with distributed data-centre storage as its focus, allowing constant up-time and availability (Kalantzis, 2014). This was an important step in the companies success as they made the transition to focus more on their streaming services rather than DVD rental and has since helped in their competitiveness. However, we are only focusing on one part of their massive ecosystem to try to improve it and to keep that same mindset, that all data needs to be available at all times, Neo4J has clustering as a core feature of its product. It allows massive throughput by having multiple machines serve requests in parallel. Data redundancy is covered by the data being replicated throughout the instances, ensuring there is no loss if one fails. With multiple instances, requests can be re-routed if an instance is unavailable to ensure that request is served and that the data is always available (Borojevic, Lyon, and Neo4j, 2016). All of these are built in to the core of Neo4J making the recommendation system in-keeping with the always-available ethos of Netflix.

Being able to read and right in as ACID transaction is a feature which is specific to Neo4J (Chao, 2016). This is extremely important when working with high volumes of sensitive data in real-time as data cannot be lost. With the amount of data that a Netflix recommendation system requires to be manipulated in real-time, it is easy to see why this feature is vital to the success of it. However, you could argue the success of any database model would rely on how they store the data, that is after all the pure concept of a database. Neo4J offers native graph storage which allows for data consistency and also allows for all the performance optimisations to be possible (Chao, 2016). Simply put, if Neo4J didn't have this option, then none of the benefits above would be possible and therefore a different database system would have been argued for the recommendation system of Netflix.

Most NoSQL handle the growth and shrinkage of data easily, they can read and write to their data in great speeds, they can also provide valuable data distribution and protection. But they cannot solve the issue of connected data

nor give the power to traverse the data to find and present the relationships that connect them together like graph databases can.

Implementation

There are two main models to consider when implementing a recommendation system. These are commonly known as Content based and Collaborative filtering. These are both methodologies to find ways of recommending items to the user but both take different approaches to do so.

Content based recommending, also known as cognitive filtering or item-to-item filtering, is a model whereby the recommendations are made up of the comparison between multiple properties of the product. If two products are seen as similar based on the content that they share, then one can assume if a user liked one of the two, then they would also like the other (Recommender-Systems, 2012b). These comparisons build up the relationships between items and work to build a picture about a subset of items in the system as a whole. An important feature is that the user isn't made to apply the attributes to the data, this is done by the system itself.

In context to graph databases and Netflix, the nodes would be the movies themselves as well as the descriptions. The relationships would link the movies to their respective descriptions. A query can be then used to find nodes of the same type that link to the same descriptor to find similar items. An example would be return movie nodes that have the same 'genre' descriptor which would effectively recommend a movie similar based on content (Neo4j and Bachman, 2015). But, problems can arise from this approach. This is only recommending movies that are similar, that are in the same genre, by the same director and so on. This works to an extent, but people are not one-dimensional, in very few cases does a single person only like a single genre of movie. The user would soon grow tired seeing the same genre of movie recommended and this would be detrimental to Netflix, as the trust in the recommendation starts to fail, other movies aren't discovered which leads to a misuse of the system and potential loss of a customer.

If movies are going to be linked on descriptions, then a standard, generalised and formalised taxonomy of description terms and classes is needed to rid the system of subjective bias. But even when standards are set, it can lead to scalability issues if one would like to add more descriptors to the items then it has to applied to all items otherwise it would never match on this property (Recommender-Systems, 2012b) . Other methods can be applied to the bias problem by scoring across movies by many people or algorithms can balance out this bias but that can be an expensive computation given the size of the dataset. Thus the bottlenecks of this approach are found. To circumnavigate this problem, the data that is applied to describe the movie can be from the

source of the item itself, e.g. the movie production office provides the genre of the movie, not critics of the movie and having a list of actors that appear in the movie which provides a factual approach to describing the item without bias. This in combination with collaborative filtering leans towards the most widely adopted hybrid approach, taking the best bits out of both and merging them together.

The collaborative filtering approach, also known as social filtering, is were the system builds profiles of users based on their behaviours and actions and then compares the user against other users of the same group to get the recommendation (Recommender-Systems, 2012a). As I see that graph database is the way to build the recommendation system, you can see how it would work well in this case as the system can easily find correlations between users to make the recommendations. Regularly on Netflix, users are asked to give ratings to the movies and in doing so it improves the correlation between groups as you can find users who also gave similar ratings to the movies, from this you can work out what genre that movie is, what actors are in it and the system can make the assumption that if such movie was rated highly by one person and a 'similar' movie was rated the same by someone of the same group then the movie can be a candidate for recommendation between the two users. Thus, the nodes of the graphs would include the users, which would be group together, and relationships could be 'RATED', 'LIKED', 'WATCHED' to all the other nodes of movies and descriptions (Neo4j and Bachman, 2015). This creating view of the user and their preferences in-line with other users of the same group. The use of a graph database, Neo4J in particular, makes these connections easy with the speed it can traverse the relationships and you can see that when working with this highly correlated and connected data that a graph database is the only solution.

Although this is a more natural way to approach the problem and can help to create a wider picture of the user and their preferences in relation to other users. This can again fall into the trap of of making users too one-dimensional (De Marzi and Neo4j, 2015). User preferences often change over time and this needs to be accounted for. A way to solve this is a commonly used technique known as 'Nearest neighbour' where the similarities between users of the group are ranked so the people with the highest similar taste in movies could be used against each other for recommendations (Recommender-Systems, 2012a). If this calculation of similarities was done across whole groups on a regular interval, you could account for the change in preferences per person as they move in similarity among one another and adjust the groups accordingly. This can also be used with machine learning techniques so that the score retrieved can be further improved upon. Although this is a step in solving the problem, it isn't the answer. With a user base as large as Netflix, regulary tweaking the groups can be expensive process that continually has to happen for it to work effectively. The large user base also provides the problem of presenting the data real-time as all groups have to be scanned and processed to make the recommendation, this can be aided with the help of pre-processing and optimisation techniques but it

is still no doubt a extensive and expensive operation. It also takes time for the profile of a user to be built unless input is asked from the user at registration, which is not the case for Netflix (Netflix, no date).

Most companies, however, try to adopt a combination of the two, merging data that describes the item along with the similarities between users. I would say this is the best course of action for Netflix recommender system as you get the advantages of both to gain valuable insights to the users, the movies that they like and the movies themselves. I also believe that Netflix could use meta-date to build up profiles on users quicker and more accurately. Information such as what time of the day they watched, what device they watched from and where they were when watching could give implicit ratings which would broaden the picture of the user and to find patterns of viewing habits and to make a richer image of the user.

Deciding which recommender methodology to use is only part of the system, the architecture of the system as a whole incorporates many other aspects. These, as defined by GraphAware (Neo4j and Bachman, 2015), can be broken down into the following steps: Model, Scoring, Blacklists, Filtering, Post-processing and Logging. All of these aspects of the system can be built with Neo4J. The Model is the actual graph database. It is how the data is represented in all of its entirety. This would be stored in the native graph storage to allow for index-free adjacency as well as other performance optimisations and modelled in conjunction with decision of the recommender system described above, as you need to take into consideration the constraints that they involve such that the data is modelled in a way that can be queried effectively using the specified methodologies. The engine of the recommendation system in Neo4J would be written in one of the languages that is compatible with it: Java or Scala. Although it is been said that Netflix services are increasingly being changed from Java to Python (Samson, 2013) this does not mean that the service that handles the database and the engine built to query it is a part of it as they can encapsulate it to work in conjunction with Python or any other language that they use in different services. If they have already built their engine in Java, then the development overhead of incorporating the same language to build the Neo4J would be kept to a minimal. Scoring is an important feature of engine that Neo4J is able to implement. It allows for ranking of the results which provides the bases of how good a recommendation is, formulas and functions can be applied on top of this such as the Pareto distribution to account for differences of few being greater than differences of many within groups of users, as described in the collaborative filtering recommender, and scores themselves (Weisstein, 2005). Post-processing techniques can also add to the weighting and score of the retrieved results, an example of this would be awarding extra points to the recommendations from user segment that are the same gender of the person requesting the recommendation thus refining the recommendation more so. These parts of the engine can constantly be improved using data analysis techniques and machine learning due to it being a numerical figure and that the success of the recommendation being logged. When filtering, two

technique are used on the results retrieved by the CQL query to refine them further, namely blacklisting and filters. Blacklisting is common technique among many areas that rids any data retrieved against a pre-defined 'list'. An example of this in recommendation systems would be the ability to exclude movies that have already been watched by the user. A filtering technique such as removing all results that are returned by users in the same group located in different countries, as Netflix only provides certian media in certain countries they would not apply to all users of a multinational segment. Finally, the benefits of logging allows the system to keep track of many aspects of the process throughout its journey through the engine and after the recommendation has been made. A notable example would be to measure whether or not the user actually used the recommendation, if so the wieghts and scores can be updated accordingly to further improve the engine. Thanks to the speed in which Neo4J allows graphs to be traversed, all of this can be done in real-time to present recommendations to the user as and when needed. However, techniques such as pre-computing would commonly be used to find the most optimal times to do these calculations before they're needed in real time to balance the computational load of the process.

The power of this engine is obvious, once a query is made there are many layers of this engine that either strip, add, log or score the data retrieved from the model to produce only the most accurate results. This skeleton engine requirements produced by GraphAware, which is Neo4J's top consultancy firm, is in use for recommendation systems in LinkedIn and other top multinational companies (Graph Aware Limited, 2016).

to conclude bring together the main points, without explanation, to emphasis why its only graph and only Neo4J that can be used

http://brettb.net/project/papers/2007https://www.statista.com/statistics/250934/quarterly-number-of-netflix-streaming-subscribers-worldwide/ https://www.statista.com/topics/842/netflix/ (43.18 million subscribers in US alone)