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

https://www.statista.com/topics/842/netflix/ (43.18 million subscribers in US alone)

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.

- Different types of NoSQL and my proposed type - Why it is suited to Netflix - think about the data it is using - Connected data

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, it is ever-changing, largely unstructured and it is highly connected (FIND STATS). In order deal with the answers, the only real solution is to look at graph databases. It 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.

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. One would then assume that you would have to create join tables in order to traverse this data effectively to get the recommendation. 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 world wide (https://www.statista.com/statistics/250934/quarterly-number-of-netflix-streaming-subscribers-worldwide/) 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 discoved 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 (REF: https://neo4j.com/blog/why-the-most-important-part-of-facebook-graph-search-is-graph/). 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. However, what graph databases offer that is unique to them, is the way in which it can represent the data. The data and its meaning is unknown, of which the relationships between the data is where the meaning arises, the data and its structure is dynamic and needs to be traversed quickly. (FIND REF)In relation to Netflix recommendation system, graph databases give a 360 view of the user or movie and all the data that is linked to it specifically, as to understand everything about these aspects from all angles. It is allowed to grow and shrink accordingly and once this consideration is in place, 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.

In the 2014 Cassandra summit, Netflix admitted to using Cassandra for 95% of their database usages, this includes everything from meta data, customer data and even recommendation system (Christos 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 in general are able to deal with unstructured or semi-structured data very well. (REF SLIDES MAYBE) However, this is the least important of the constraints. The edge that graph databases has over any NoSQL database model is the fact it can handle connected and correlated data. This due to it being based on graph theory. When you think of graphs and what they represent, it is usually only ever to show how the data is correlated and connected.

When dealing with big data, it is all too easy to be overwhelmed, get lost and to not be able to make sense of the data that is there due to the sheer size of it.

Intuitiveness: An abstract benefit of graph databases is that they're very intuitive. - Schema that can easily be drawn out or thought of can be directly implemented into a graph databases, meaning no details are lost and the cognitive unload of not having to worry about these implementation problems means that the focus can be on what the data now represents and shows.

Speed: - being able to process the data that is in the graph database is quickly is by far one of the biggest benefits, Neo4J and offers 'index-free adjacency' traversing. That is, it doesn't need an index to be able to travel between nodes that are connected by a relationship. This saves expensive index look-ups that you have to use when working with relational-models and other NoSQL models (https://neo4j.com/blog/why-graph-databases-are-the-future/). Although these relationship queries can and are used in models that are not graph databases, what is unique to graph databases is that no matter the growth in data, the index-free adjanecy gives use 'constant-time traversal' meaning that the performance of the traversal of nodes will not change (https://neo4j.com/blog/why-graph-databases-are-the-future/). eBay found this to be true when they migrated their delivery 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.' (REF: https://neo4j.com/case-studies/ebay/)

A query language designed for connectedness: 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. - all about pattern matching - This is an important concept when working with big data. - 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 other patterns to form conculsions about what other movies to watch. - This pattern matching is also benefited because of connected data - As you want to be able to find various levels of relationships of the data and CYPHER is specifically made to do that - it means that the queries are a lot more concise - In an example from the Neo4J website, there is an typical and common SQL statement that is 6 lines and contains two joins, the matching CYPHER query is only 3 lines and the join is built into the language so naturally you often fail to realise it is there. ref https://neo4j.com/blog/sql-vs-cypher-query-languages/ - This is done a small scale, as data and their connections grow exponentially, so do the SQL statements, this is not the case for CYPHER. Again, the later example shows a complex SQL statement 16 lines, the equivalent in CYPHER is 4 lines. - The fact that they're so concise means that they're easy and fast to implement, they're manageable and maintainable. - The way in which is formulated is close to natural language - this helps with the intuitiveness as described in previous point, helps cognitive unload in order to only have to think about what you actually want from the data, not how to do it.

Native graph storage - Storing the database as a graph. This allows for data consistency and also allows for all the performance optimisations to be possible.

Fully ACID transactions which is specific to Neo4J. This is extremely important when working with high volumes of sensitive data in real-time as data cannot be lost.

NoSQL handle the growth and shrinkage of data easily, but cannot give use the power to traverse the data to find the relationships that connect them togther.

- research other points that i made in other NoSQL systems to contrast

- write intro and conclusion

- re look at implementation in consideration to the overview points

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 it would also like the other (ref:http://recommender-systems.org/content-based-filtering/). 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. From this you can see that to find a movie of a similar genre, the query would have to traverse relationship 'GENRE_IS' to the node 'genre' and back down the same relationship to a different movie, thus recommending a movie similar based on content. But, you can already start to see problems 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.

One can also see that if the data is needed to form properties of the items that the application of this data to form properties can be a problem. A real-world example of this would be Pandora's music recommendation system, which uses 'The music genome project' as a bases for description of the songs. This consists of a general taxonomy to describe the music of which comparisons can be made. In Pandora's business case, I noticed that the musicians tag the music themselves based on the properties provided by the taxonomy (REFRENCE). If an individual person is 'tagging' the item, it is subjective to the person and therefore can easily be bias. This same problem can be directly applied to Netflix. These descriptors that represent the item along with the terms also need to be stored, they need to be monitored and a standard has to be enforced to make sure that a sufficient amount of descriptors represent the item truthfully without being overloaded by more menial terms. Problems arise when assigning these descriptors, are they done manually or automatically? How can they be used in a meaningful way to describe the item truthfully, succinctly and rid of bias judgement? Methods such as setting a standard of descriptors can be put in place 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. 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(ref: http://recommender-systems.org/collaborative-filtering/). 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. This creating a '360' degree view of the user and their preferences in-line with other users of the same group. The use of graph database makes and Neo4J in particular makes these connections easy with the speed it can traverse the relationship of 'RATED' to the node

'MOVIE'. It is working with this highly correlated and connected data that makes it obvious that a graph database is the only solution.

This is a more natural way to approach the problem, in real life users are more likely to ask friends or people they know that have similar taste for recommendations. The benefit of this approach is that you can create a wider picture of the user and their preferences, they can belong to many different levels of groups and therefore a greater range of recommendations can be applied. However, this approach can again fall into the trap of being too one-dimensional, painting many people with the same brush. 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' (REFERENCE) 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. 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 improved upon without the adjustments to the criteria and formula made by human counterpart. 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 (REFERENCE REGISTRATION PAGE).

Although I've mentioned a company that uses content based, and Spotify that uses collaborative filtering (reference: http://www.slideshare.net/erikbern/collaborative-filtering-at-spotify-16182818/62), most companies 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 us is only part of the system, the architecture of the system as a whole incorporates many other aspects. These, as defined by GraphAware(reference: https://www.youtube.com/watch?v=KnQoNJJ4k8I), 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 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 following steps would build the engine that uses the model to produce the recommendation. 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 (ref http://www.javaworld.com/article/2078738/java-se/why-netflix-is-embracing-python-over-java.html) this does not mean that the service that handles the database and the engine built to query it is a part of it. One would assume that they would choose a language that is fit for the job, Java and Scala being one of these, and the fact 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 (ref youtube). 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 (FIND REFERENCE OF SPEED), 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 (ref: http://graphaware.com/) and other top multinational companies.

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

architecture item - > engine -> recommendation scoring - parrot (not right) function to adjust the differences - the difference to one and two moves liked is massive compared to the difference between 10 and 11 - these algorithms could be self learning to continuously improve

blacklists - build a list of movies already watch - dont want to re-recommend them - dont recommend the movie just watched filtering - after recommendations have been found post procsessing - alters scores on different criteria context - how many - how fast recording - logging - have such big data, need to benchmarking to figure out which is best and why

what type of recommendation engine - how would i naturally recommend someone a movie - get a friend that i trust (collaborative) - on a bigger scale can use techniques like k nearest neighbour to get people who are similar to you which would factor in the recommendation - updated daily or something, this way you get ever changing groups and therefore ever changing recommendations - problem is get a list of recommendations that is simple and one sided, get same thing over and over - or look at the properties of the film (content based) - genre, directors etc - problem is always the same, same movie but you need to push people out of this

- people aren't so one dimensional, so a hybrid approach is best - comibination of multiple thing - scores and weights

how to implement - he rated this movie highly - he also rated this movie highly - correlated on rating - He liked this movie which is in this genre - I liked this movie that is also in the same genre - correlated on genre

how to represent easily in cypher - how natural that works - that is the power of graph databases

however, finite movies - same movie can be discovered multiple times - how to change that - how to rank the relevance - how to rid of data that is not part of the recommendation - encapsulate in another graph

you can see how easy it is to build up a simple thing - cost of development and implementation is low http://brettb.net/project/papers/2007https://www.statista.com/statistics/250934/quarterly-number-of-netflix-streaming-subscribers-worldwide/

Clustering

Scalability and performance