Survey on Unicorn: A Powerful In-Memory Database with Inverted Index for Social Graphs Using Type-Ahead Method

Rubab Zahra Sarfraz[1] **and Qurat-ul-Ain Zafar**[2]

[1]CSE department, UET, Pakistan
[2]CSE department, UET, Pakistan

Abstract

Facebook has evolved to become one of the world's biggest social media network and to keep up with the its extraordinary pace of growth, it has had to make several changes to its search techniques. Unicorn is an inverted index database that Facebook currently uses. In a simplified version, the Social Graph can be represented in terms of nodes (objects) and edges (relations between objects). Unicorn not only allows users to search for the objects explicitly related to them but it also harnesses the full power of social graph and produces search results from across the nodes and edges such as searching for people nearby and suggestions for friends based on common friends or common friends of friends. Each node is indexed and thus Unicorn can use a dataset comprising of hundreds of millions of different points to search across multiple nodes at strikingly fast speeds. Unicorn is built using the search principles and techniques employed in the Type-Ahead method but has many advanced features that a typical Type-Ahead Search based system lacks.

Introduction

The growth of Social Networks such as Facebook and Twitter has given a whole new dimension to graph search and subsequently given birth to search indexing methods that are not only efficient but are also specifically designed to support the graph structure. Traditional Relational Model has failed to encompass the large amounts of data that make up today's Social Networks, primarily because a typical relational database requires a set structure. A traditional Relational database is based on 4 main principles or the more commonly used acronym, ACID that is consistency, atomicity, isolation, and durability. NoSQL Models reject the concept of ACID. In general data such that it consists of tables with a number of columns but few rows or has attribute tables, data that has a large number of many to many relationships with characteristics that govern tree like structures or data that requires constant changes in schema is better suited for the NoSQL model[1].

Graph Search, therefore, has grown huge and current research is focused on devising Graph Models for selecting the Top items in a user's news feed as well as methods that can refine searches by taking them to the next level. Facebook is perhaps one of the best examples of giant, NoSQL databases. The collection of data that the site maintains is overwhelmingly large and it is integral that the accompanying search method can not only deal with such large amounts of data but also extract maximum meaningful information from it. Graph databases as a general rule, can yield different aspects of information, even those which are not obvious at first sight. For example, research is being done to evolve methods which can search current graph databases to find hidden patterns and use these patterns to prevent crime[2]. Similarly, Facebook too realized that with its previous search method, Type-Ahead it wasn't utilizing the information fully and that the user searches could be made more socially meaningful. Unicorn is specifically build with the intent to connect dots and search for explicit as well as implicit connections in the Social Graph.

Facebook took the initiative to build a in-memory indexing system that has been designed from day one to handle graph data. Unicorn in all its complexity is thus a great indexing system, one that is at par with Google and Microsoft's search facilities. The paper will proceed to discuss the various aspects of Unicorn. These will include the reasons behind its conception, the method of operation and future developments. Prior to Unicorn, Facebook has used two main search indexing methods, namely Type-Ahead and Keyword Based Search. In this Survey Paper, an attempt will be made to describe how Unicorn works, its different features and how it compares with its predecessor, the Type-Ahead method. In the end, a feature comparison of both methods will help develop better understanding of how Unicorn's complexities and abilities derive from the many features of the humble Type-Ahead.

Related Work

We looked into the principle difference between relational databases and graph databases[3] so that the need for a complex graph searching system like Unicorn can become more clear. Relational databases have been around for almost 30 years now and it is

important to understand why a structure that has been stable for three decades can no longer support the kind of data that is needed to operate Social Media sites today.

Another paper[4] which makes for an interesting read, looks into detailed aspects about the query languages and their various components that can constitute graph querying methods. It also discusses graph algebra and relational algebra which can be the basis of building a more generic, more widely applicable version of Unicorn. Srihari[2] takes the concept of IR (Information Retrieval) to the next level by developing the framework for Concept Chain Graph which can help in UIR (Unapparent Information Retrieval). Facebook Unicorn is currently heavily reliant on the user to provide the necessary information, the ability to anticipate and thereby predefine specific scenarios that can be of interest to the user as well as the assumption of 'tagged' or 'named entities'. This paper goes beyond the scope of Unicorn's current abilities, detailing a framework that can help establish an evidence trail based on multiple links. Concept Chain Graphs could help Facebook analyze user data more throughly and could unveil underlying patterns that could be of significant importance to understanding and improving social networks as well as devising new, effective marketing techniques.

Result Aggregation is a main part of Search Indexing Methods like Unicorn. Currently Unicorn employs the traditional operations such as Join to aggregate the results.[5] looks at alternative methods of aggregating the results of joining on the top k results, making retrieval faster and efficient. The two alternate methods discussed in the paper are Graphity and STOU. As Facebook grows larger, the cost of aggregating search results is likely to grow.

Finally, Facebook's official paper on Deepface,[6] talks in detail about one of organization's major achievement: an intricate system of face detection that leads to almost 97% accuracy in LFW data set. The system is based on 120 different parameters which give it an amazing capability to detect faces accurately. Given such individual systems and techniques are efficient, they can be combines with Unicorn to achieve some remarkable results. Imagine being able to search for a person in hundreds of photos based on their profile image.

System Overview

The Concept of The Social Graph

Before attempting to understand Unicorn, it is vital to grasp the concept of The Social Graph. Facebook defines The Social Graph as a database that catalogs the inter-relationships between people and real world objects such that each 'node' is a person or an object and each 'edge' is the relationship between the two nodes[7]. The Social Graph serves as the main tool that gives context to the data. It is therefore important to equip the modern indexing systems with enriched information and advanced analyses methods so that they can produce meaningful output. This calls for a high-performance tools which are generic in nature as well as the ability to integrate information from several data sources[8].

Understanding Unicorn

Unicorn has been designed to traverse The Social Graph. Facebook is a social network that supports billions of users from all over the world and the network keeps growing. Each user has a friend list, a list of statuses and posts by the users, a list of posts and pages liked by the user, photos and videos and associated profile information among other things. Apart from users, there are other entities that Facebook allows: a Page is an easy example where businesses or groups can interact socially. There are numerous entities that Facebook supports and each comes with a largely unstructured, massive information bank. All this information is organized in the form of The Social Graph described above that is in the form of nodes and edges. Unicorn uses the same underlying concept to manipulate data and produce meaningful results.

One of the many analytical methods that Unicorn is equipped is the use of edge definitions. There are, quite literally, thousands of edge types [1] which are both symmetrical and directional. A simple friend relationship between two people will qualify as a symmetrical edge. A typical person will have roughly one thousand edges connecting it to other people and objects[7]. which we will henceforth refer to as 'nodes'. Facebook itself is made up of billions of nodes. Using the node and edge structure, Unicorn can model real-world structures beautifully. Suppose that there is a node A which represents the concept of male. Unicorn can then link all males to this concept using a simple 'gender' edge type. Now lets say that we want to see one node's male friends: we can find the results simply by taking the intersection of the result sets returned by node's friends and A's set.

Unicorn is an inverted index framework that has the capabilities of building indices and retrieving data from the index. A Unicorn vertical can contain a very large-sized index that cannot be stored in a single machine. For this purpose these indices are broken down into many shards such that each shard can fit into one such machine. These machine are called Index Servers. When the retrieval query is entered in the search box, it is sent to a Vertical Aggregator which further broadcasts the queries to Index Servers. Each Index Server gets the desired results from its shards and pass them to the Vertical Aggregator which in turn combines all these entities and return them to the user[9].

To understand the kind of data that Unicorn is responsible for indexing, some figures are worth mentioning. Every week Facebook adds a huge batch of relevant information to Unicorn which is processed and indexed throughout the week. The

batch typically consists of billions of data points and relationships which is to be expected given Facebook is the hub of activity. Every day, more than 2.7 billion likes and roughly 2.5 billion pieces of new content which includes user posts are added to the network[10].

One of the most important feature or a performance metric of a successful search engine is how it ranks the search results as the relevancy of search depends on it. Unicorn's functionality was extended by adding search ranking capabilities. Each entity type is maintained in a separate Unicorn Vertical because every entity type has different requirements for ranking. A Top Aggregator is created above all these Verticals to coordinate activities across them. It contains a query rewriter that translates the user's retrieval query into a structured Unicorn retrieval query and a set scorer that assigns a numeric score to an entity on the basis of entity data and query. Scoring requires some additional information which resulted in the storage of a blob of metadata related to each entity in index.

These results are passed through yet another filter called Result Set Scorer that returns a subset of most important entities not necessarily highest scored. As the results are received from a number of different Verticals, they are blended together in the Top Aggregator . Unicorn can support multiple searches which means the results need to be "joined" together. The 'apply' operator used by Unicorn natively for nested queries was extended with a framework that made the data sharing between scoring components possible. Graph Search interacts with the user in two phases: 1) Query suggesting phase 2) Search phase. Both of these are explained below:

Query Suggesting Phase

When a user enters a query into the box, it is parsed on the basis of grammar by a Natural Language Processing module (NLP). The NLP module provides the Unicorn with hints based on the type of entity. For example, if the user types "people who live in Park", the module will send the query "Park" to Unicorn and suggest a bias towards cities and places. The results like "Park Lane" and "Park Hotel" are returned instead of "Parker". Basically, the retrieval query gets parsed by NLP module which identifies its portions for searching in Unicorn. The Top Aggregator fans out the request for different Verticals. It also rewrites the query which is then sent to the Vertical Aggregator which broadcasts it on each Index Server. Index Servers retrieves the required entities from Index. The top results get returned after scoring. These results are combined in the Vertical Aggregator and returned to the Top Aggregator which performs blending. The combined result is sent to NLP module which constructs parse trees and assigns score to each one of them, the top of which are showed to the user as suggestions.

Search Phase

Search phase starts when the user selects one of the results shown by the NLP module. Top Aggregator makes a query plan after receiving the parse tree selected by the user. It rewrites the retrieval query and sends to Vertical Aggregator of user vertical. Vertical Aggregator sends it to Index Servers which receives entities from Index. Score is set and top of the entities are sent back to Vertical Aggregator which combines the entities returned by each Index Server and returns them to Top Aggregator for performing result set scoring. The results are then prepared for a specific vertical -for example places vertical, if the user has entered some kind of place to search for. The steps from sending query to Vertical Aggregator to the results returned by it to Top Aggregator are repeated. Top Aggregator passes the result through result set scorer and the final set of results is displayed to the user.

Unicorn Query Language

Unicorn Query Language provides the necessary tools that allow users to structure more meaningful queries. Apart from the traditional AND and OR operations on data, Unicorn Query Language also supports an operation called Difference. The operation supported allows it to filter results from the first operand using the second operand. To put it in simpler words, when the Difference operator is called, it produces the results of the first and second operand and returns (result of first operand – result of second operands). For example you could ask it to return a list of your friends who are not a friend of some node A. Table below shows which operators are provided to the searcher, the rewritten query by Top Aggregator and description of what that query means.

Restrictions Imposed by Lineage and Privacy

Unicorn is designed to use the full power of The Social Graph we talked about at the beginning of this paper. However, it is also restricted by Lineage and Privacy. Facebook allows users to write posts or upload photos which are visible only to a certain set of people such as a person's friends. In this case, these edges can be used to refine results for A and A's friends but outside the set of A's friends, Unicorn should not use these private graph edges to filter and score the results.

Facebook's current methodology is to give Unicorn a free hand in computing results. Along with the results, Unicorn is also designed to specify how it generated the results. The caller, usually a PHP front application can then filter the results based on privacy restrictions showing the end user a set of results that conform to the privacy rules set by the user. This design however imposes a penalty of the system which can bring down the efficiency slightly.

Facebook's Previous Strategy Type-Ahead

In the realm of Software Engineering, one of the primary questions asked in elicitation of requirements is what would happen if the new software is not built. Prior to Unicorn, Facebook was using Type-Ahead, a search method which is still very popular and widely used. In order to understand the importance of Unicorn, it is essential to examine Type-Ahead in some detail.

Type-Ahead engine started generating results and suggestions as soon as the user typed the first letter. With the addition of further letters and completion of entire word strings, the search results would change dynamically in order to best fit the query. It was simple, fast and an acceptable method of searching. Therefore it doesn't come as a surprise that by the end of year 2012, Facebook's Type-Ahead was handling over 1 Billion queries per day![7].

Type-Ahead returns a ranked search. Type-Ahead index servers typically consist of posting lists which are made up of user ids whose first or last name matches the alphabet given in the search. When a search begins, the query is merely mapped to these posting lists and the resulting ids are returned to the querying user. The mechanism thus allows user to progressively filter the results as (s)he types in the query and then returns the result that match best. These matches are based on the edit distance that is the number of delete, insert and substitute operations that are required to make the two words (the queried string and the string in the database) equal to each other. With each key press, Type-Ahead thus computes results from the server and displays them. It has to be extremely fast to enable this kind of interactive search and when large amounts of data are involved, this search time can grow large making Type-Ahead inefficient[11].

Problems with Type-Ahead that are Solved with Unicorn

The interactive search provided by Type-Ahead significantly reduces typing times and takes care of minor errors in keyword typing by using the statistics of the XML database to identify and process results according to user intentions[12]. However, it still lacks in one major area social relevance. Type-Ahead search would be a great idea is a website like IMDB where typing a string would start producing the movie names but in a network like Facebook, finding a certain person A is more than a matter of simply matching the ids. There could be thousands of people that map to the query but at the end of the day, the user would only be interested in those ids which are socially related to him , for example he could be looking for an A in his friend list or friends of friends list. Such scenarios are very common and in this case, producing a list of all user ids that maps to A would be insufficient as it would still require a lot of user filtering, dramatically reducing the efficiency of search.

One could address the problem by adding more conditions like an AND condition: filter results according to the string input AND make sure that the results are included in my friend list. This will work but given the phenomenon of social relevance, its reach will be largely limited. This problem is solved by making sure that some fraction of the result adheres to conditions of social relevance and the end result produced consists of both the socially relevant fraction and the search result based on query so that user can get easily find what (s)he is looking for. This approach thus ensures diversity of the result using operators such as Weak And and Strong Or.

Unicorn acquires a query via Thrift. Based on the parameters that have been provided in the Thrift query, it establishes a scoring function (The Query Suggesting Phase). Finally, in the aggregation the results with a higher score are given priority. However, even results that do not conform to the scoring function but relate to the query will be displayed. In order to do this, Facebook maintains special data structures that are heap based. Thus even the results which have a lower score can go on to form a part of the final result and thus increase the diversity of the result. With Unicorn, Type-Ahead gets a huge lift and the results suddenly become much more relevant and refined.

Unicorn thus harnesses the power of Type-Ahead and builds upon it to create an efficient indexing system. Below is a tabular comparison of the features of Type-Ahead and Unicorn. As the table shows, Unicorn adds inverted index framework facility to Type-Ahead as well as many other features that make it a superior system.

Unicorn's Performance

Type-Ahead is highly efficient in latency rate as it loads all information associated with the user in cache of the browser as soon as s(he) focuses on the search bar. As Unicorn has to search through relations in addition to searching nodes, it is necessary for it to truncate results especially in the case of nested queries, if it wants to achieve the response time that users can get with a Type-Ahead method. With the improved ranking algorithms, it can satisfy its user. An average user is satisfied with the performance of some search engine when the results are relevant.

Since, there could be millions of results of the inner query, these have to be truncated in order to avoid performance overheads.

Conclusion and Future Work

Facebook introduced its Graph Search with Unicorn at its heart last year. As one of the biggest social networks in the world, Facebook has the Social Graph at its disposal. If Facebook developers are able to integrate the extensive qualities of search engines like Google and Yahoo with their Social Graph, the organization stands to gain huge benefits in terms of digital

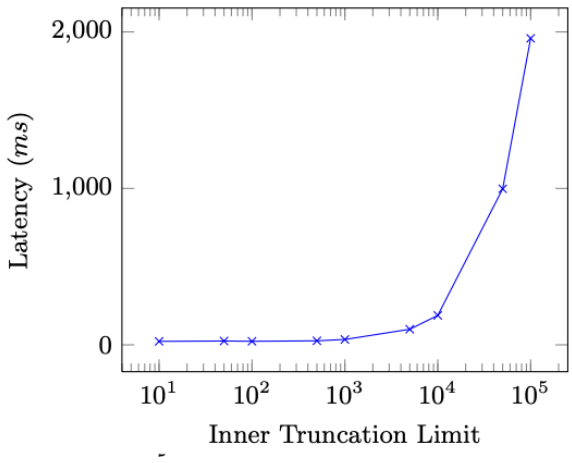

Figure 1. The graph shows that as the inner truncation limit increases the performance of this query execution gets affected[7].

advertisement. There is thus a lot of room for improvement in Unicorn's features which in turn can help it realize its true potential.

Relevancy is an important feature of a search engine. Since Unicorn is search engine for Social Graph, the definition of relevancy may vary from user to user. For some users, relevancy may mean

Unicorn uses Natural Language Module. This imposes a limit on users as well as on the functionality of the system. By adding more features to NLM for international users and by taking into account the type of queries which are being fed to unicorn that it does not fully comprehend, it can improve itself immensely.

Freshness means the recentness of results provided by the search engine. For example, if a searcher writes a query "Movies watched by my friends" the recent results would be useful for the searcher if s(he) wants to know about the recent movies. Or this set of results can also be obtained if "recently" keyword is added to the system or another filter of "year" or "month" is added, the Top Aggregator now plays the role in ranking the results sent back by Vertical Aggregators according to Diversity is the ability to provide the searcher with high quality result in case of ambiguous queries[13]. For example if a person writes "me ambiguous" in the Graph Search, Unicorn would provide the user with posts of the people that contains the word "ambiguous" or "ambiguity". There should be a refined mechanism for handling these types of queries. For instance, it could ask the user what does it mean by "ambiguous"; a person, a place or a page etc. This will help user to identify what he wants to search for.

For Unicorn to be successful a methodology should be adopted for active user response/comments on performance of this search engine. Facebook should devise a method to record the input directly from the user because it is based on people and what they do. Unicorn is dependent on people for its performance. Over the time, feedback can help in minimizing the issues like relevancy and diversity. People have different choices, tastes, preferences in this world but there's always a subset of people who have same choices, same taste and same preferences. Maybe in the coming years, Unicorn is able to handle a particular subset of people differently than the other subset of people having common things. Maybe it is able to apply different ranking and suggestion algorithms for different subsets.

References

1. Vicknair, C. *et al.* A comparison of a graph database and a relational database: a data provenance perspective. In *Proceedings of the 48th annual Southeast regional conference*, 1–6 (2010).
2. Srihari, R. K., Lamkhede, S., Bhasin, A. & Dai, W. Contextual information retrieval using concept chain graphs. *ceur-ws. org* (2005).
3. Raymond, J. W. & Willett, P. Effectiveness of graph-based and fingerprint-based similarity measures for virtual screening of 2d chemical structure databases. *J. computer-aided molecular design* **16**, 59–71 (2002).
4. He, H. & Singh, A. K. Graphs-at-a-time: query language and access methods for graph databases. In *Proceedings of the 2008 ACM SIGMOD international conference on Management of data*, 405–418 (2008).
5. Pickhardt, R., Gottron, T., Scherp, A., Staab, S. & Kunze, J. Efficient graph models for retrieving top-k news feeds from ego networks. In *2012 International Conference on Privacy, Security, Risk and Trust and 2012 International Confernece on Social Computing*, 123–133 (IEEE, 2012).

6. Taigman, Y., Yang, M., Ranzato, M. & Wolf, L. Deepface: Closing the gap to human-level performance in face verification. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, 1701–1708 (2014).

7. Curtiss, M. *et al.* Unicorn: A system for searching the social graph. *Proc. VLDB Endow.* **6**, 1150–1161 (2013).

8. Martínez-Bazan, N. *et al.* Dex: high-performance exploration on large graphs for information retrieval. In *Proceedings of the sixteenth ACM conference on Conference on information and knowledge management*, 573–582 (2007).

9. Indexing and ranking in graph search.

10. Facebook rides unicorn to graph search nirvana (2013).

11. Nivedita, K., Naveen, R. & Sravani, K. Fuzzy type-ahead search in xml data. .

12. Rathod, S. & Kothari, D. S. Survey on interactive keyword search over xml data to obtain top-k results. *Indian J. Appl. Res.* **4**, 158–160, DOI: 10.15373/2249555X/MAR2014/46 (2011).

13. Comparing search engine performance: How does cuil stack up to google, yahoo!, live ask.