Alternate *ACM* SIG Proceedings Paper in LaTeX Format

Johannes Spießberger Ralph Hoch Carola Gabriel

ABSTRACT

Targeted advertising is one of the key revenue sources for internet services. While traditional approaches tried to suggest ads to users based on statistics derived from historical data, modern approaches try to make use of big data. This paper tries to give a short overview of current scientific trends in NoSQL and big data management. As graph databases are especially fitting for modelling social interactions, this paper puts an emphasis on this type of NoSql.

General Terms

Big Data, targeted advertising, NoSQL, graph database

Keywords

Big Data, targeted advertising, NoSQL, graph database

1. INTRODUCTION

The recent years have shown a massive growth in data used for analysis in a broad range of fields. Big data as a concept for handling this amount of information has become a huge field for scientific research and business models alike. One of the cornerstones of the technical implementation of such systems is the usage of NoSQL databases. While these are available in many different flavours, graph based approaches are the one that differ the most from traditional database systems. Modelling social interactions as graphs and extracting various information is, among other applications, one of todays key techniques for targeted advertising. TODO cite http://ieeexplore.ieee.org/stamp/stamp.jsp?tp=arnumber=1579567 While the usage of such graph databases makes for a natural abstraction of some real life observations, the technical implementations of the graph database itself becomes more challenging than those of traditional RDBMs system. The following sections will summarize some recent research concerning the inner workings of graph databases.

2. GPU BASED FREQUENT GRAPH MINING

With the availability of CUDA and opencl GPUs have become a source of cheap computation power for significantlly less money than general purpose CPUs. While not applicable to all types of computational loads, there are various fields which benefit immensly from a high parallelization degree TODO source. Especially loads without the need of communication between the threads are well suited for GPU architectures TODO source. The following sections will describe some applications in graph databases.

Source: http://jmlr.org/proceedings/papers/v36/kessl14.pdf Frequent graph mining describes the search of reoccuring sub patterns in a set of graphs. Among various applications this technique can be used to find similar communities within social networks TODO source. Figure 1 shows a pattern P that occurs in $G1$ and $G2$

As this problem has been shown to be NP-complete TODO source it is especially important to find scalable algorithms to deal with this problem. The GPU-based graph mining algorithm combines concepts from gSpan TODO source and DMTL TODO source. For the effecient pruning of duplicate subgraphs the DFS-code from gSpan is used. Parts of DMTL are used to store all isomorphisms for each pattern, which allows a fast computation for how many graphs in a graph database contain a certain subgraph. In order to achieve good performance it is necessary to change the data structure of the graph in main memory. Hashmaps or linked lists are not well suited for GPUs as data is not cached and therefore access locality must be considered for performance reasons. The memory layout has to happen in a way that the id and neighborhood can be quickly looked up. To achieve this goal the combined neighborhoods of all vertices, edges and vertices are stored in seperate arrays. A fourth array is introduced which holds offsets to speedup the lookup between arrays. This memory layout including the implications of how data is looked up by gSpan and DMTL allow for a very effecient parallelization on GPUs. For various benchmarks this technique led to significant speedups. Figure 2 shows the performance comparison between a GPU and a sequential implementation for frequent graph mining on multiple datasets. The principles of this research can also be extended to additional graph algorithms like closed graph, maximal graph and temporal graph mining.

3. DIFFERENTIAL QUERIES

http://ceur-ws.org/Vol-1133/paper-34.pdf http://download.springe
Abandoning the rigid predefined schema of traditional

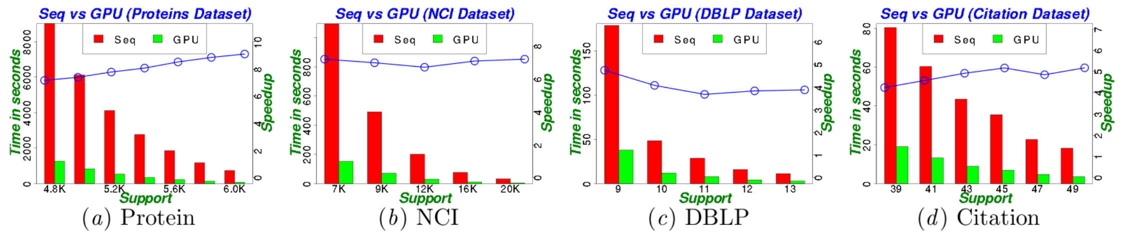


Figure 2: GPU Benchmarks

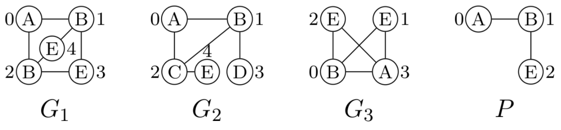


Figure 1: Graphs and pattern P

database systems is one of the key features of NoSQL. While offering more flexibility in terms of development, users have a hard time gaining deep knowledge of the data and its structure. This mostly manifests in queries giving unexpected or empty result sets. The research provided by TODO cite gives an idea of how mitigate this problem. Diff-queries provide users with the information about which part of a query actually matches data in the graph and which parts lead to an empty result set. A query in a graph database can be understood as a pattern that has to match parts of the graph. These pattern can be modeled as graphs themselfes. In order to be able to determine which parts of a query provide a match and which parts will finally lead to an empty result, a common connected subgraph algorithm is applied to the query and the graph data intself. The resulting common subgraph can then further be used to partition constraints in a query into matching and non-matching conditions.

Figure 3 visualizes the basic idea of differential queries. Assume a query that searches for two soccer players that play for the same club and are in the same national team. If the graph database has a player that matches parts of the query conditions, namely playing for a national team and a club, but there is no second player with the same attributes, the resulting differnece graph will contain the non matching conditions.

Due to the nature of these operations are large number of intermediate results may lead to performance problems. This problem may be mitigated by introducing Top-K differential queries. While modelling the query of a graph the user is given the option to weight certain edges, vertices or subgraphs according to his preferences. By using a new algorithm called relevance flooding, the specified weights are attached to the data in the graph database. When calculating the maximum common subgraph, not the number matching edges and vertices is maximized, but the score resulting from the weights given by the user.

4. SLQ

Another approach of easing the writing of graph database queries for non professional users is the SLQ language. A property graph model is used for formulating queries. Aside

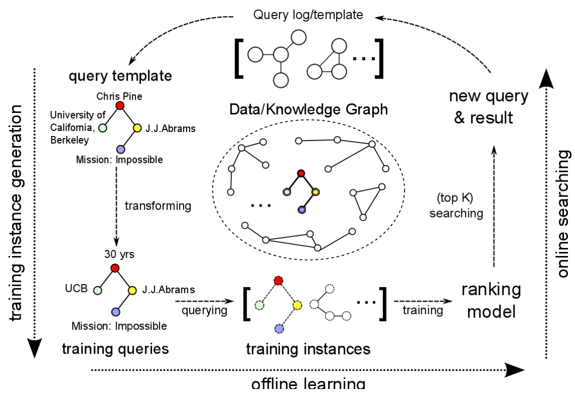


Figure 4: SLQ: graph querying framework

from having nodes and edges there are key value pairs associated with each node. These key value pairs will be further treated as keywords. Queries are then further transformed using a library L consisting of transformations functions. Those may include simple string replacements, semantic transformations, numerical and topological transformations. These functions allow for the compensation of spelling errors or the usage of synonyms.

In an SLQ query a match in the data is derived from the mapping of a query, including possible transformation, to the data itself. It is therefore necessary to define a weight for such transformations to only show the most relevant results. As equal or predefined weighting for all transformations may not lead to an optimal query outcome, SLQ is able to learn and further improve the relevance of its transformations.

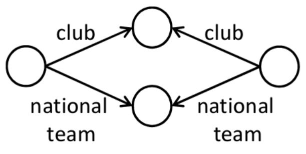
The first possibility is to use existing query logs of real users during an offline learning stage. As a high quality pair of queries and their results may not always be available an online learning approach is added. During the online operation, users can specify a number k which will then be used to display the most relevant results after applying a set of transformation. Be receiving feedback from the user SLQ is able to further improve the rankings of the transformations.

Figure 4 shows the basics of this learning process.

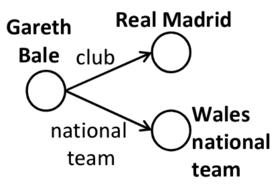
Figure 5 gives an example for three possible queries and their final results. The second query tries to find a person serving in the union army and was in a battle. By introducing the node Missouri the user specifies that it may be somehow related to the first conditions.

5. PARALLEL ADJACENCY LISTS

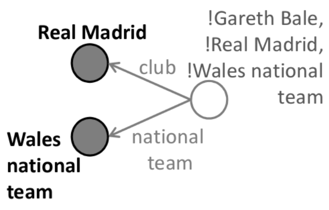
http://arxiv.org/pdf/1403.0701.pdf Modelling social net-



(a) Diff-query



(b) Maximum common connected subgraph



(c) Missing part of an answer

Figure 3: Differential graph for data and query

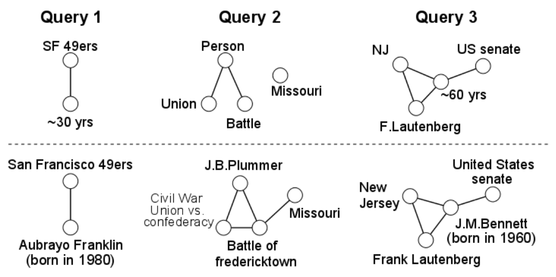


Figure 5: SLQ query examples

works like Twitter has some particular challenges for graph DB which come from the uneven distribution of edges between nodes. Such graphs have the tendency to have a small number of nodes with a high number of edges and a high number of nodes with a small number of edges. TODO zitat aus Paper 11 This makes partitioning a graph to allow for localized acces hard. Partitioned Adjacecny Lists PAL tries to reduce the number of random accesses while minimizing the storage space required. Edges are stored as pairs of vertices and are partitioned by a range of destination vertices and are ordered by the source ID. These partitiones must not be of equal length but should be chosen such that a partition can fit into memory. This theoretically limits the number of incoming edges a vertex may have. As the graph connectivity is fully represented by the edge partitions, vertex data can be stored seperately and is partitioned based on the edge partitions. Retrieving verted data can be achieved by simly calculating an offset from the edge partitions. Edge partitions are immutable data structures and do not allow direct insertion of edges. To counter this problem new edges a stored in a buffer and inserted if their number exceeds a certain threshold. During search operations these buffers are also considered. Todo zahlen und benchmark bilder

6. DISTRIBUTED GRAPH DATABASE

Current graph databases are mainly inteded to run on a single instance and therefore face performance and availability problems. TODO Zitat aus paper Acadia is a distributed graph database intented to run in hybrid cloud settings. By utilizing graph partitioning and a Master-Worker pattern it is well fit to operate in an environment with less than 100 workers. Each distributed process has an associated local data store with an assigned storage quota. When adding a new graph the partitioning happens via Hadoop and Metis and the subgraphs are then inserted into local Neo4j instances.

7. CONCLUSIONS

TODO