Aurum: A Data Discovery System

Susmitha Marripalapu (A20489531)
Pranava Brunda Manuguri (A20472679)
Yogith Reddy Venkanagari (A20493317)

Group-7

1 INTRODUCTION

Organizations need more data to uncovere the insights and trends to make strategic choices for a company. Because data is stored in multiple tables which have various relationships, companies face a data discovery problem where their analysts spend more time looking for relevant data than analyzing it.With addition of new data and the scale at which the data generation process is increasing these days, we should have a reliable solution which will not require much changes with changing data, to be able to work with structured or unstructured data, and ability to provide querying flexibility to support any type of data needs.

To solve this problem, a system AURUM is developed and used to build, maintain and query the EKG (Enterprise Knowledge Graph) which captures relationships between datasets.

2 AURUM ARCHITECTURE

2.1 Major Components

2.1.1 EKJ BUILDER

An enterprise knowledge graph is a representation of an organization's artifacts and knowledge domains. The nodes in the graph represent the columns and the edges represent the relationship between the columns. The main model for responding to user queries is created by model builder. This model is built by reading the data summaries created by the profiler and storing the output model which is then used for pickle serialization.

2.1.2 PROFILER

The ddprofiler is in charge of reading the data from wherever it lives (e.g., CSV files, tables, the cloud or an on-premise lake) and creating a set of summaries that represent the data in a way that allows us to discover it later. All the data summaries are stored in a store, which at the moment is elastic search.The underlying data is represented by signatures, which are stored in profiles by a profiler. The data is read only once to create these profiles.

2.1.3 SRQL

To allow users to describe complicated discovery questions based on set of discovery primitives, a source retrieval query language is provided. To build the SRQL, RDMS-based execution engine is used. To speed up the processing of discovery path queries, we use a graph index.

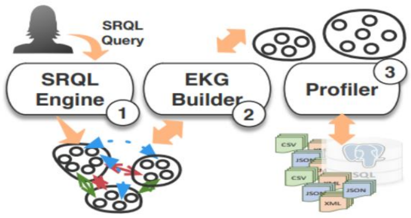

Fig 1 Architecture

3 Enterprise Knowledge Graph - EKG

An enterprise knowledge graph is a representation of an organization's artifacts and knowledge domains. It is a collection of references to your organization's knowledge assets, content, and data that leverages a data model to describe the people, places, and things and how they are related.
Using the naive process of reading each column and comparing it with all other columns of other tables to find similarities will take multiple access rounds to data, which result in heavy CPU and input operations.

3.1 BUILDING

The building of EKG consists of two stages, source and sink. Each of these stages compute a part of the profile for each column. The source is used to read data from files, RDBMS, etc., and provides the input to the compute_profile() function. The sink stores the computed profiles, so that they are accessible to the graph builder during the second stage of the building process.

3.1.1 Profiler

It is used to Read data and produce signatures. Sketching technique is used to summarize data in one pass. Sketching simplifies the computational task by generating a compressed version of the original dataset that then serves as a surrogate for calculations. The compressed dataset is referred to as a sketch, because it acts as a summarized representation of the full dataset.

- **Achieving Efficiency:** To achieve parallelism, multiple threads are assigned to one pipeline or stage. Task grain technique is used for the full utilization of available hardware.
  Partition data at column level. The threads which are involved in reading the data are separated from the processing stage using a queue.
  Task creator partitions each column into subtasks which compute a partial profile for that partition. This partial profile is then sent to a merger component which is responsible for creating a final profile. This design is powerful and resists distorted data and utilizes the hardware which is available for processing.

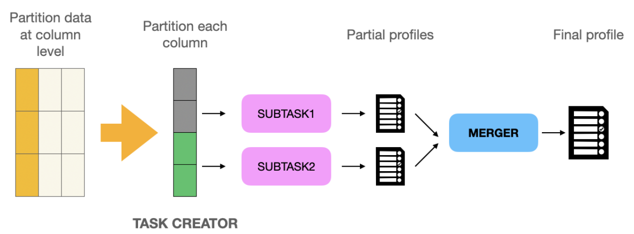

Fig 2 Profile Creation

3.1.2 Graph Builder

It is used to compute relationships b/w columns using signatures from profiler. This will avoid reading data again. This is done in linear time. The idea is that the relationship between the profiles should give us a relationship between the underlying data.
Locality Sensitive Hashing is used to transform the problem of pair by pair comparision of columns to nearest neighbour problem in which profiles similarity is determined using hash function. This will reduce the complexity to O(n). LSH is an algorithmic technique that hashes similar input items into the same "buckets" with high probability. This means, if signatures hash into the same bucket after LSH, that means they are similar.
The relationship strength can be measured in two ways:

- **Jaccard similarity** for which a minhash function is used to convert tokenized text into a set of hash integers, then selects the minimum value.

- **Cosine similarity**, with a term frequency – inverse document frequency signature is a numerical statistic that is intended to reflect how important a word is to a document in a collection or a resource having text.

At this point each relationship has an associated score which tells how similar two columns are and how likely the columns are primary or foreign keys, etc. This score or relationship strength allows users to redefine queries and setup thresholds. The edge weight in the EKG represents this score. To compute the value of the relationship strength or score using LSH, the same signature is indexed multiple times, obtaining multiple LSH indexes with different similarity limit and balance the probability of false negatives and positives.
Primary Key/Foreign Key is known by computing uniqueness_ratio of a column, which is ratio of number of unique values divided by total values. For a primary key column this is going to be a perfect 1. But because profiler uses sketching method, this might be prone to small errors and hence we take values near to 1. After retrieving content similar candidates, we iterate over them to check if they belong to PK/FK and add them to EKG.

3.2 MAINTAINING

When data changes, we want to keep the EKG up to date without having to recompute everything from scratch, and we want to keep access to the underlying data sources to a minimum to reduce input operations.

- **Incremental profiler maintenance**: Using naive approach, computing MinHash signature for column c at time is mt. At time t+1, we can compute m(t+1). If m(t) != m(t+1) i.e. Jacarrd similarity is not 1 which implies that data has changed. Then compute the magnitude of change by 1- JS(m(t), m(t+1)) between signatures. If this difference is larger than the threshold, then change the signature and update EKG. This process needs to compute m(t+1) which needs to read the whole column again, which is a challenge. Hence we have a RESS method.

- **RESS Algorithm (Resource-Efficient Signature Sampling)**: By reading only a small portion of the data, our Resource Efficient Signature Sampling (RESS) determines what has changed. The EKG is then updated after AURUM re-indexes these sources, calculates their new associations, and re-indexes them. We compute the measure of change for 'C' column only using a sample 'S' instead of the whole data.

– Assume that Jacarrd similarity is true, i.e. m(t) = m(t+1)

– Rejecting this assumption, Jacarrd similarity of 2 columns = intersection of columns / union of columns Max JS can be obtained when numerator is minimum and denominator is maximum.

We compute the JS for sample and assume that it is not changed and is similar to JS of C which is already computed by the profiler. If in comparison, the difference is greater than maximum threshold γ, then we trigger a request to recompute profiler.

- **Other factors for continuous evolution of EKG**: Users are provided with some metadata utilities for giving annotations to existing nodes and relationships in EKG. These are visible to other users . Users with right permissions can later materialize the feedback and modify EKG accordingly.

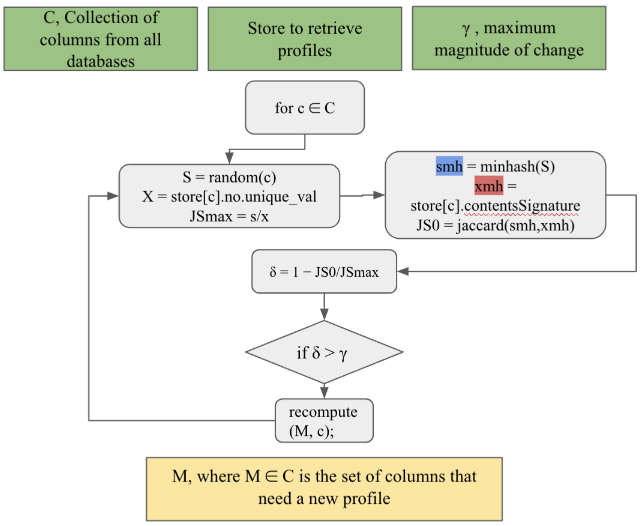

Fig 3 RESS Algorithm

3.3 QUERYING

Querying using SRQL (Source Retrieval Query Language) mainly focuses on two concepts:
1. Discoverable Elements(DE's)
2. Discovery Primitives(DP's) The Discovery primitives are divided into two groups.
1. Pure
2. Lookup
SRQL language uses Discovery Result Set (DRS) to support metadata functionality. Broad Search of Related Source:
Schema Search is used to find all the columns across all databases by using keywords. For example:
keywords such as "Stocks","profits".
Query: results=SchemaSearch("stocks","profits").
To search for similar content, ContentSim command is used. This command also returns their syntactic relationships.

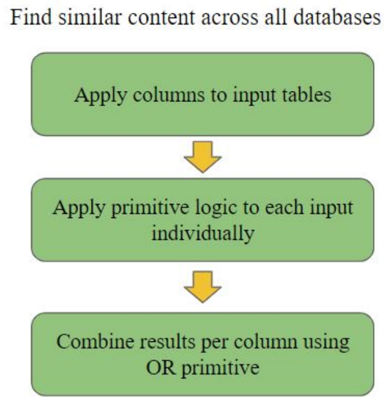

Fig 4 Similar Content Search

3.4 SRQL EXECUTION ENGINE

G-index: To represent the EKG in space efficient manner, we use G-index. This is constructed gradually as the EKG is constructed step by step. This serves as an essential supporting structure to write new strategies if needed. To express the ranking strategies, this directed acyclic graph is abundant.
Ranking Strategies are categorized into two:
The default technique used is known as certainty ranking. In this, the weights obtained from the provenance graph is used and combined. If the juction is reached during the process, the upstream link with highest weights is choosen and the path is continued.
The second technique is more advanced ranking approach for SRQL queries which is coverage ranking. This give the tables as the result where these tables are sorted according to the input discoverable elements.

4 EVALUATING PERFORMANCE

Aurum was deployed with three collaborators with real data discovery problems and surveyed their experience to evaluate it performance. To test it on real use cases, this was deployed in University DWH, Pharma Company and sustainability team of the university. Four users who used AURUM were surveyed to know the usefulness of this discovery system. The different primitives provided by AURUM were useful to express discovery queries for all users. Not all the primitives were used by all of them. It varied according to requirement.

Out of four users, three users gave a 4 and one gave a 3 when asked of likelihood of using AURUM in their organization.

To evaluate performance of building profiler and graph builder, profiler runtimes were computed with index by

varying data amounts. No index profile time computation was done with MinHash signature of 512 and 128 permutations. The results show that all modes scale linearly with the input data.
In the case of NoIndex-512, the limiting factor is computing the signature; when we run NoIndex- 512, it is 4 times more costly to compute than NoIndex-128 due to the reduced number of hashing operations. Therse results are still very positive.
Profile build evaluation was done using finer-grain tasks (Fine-Grain) qhere we created taskes with multiple threads in comparison with creating a task per table (Coarse-Grain). Coarse-Grain did not perform as well as Fine-Grain. The real datasets are distorted and hence this justifies the profiler design.
To evaluate the performance of the graph builder, runtime is measured as a function of the number of input profiles with the content-similarity relationship.
The results show gradual growth of the graph builder performance while using both MinHash and TF-ID. Since graph builder must read data from disk to compute TF-IDF signature and to calculate the IDF across the profiles, its runtime is higher than minhash. MinHash signatures have lower runtime as they are created as a part of profiling process.

The final experiment is to evaluate the efficiency of our resource-efficient signature sampling (RESS) method. The aim is to understand whether data changes are detected effectively and can keep the EKG up-to-date at minimum cost. The results shown in paper justify that by reading only 10% of the original data, RESS identifies 90% of the modified datasets , with a maximum error of less than 15%. For the other sample sizes, RESS gave expected results: the bigger the sample size the lower the error.

5 CONCLUSION

Systems similar to Aurum were developed by Linkedin (WhereWorks) , by Apache foundations (Atlas) and by Google (Goods). None of these systems allow users to change queries on demand.
For more general discovery problems simple keyword search is not sufficient. AURUM is built to tackle this problem. This can be seen as a stepping stone towards addressing the substantial challenges that the modern flood of data presents in large organizations. The experiments so far with AURUM helped to confirm that AURUM is useful for multiple different discovery needs. In our opinion, the ability to put graph technology to work in effective ways will be an adapting skill in the next 5 to 10 years. AURUM has a powerful approach to creating knowledge graphs, which will be a huge benefit to those who decide to adopt graph technology at scale.

References

[1] Raul Castro Fernandez, Ziawasch Abedjan, Famien Koko, Gina Yuan, Sam Madden, Michael Stonebraker, ”Aurum: A Data Discovery System” LINK

[2] Dan Woods, ”Cambridge Semantics,” LINK