In the world of big data, fierce competition and minimal brand loyalty, companies need to use all the information and resources they have to keep their customers interested and wanting more. Recommendation systems have become a crucial part in offering this. It is one of the most effective ways to show the customer that the company knows them and to stop them wasting time searching for what they 'think' they want, and to offer what company knows they really want. When thinking about implementing a part of a system that is data-oriented, one has to understand the constraints, rate of change, size and what it represents to decide which direction is best to take. If this is applied to the recommendation system 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) all of which are rating movies constantly and therefore it is highly connected. Graph databases is the only system out of NoSQL or relational models that specialises in all of these outcomes. Relational-models are known to be good with data that is well structured and well understood but when you consider the questions presented about the data that we're using, one can see the implications of using it. Facebook discovered that 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).

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 reasons discussed later, this is because NoSQL databases are used at an advantage when dealing with unstructured or semi-structured data. Options like document store databases or similar alternatives offer the ability to store information about an entity in a single 'document'. This benefits speed of reading and writing, 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. The meaning of data comes from relationships between it. In a recommendation system, graph databases give a complete 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, making it easy to understand everything about these aspects from all angles (De Marzi and Neo4j, 2015). Below, I discuss and present the reasons why out of all graph database providers, Neo4j is the best solution.

When dealing with big data, it is easy to be overwhelmed and to not be able to make sense 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 into the actual graph database (Lyon and Neo4j, 2016). This opposed to other NoSQL alternatives that still require the mentality of data to be organised into entities with custom layout within and for it 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 the data. It also helps 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 represents.

This flexibility when 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 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. Cypher is optimised for Neo4j and its functionality contrasts the basic CRUD operations found in other NoSQL databases. Its main focus 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 it shifts the focus to what you actually want from the data, not how to get it. Queries are 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). As the data and their connections grow exponentially, so do the SQL statements, unlike Cypher queries that stay concise which 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. 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. Clustering allows multiple machines with multiple instances to be grouped together. This ensures consistent throughput as multiple machines serve requests in parallel, that the data is replicated throughout the instances so there is nothing lost if one fails and requests can be re-routed if an instance is unavailable to ensure that request is served (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 write in ACID transactions is a feature which is specific to Neo4j (Chao, 2016). This is important to the success of a Netflix recommendation system as when working with high volumes of sensitive data in real-time the data cannot be lost. 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 would be possible and therefore a different database system would have been argued for the recommendation system.

Content based recommending 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 it is a candidate for recommendation (Recommender-Systems, 2012b). 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 to return movie nodes that have the same 'genre' descriptor (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 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). 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, which would provide a

factual approach to describing the item without bias.

The collaborative filtering approach is were the system builds profiles of users based on their behaviours and actions, it then compares the users to form segments of similar users of which recommendations can be made based on actions made by other users of the same segments (Recommender-Systems, 2012a). Regularly on Netflix, users are asked to give ratings to the movies and in doing so it improves the correlation between segments. The system can assume that if one user of a segment rated a certain movie highly, then that movie can become a candidate for recommendation to other members of the segment Thus, the nodes of the graphs would include the users, which could be grouped together, and the relationships could be 'RATED', 'LIKED', 'WATCHED' to all the 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 Neo4j 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 it is the only solution.

Although this can help to create a wider picture of the user and their preferences in relation to other users. It can again fall into the trap 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. However, 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 users in a group have to be processed to find the informatin 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.

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 help 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 whole system as the complete architecture incorporates many other aspects. These, 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 schema. 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 would be modelled in conjunction with decision of the recommender system described above. 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 would be a part of it, as they can encapsulate it to work with 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 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 differences of relationships within segment of users, as described in the collaborative filtering recommender (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 helped by the success of the recommendation being logged. Blacklisting and filters two technique are used to refine the results retrieved further. Blacklisting is common technique 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 techniques could remove all results that are returned by users in the same group located in different countries, as Netflix only provides certain 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 log 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 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).

The Netflix recommendation system is only possible when working with connected data as it has to be drawn from multiple sources, connected and calculated to present a relevant recommendation. Most NoSQL systems can handle the growth and shrinkage of data easily, they can read and write their data in great speeds, they can also provide valuable data distribution and protection. But none can do all of these and solve the issue of connected data like graph databases can. When you look at the companies that use graph databases as the core of their products, you start to see some of the most successful companies in the world right now. Walmart, PayPal, Google all share common themes of their data being real-time, connected, ever growing, and Netflix is no exception (Lyon and Neo4j, 2016). As Neo4j is the market leader in graph databases and can handle the entire implementation of a recommendation system, it is the obvious solution (Agricola, 2012).