INTERNATIONAL JOURNAL OF MERGING TECHNOLOGY AND ADVANCED RESEARCH IN COMPUTING

ISSN: 2320-1363

SIMILARITY EXTRACTION IN USER DATA USING LARGE GRAPH DATABASE

U.Gayathri[1], U.Hindhu[2].C.Janapriya[3], Daphney Joann[4]

[1]Student, Department of CSE, Kingston Engineering College, Vellore, Tamilnadu, India

[2]Student, Department of CSE, Kingston Engineering College, Vellore, Tamilnadu, India.

[3]Student, Department of CSE, Kingston Engineering College, Vellore, Tamilnadu, India

[4]Assistant Professor, Department of CSE, Kingston Engineering College, Vellore, Tamilnadu, India

Abstract

Social networks play an essential role in today's world where the information about several aspects is being shared across users. The data is completely heterogeneous in nature where it varies among multiple users. In this case, the data maintenance and data retrieval requires the effective algorithm to reduce the latency in fetching the information. Recently Semantic web is gaining potential importance in terms of representing the real world data's. In this project, the concept of 'Graph Algorithm' is used to handle the user information where links among the sub links of user information is made. It matches with set similarity (SMS2) query over a large graph database, which retrieves sub graphs that are structurally isomorphic to the query graph, and meanwhile satisfy the condition of vertex pair matching with the (dynamic) weighted set similarity. Based on the index and signatures, we propose an efficient two-phase pruning strategy including set similarity pruning and structure-based pruning, which exploits the unique features of both (dynamic) weighted set similarity and graph topology. A similar social network is modelled as an application to implement the concept and demonstrate.

Introduction

Cloud computing offers the vision of a virtually infinite pool of computing, storage and networking resources where application can be scalable and deployed.

In particular Google cloud service provides Google App Engine for Java! With App Engine, you can build web applications using standard Java technologies and run them on Google's scalable infrastructure.

ISSN: 2320-1363

With the emergence of many real applications such as social networks, Semantic Web, and biological networks, graph databases have been widely used as important tools to model and query complex graph data. Much prior work has extensively studied various types of queries over graphs, in which subgraph matching is a fundamental graph query type. Given a query graph Q and a large graph G, a typical subgraph matching query retrieves those subgraphs in G that exactly match with Q in terms of both graph structure and vertex labels. However, in some real graph applications, each vertex often contains a rich set of tokens or elements representing features of the vertex, and the exact matching of vertex labels is sometimes not feasible or practical. Education at present even though given at its good form, it is not being given at its best.

This is an attempt to present social network at its best form so that users start finding interests by themselves, and can share information among their friends , thus helps them to complete the course or the user they work on. The entire application is built using Google App Engine and hence all application data's are stored in Google's Server which is much secured and cost effective. The appointment could be made for any business Purpose in a very user interactive way. It helps Business people to manage their client's easily in time and Service them.

2. Literature Survey

2.1 Fast graph pattern matching

Due to rapid growth of the Internet technology and new scientific/technological advances, the number of applications that model data as graphs increases, because graphs have high expressive power to model complicated structures. The dominance of graphs in real-world applications asks for new graph data management so that users can access graph data effectively and efficiently. In this paper, we study a graph pattern matching problem over a large data graph. The problem is to find all patterns in a large data graph that match a user-given graph pattern. We propose a new two-step R-join (reach ability join) algorithm with filter step and fetch step based on a cluster-based join-index with graph codes. We consider the filter step as an R-semijoin, and propose a new optimization approach by interleaving R-joins with R-semijoins. We conducted extensive performance studies, and confirm the efficiency of our proposed new approaches. A graph provides great expressive power to describe and understand the complex relationships among data objects. With the rapid growth of World-Wide-Web, new data archiving and analyzing techniques, there exists a huge volume of data available in public, which is graph structured in nature including hypertext data, semi-structured data [1]. RDF also allows users to explicitly describe semantic resource in graphs [7]. In [27], Shasha et al. highlighted algorithms and applications for tree and graph searching including graph/subgraph matching in data graphs. The demand increases to query graphs over a large data graph. In this paper, we study a graph pattern matching problem that is to retrieve all patterns in a large graph, GD, that match a user-given graph pattern, Gq, based on reachability. As an example, based on business relationships, a graph pattern can be specified as to find Supplier, Retailer, Whole seller, and Bank such that Supplier directly or indirectly supplies products to Retailer and Whole-seller, and all of them receive services from the same Bank directly or indirectly over a large data graph which can be obtained from the Web.

2.2 A tool for approximate large graph matching

Large graph datasets are common in many emerging database applications, and most notably in large-scale scientific applications. To fully exploit the wealth of information encoded in graphs, effective and efficient graph matching tools are critical. Due to the noisy and incomplete nature of real graph datasets, approximate, rather than exact, graph matching is required. Furthermore, many modern applications need to query large graphs, each of which has hundreds to thousands of nodes and edges. This paper presents a novel technique for approximate matching of large graph queries. We propose a novel indexing method that incorporates graph structural information in a hybrid index structure. This indexing technique achieves high pruning power and the index size scales linearly with the database size. In addition, we propose an innovative matching paradigm to query large graphs. This technique distinguishes nodes by their importance in the graph structure. The matching algorithm first matches the important nodes of a query and then progressively extends these matches. Through experiments on several real datasets, this paper demonstrates the effectiveness and efficiency of the proposed method.

2.3 Torque: topology-free querying of protein interaction networks

TORQUE is a tool for cross-species querying of protein-protein interaction networks. It aims to answer the following question: given a set of proteins constituting a known complex or a pathway in one species, can a similar complex or pathway be found in the protein network of another species? To this end, Torque seeks a matching set of proteins that are sequence similar to the query proteins and span a connected region of the target network, while allowing for both insertions and deletions. Unlike existing approaches, TORQUE does not require knowledge of the interconnections among the query proteins. It can handle large queries of up to 25 proteins.

3. Existing System

Generally social networking websites does not contain the graph concept. And all e-current system has same content to all users. We can't able to create dynamic content, and if any user want to filter the data by some constraint of interests, the user has to click and visit each and every profile to get the data. This method is time consuming and in efficient. Many studies have been conducted on seeking the efficient solution for subgraph similarity search over certain (deterministic) graphs due to its wide application in many fields, including bioinformatics, social network analysis, and Resource Description Framework (RDF) data management. All these works assume that the underlying data are certain. However, in reality, graphs are often noisy and uncertain due to various factors, such as errors in data extraction, inconsistencies in data integration, and privacy preserving purposes. Therefore, in this paper, we study subgraph similarity search on large probabilistic graph databases.

ISSN: 2320-1363

3.1 Disadvantages of Existing

1. Fetching Interests and Filtering users is done manually and consumes lot of time and man work.
2. Much overhead arises for the administrator to manage the users.
3. Users can't see the status regarding the friends update.
4. Users doesn't have any intimation regarding the date and time of the results

ARCHITECTURE

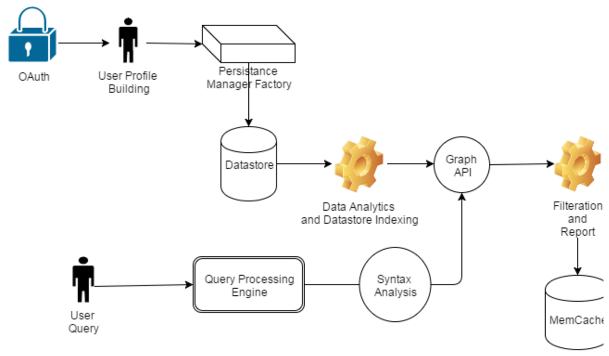


Figure 1.1 Architecture

4. PROPOSED SYSTEM

In our proposed system we introduce a new feature of search query using Graph Algorithm. Graph Databases are rapidly increasing in popularity, size and application. Currently, graph query processing involves some form of isomorphism test, which results in very high response times. Indexing is the most popular way to optimize query processing times. In this paper, we compare some of the existing work on subgraph query processing including cIndex, gIndex and FG-Index. There is a precision performance trade-off involved in subgraph queries on graph databases. There is a need to distinguish efficient querying methods tailored to certain applications. By analyzing the state of the art and comparing the methods in use, we can identify the key aspects in each and build a new indexing mechanism that can be adjusted according to the application and boost performance. This new index will map graphs in the dataset onto a plane and borrow some properties of similarity search techniques to greatly reduce the size of the candidate set of graphs on which the isomorphism test is performed.

4.1 ADVANTAGES OF PROPOSED

One of the best query approach on a graph database is the subgraph query; where, given a graph, it is required to determine a subgraph that satisfies certain properties.

ISSN: 2320-1363

Subgraph queries include both exact querying, where we need to find exactly the same subgraph as the query q, and approximate solutions, where a subgraph close enough to q will be deemed satisfactory. Algorithms for queries of this type can be feature, closure or coding based, or verification based approaches. The first works in this area concentrated on filtering and candidate verification. The first process filters the dataset according to some query parameters. This filtered set of graphs is much smaller than the original dataset. The second step, candidate verification, is a subgraph isomorphism problem. Even if the number of graphs is greatly reduced, performing this operation every time the database is queried proves to be inexcusably expensive. Once it was accepted that the filter-verify approach is not producing results as expected, especially given the way datasets were expanding, different indexing approaches were proposed.

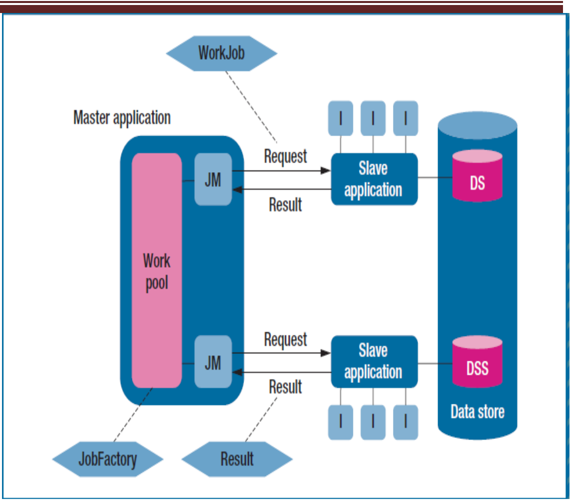


Fig 1.2 Parallel Sync

Our parallel computing framework architecture. The boxes labelled I denote multiple slave instances. The master application is responsible for generating and distributing the work among parallel slaves implemented as GAE Web applications and responsible for the actual computation.

5. MAIN MODULES

The system after careful analysis has been identified to be presented with the Following modules.

The modules involved are:

- OAUTH (Open Authentication)
- User Interests
- Friend List
- Notifications

- API Synchronization
- Syntax Analysis
- Filtration and Report

5.1 Module Description

In this section, the total web service repository building is demonstrated. The QoS on web services is highly essential to be maintained. Unfortunately the existing system doesn't meet the QoS standards. To maintain QoS on web services, the following methodology is used.

5.1.1 OPEN AUTHENTICATION

In general, this module deals with authenticating the user inside the application, this is considered to be the efficient and secured method in order to allow the user to be authenticated inside the application. An open authentication protocol is an open standard to authorization. It specifies a process for resource owners to authorize third-party access to their server resources without sharing their credentials. Designed specifically to work with Hypertext Transfer Protocol (HTTP), OAuth essentially allows access tokens to be issued to third-party clients by an authorization server, with the approval of the resource owner, or end-user. The client then uses the access token to access the protected resources hosted by the resource server.

Steps involved in OAuth

1. Obtain OAuth 2.0 credentials

Both Service Provider (SP) like Google and the application know OAuth 2.0 credentials such as a client ID and client secret. The set of values varies based on what type of application you are building. For example, a JavaScript application does not require a secret, but a web server application does.

2. Obtain an access token from the Google Authorization Server.

Before your application can access private data using a Google API, it must obtain an access token that grants access to that API. A single access token can grant varying degrees of access to multiple APIs. A variable parameter called scope controls the set of resources and operations that an access token permits. During the access-token request, your application sends one or more values in the scope parameter. There are several ways to make this

ISSN: 2320-1363

request, and they vary by type of application you are building.

For example, a JavaScript application might request an access token using a browser redirect to Google, while an application installed on a device that has no browser uses web service requests. Some requests require an authentication step where the user logs in with their Google account. After logging in, the user is asked whether they are willing to grant the permissions that your application is requesting. This process is called ***user consent***. If the user grants the permission, the Google Authorization Server sends your application an access token (or an authorization code that your application can use to obtain an access token). If the user does not grant the permission, the server returns an error.

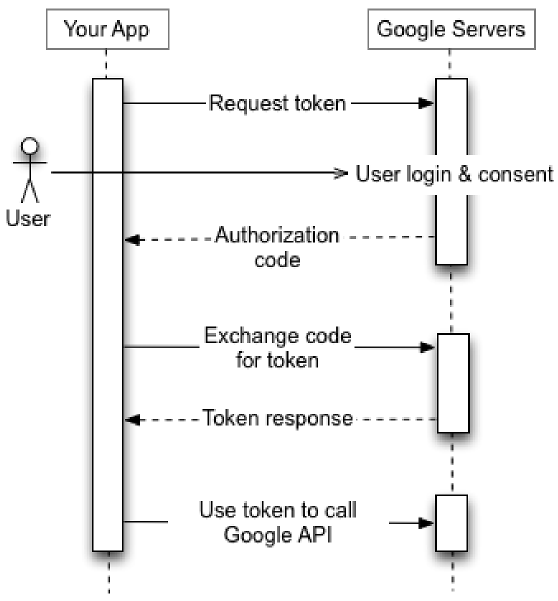
3. Send the access token to an API.

After an application obtains an access token, it sends the token to a Google API in an HTTP authorization header. It is possible to send tokens as URI query-string parameters, but we don't recommend it, because URI parameters can end up in log files that are not completely secure. Also, it is good REST practice to avoid creating unnecessary URI parameter names.

Access tokens are valid only for the set of operations and resources described in the scope of the token request.

4. Refresh the access token, if necessary.

Access tokens have limited lifetimes. If your application needs access to a Google API beyond the lifetime of a single access token, it can obtain a refresh token. A refresh token allows your application to obtain new access tokens. Final after getting the token from google, the key is exchanged between the google server and the web application, then if the results matches, then the user is allowed to enter in to the application.



ISSN: 2320-1363

Figure.4.3.1 Open Authentication

5.2WEB SERVICE IDENTIFICATION

In this module, the web services based on the functionalities required are generated. In general, each web services are intended to perform a separate operation. A web service repository is a set of disjoint services. We denote it as {w1, w2,….}

5.3 FUNCTIONALITY REQUEST

As the growth of web related requirements increases gradually, the users who interact with the web needs the system to system interaction, created the need for a structure which provided interaction not only with the user but also help application to application interaction. This problem called as Application Integration which is resolved by Web Services. Moreover, with the addition of the intelligence and autonomy of software agents, transactions may be equally automated for consumer-to-consumer, business-to-consumer, and business-to-business collaborations.

But it is a clear observation that number of web services is increased where a problem is raised to find an appropriate web service which satisfies user needs. The user wants an efficient way to find an appropriate web service which satisfied his needs in a short time. The functionality is mainly decided based on the input that is set for the web services and the output of the web services that the user needs. This is done in this module.

5.4 User Interests

- In this phase, we build the user profile, obtaining various interests from the users.
- The data obtained from the users are persisted on Google Datastore using Datastore API and set ready for indexing. The persistence is done using Persistence Manager Factory.

5.5 Data Analytics

In this module, the Datastore indexing operation is performed in order to perform the JSON mapping among the nodes. JSON Mapping is performed to analyze the statistics among the interests and data linking between various node and edges.

5.6 Syntax Analysis

- Graph API requires the predefined set of string required for indexing. The set can include the terms listed in the API:
- Eg : List of, Get the, Where we, How to
- So in this module, we set the list of predefined strings.

5.7 Filtration and Reports

In this phase, the results are obtained via querying through Graph API and the reports / output is rendered in View. For View, the dataTable API is used.

6. Conclusion

In this project, we study the problem of subgraph matching with set similarity, which exists in a wide range of applications. To tackle this problem, we propose efficient pruning techniques by considering both vertex set similarity and graph topology. Finally, we propose an efficient dominating-set based subgraph match algorithm to find subgraph matches. Extensive experiments have been conducted to demonstrate the efficiency and effectiveness of our approaches compared to state-of-the-art subgraph matching methods.

7. Future Enhancements

- ✓ In this future, the concept of 'Graph Algorithm' will be able to handle the user information where links among the sub links of user information is made.
- ✓ It matches with set similarity (SMS2) query over a large graph database, which retrieves sub graphs that are structurally isomorphic to the query graph, and meanwhile satisfy the condition of vertex pair matching with the (dynamic) weighted set similarity.
- ✓ Based on the index and signatures, we propose an efficient two-phase pruning strategy including set similarity pruning and structure-based pruning, which exploits the unique features of both (dynamic) weighted set similarity and graph topology

ISSN: 2320-1363

REFERENCES

[1] L. Zou, L. Chen, and M. T. Ozsu, "Distance-join: Pattern match q

[2] J. Cheng, J. X. Yu, B. Ding, P. S. Yu, and H. Wang, "Fast graph pattern matching," in Data Engineering, 2008. ICDE 2008. IEEE 24th International Conference on. IEEE, 2008, pp. 913–922.

[3] Y. Tian and J. M. Patel, "Tale: A tool for approximate large graph matching," in ICDE, 2008.

[4] S. Bruckner1, F. Huffner, R. M. Karp, R. Shamir, and R. Sharan, "Torque: topology-free querying of protein interaction networks," Nucleic Acids Research, vol. 37, no. suppl 2, pp. W106–W108, 2009.

[5] Y. Tian, R. C. McEachin, C. Santos, D. J. States, and J. M. Patel, "Saga: a subgraph matching tool for biological graphs," Bioinformatics, vol. 23, no. 2, pp. 232–239, 2007.

[6] P. Zhao and J. Han, "On graph query optimization in large networks," PVLDB, vol. 3, no. 1-2, 2010.

[7] Z. Sun, H. Wang, H. Wang, B. Shao, and J. Li, "Efficient subgraph matching on billion node graphs," PVLDB, vol. 5, no. 9, 2012.

[8] W. Lu, J. Janssen, E. Milios, N. Japkowicz, and Y. Zhang, "Node similarity in the citation graph," Knowledge and Information Systems, vol. 11, no. 1, pp. 105–129, 2006.

[9] S. Auer, C. Bizer, G. Kobilarov, J. Lehmann, and Z. Ives, "Dbpedia: A nucleus for a web of open data," in ISWC, 2007.

[10] M. Hadjieleftheriou and D. Srivastava, "Weighted set-based string similarity," in IEEE Data Engineering Bulletin, 2010.

[11] L. P. Cordella, P. Foggia, C. Sansone, and M. Vento, "A (sub)graph isomorphism algorithm for matching large graphs," PAMI, vol. 26, no. 10, 2004.

[12] W.-S. Han, J. Lee, and J.-H. Lee, "Turboiso: Towards ultrafast and robust subgraph isomorphism search in large graph databases," in SIGMOD, 2013.

[13] H. He and A. K. Singh, "Closure-tree: An index structure for graph queries," in ICDE, 2006.

[14] J. R. Ullmann, "An algorithm for subgraph isomorphism," Journal of the ACM, vol. 23, no. 1, 1976.

[15] S. Zhang, M. Hu, and J. Yang, "Treepi: A novel graph indexing method." in ICDE, vol. 7, 2007, pp. 966–975.

[16] S. Zhang, S. Li, and J. Yang, "Gaddi: Distance index based subgraph matching in biological networks," in EDBT, 2009.

[17] H. Shang, Y. Zhang, X. Lin, and J. X. Yu, "Taming verifi- cation hardness: An efficient algorithm for testing subgraph isomorphism," Proceedings of the VLDB Endowment, vol. 1, no. 1, 2008.

[18] R. Di Natale, A. Ferro, R. Giugno, M. Mongiov`ı, A. Pulvirenti, and D. Shasha, "Sing: Subgraph search in nonhomogeneous graphs," BMC bioinformatics, vol. 11, no. 1, p. 96, 2010.

[19] H. He and A. K. Singh, "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. ACM, 2008, pp. 405–418.