Large-Scale Continuous Subgraph Queries on Streams

Sutanay Choudhury
Pacific Northwest National Laboratory
902 Battelle Boulevard
Richland, WA, USA 99352
sutanay.choudhury@pnnl.gov

Lawrence Holder
School of Electrical Engineering & Computer Science
Washington State University
Pullman, WA, USA 99164-2752
holder@wsu.edu

George Chin
Pacific Northwest National Laboratory
902 Battelle Boulevard
Richland, WA, USA 99352
george.chin@pnnl.gov

John Feo
Pacific Northwest National Laboratory
902 Battelle Boulevard
Richland, WA, USA 99352
john.feo@pnnl.gov

ABSTRACT

Graph pattern matching involves finding exact or approximate matches for a query subgraph in a larger graph. It has been studied extensively and has strong applications in domains such as computer vision, computational biology, social networks, security and finance. The problem of exact graph pattern matching is often described in terms of subgraph isomorphism which is NP-complete. The exponential growth in streaming data from online social networks, news and video streams and the continual need for situational awareness motivates a solution for finding patterns in streaming updates. This is also the prime driver for the real-time analytics market. Development of incremental algorithms for graph pattern matching on streaming inputs to a continually evolving graph is a nascent area of research. Some of the challenges associated with this problem are the same as found in continuous query (CQ) evaluation on streaming databases. This paper reviews some of the representative work from the exhaustively researched field of CQ systems and identifies important semantics, constraints and architectural features that are also appropriate for HPC systems performing real-time graph analytics. For each of these features we present a brief discussion of the challenge encountered in the database realm, the approach to the solution and state their relevance in a high-performance, streaming graph processing framework.

1. INTRODUCTION

Graph pattern matching is defined as the problem of searching a graph for all instances of a pattern that is also expressed as a graph [11]. This is mathematically defined as the problem of subgraph isomorphism, which given a pattern or query graph G_q and a larger input graph G_d, requires finding all isomorphisms of G_q in G_d. Following the definition of isomorphism, the matching involves finding a one-to-one correspondence between the vertices of a subgraph of G_d and vertices of G_q such that all vertex adjacencies are preserved. Dynamic graphs refer to graphs that evolve over time through addition or deletion of vertices and edges. Therefore, the problem of graph pattern matching for dynamic graphs can be described as the *continuous* process of searching for patterns in the graph as it is updated. News [23], finance [7], cyber security and intelligence [10] are among the primary domains that drive the real-time analytics market [1, 19] and motivate development of HPC systems. These domains present data sources that lend themselves naturally to a graph based representation and additionally, provide semantic information in terms of types, labels and timestamps, which can be more generally described as attributes of the vertices and edges of the graph. The availability of the attributes influence the isomorphism computation because assigning a correspondence between a pair of vertices in the search and query graph requires them to satisfy equality constraints on type and possibly, other attributes as well. All these domains are also characterized by massive streaming data that are continuously providing updates from social networks, financial markets and malicious activities on the internet with a high emphasis on time-to-insight, the capability of learning about an event as soon as it happens. This motivates our investigation of subgraph pattern matching on streams using high-performance computing architectures.

1.1 Continuous Queries in Databases

This problem can also be described in general as computing a function f over a stream S over time and notifying the user whenever the output of f satisfies a user-defined constraint. A *continuous query system* is defined as one where a query logically runs continuously over time as opposed to being executed intermittently and then running to completion [20]. There are some obvious challenges associated with computing continuous queries [5]. In the following, we state them as two focus areas from our perspective.

Area I The memory requirement for computing f and maintaining its partial results over time may be unbounded. Adding constraints, such as restricting the class of queries to control their complexity or using approximation techniques, is often the solution to this problem. Incorporation of temporal aspects such as specifying order of arrival for entities in the data are examples of such efforts to reduce

memory requirements. We view this as an area focusing on identifying practical constraints found in real-life applications to make the problem tractable.

Area II Timeliness requirements of query processing is set by the underlying application or the end-user. Typically there exists a data source (e.g. a relational or graph database) that needs to be updated with the streaming input and computation of f. The relative cost of updating the data source and computation of f strongly influences processing strategies such as on-the-fly processing, batch processing, incremental processing, sampling etc. We broadly view this under the category of algorithmic challenges. Continuous queries are also distinguished from ad-hoc query processing by their high selectivity (looking for unique events) and need to detect newer updates of interest as opposed to retrieving lots of past information. Conventional databases are passive repositories with large collections of data that works in a request-response model, whereas continuously monitoring applications are data-driven, or trigger oriented. These features coupled with the real-time demands challenged many of the fundamental assumptions for conventional databases and established continuous queries processing on relational data streams as a major research area.

1.2 Applications to High-Performance Graph Analytics

The emergence of continuous subgraph matching on massive data streams can be viewed as a resurrection of the above research theme, albeit in the graph domain. The paramount importance of timeliness of detection of a match dictates that the runtime of a query should be dependent on size of the latest information update and independent of the size of the database. Given the massive scale of the problem and the high computational complexity of the algorithm one is naturally tempted to turn towards established parallel and distributed computing techniques. Unfortunately, the tools and techniques that perform exceptionally well for traditional HPC applications are not equally effective for graph-theoretic applications due to issues such as irregular memory access and data intensive nature characterized by high data access to computation ratio [15]. Hence for Area II, we are actively developing an incremental query processing processing framework [9] for massively multi-threaded architectures such as the Cray XMT [17]. We believe the availability of rich semantic information in terms of labels, timestamps and other attributes in the graph data primes them for leveraging on findings from earlier CQ research. In the broader context, we are interested in exploring useful but restricted classes of queries where the subgraph matching problem may be tractable (Area I) and will benefit other large-scale graph processing frameworks in an architecture agnostic way.

The contribution of this paper is to draw the attention of the parallel graph mining community to the approaches undertaken by the database community for challenges that are similar in spirit and stimulate discussions on their extension to the graph based applications. Section 2 presents a brief overview of the continuous query systems research in the database community as well as of research on graph queries. We present an example graph query and illustrate the associated challenges in section 3. We review the approaches adapted by the database community to address similar challenges and discuss their relevance for the running example in section 4. Section 5 concludes with our discussions and suggestions for future research.

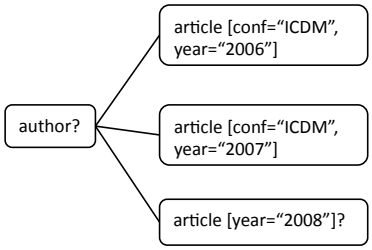

Figure 1: Example graph query that emerges over time

2. RELATED WORK

The following sections provide a quick overview of past research from both the database and graph mining community. For the former, we omit important topics such as load-shedding on streams or approximate techniques for stream-based queries and restrict our attention to works relevant to exact subgraph matching.

2.1 Continuous Queries in Databases

The literature on database research from past the two decades is abundant with work on continuous query systems [8, 3]. A common theme across various solutions are the implementations of window-based operators that exposed a data stream as a buffer or queue on which traditional relational operators as selection, projection and joins were applied [2]. A large body of work was dedicated to extending SQL and incorporation of continuous query semantics by adding various constraints [4, 13]. Continuous query processing involved two types of activities like conventional databases. Query registration involved generating an optimal query plan at compile-time and the run-time processing can be viewed as a dataflow system where a tuple flows through a directed graph of processing operators [6]. However, adaptive query processing systems [16] are an exception to this approach due their dynamic routing of tuples through various operators.

2.2 Graph Querying Systems

Subgraph matching in static graphs has been studied extensively. Recently, there has been a surge of interest in the database community that seeks to develop hybrid approaches by combining indexing techniques that exploit the distribution of labels in the graph along with graph edit distance or motif based approaches [21, 22, 14]. However, investigation of subgraph matching in the context of dynamic graphs did not receive much attention in the broader research community until recently. The work by Fan et al. [12] presents incremental algorithms for graph simulation, bounded simulation and subgraph isomorphism. They identify a class of queries for the first case where the problem is bounded and optimal. Our work [9] can be viewed as following the same spirit as focusing on a specific class of queries (temporal in this case) and exploring query constraints that would make the problem tractable.

3. INCREMENTAL FRAMEWORK FOR CONTINUOUS SUBGRAPH MATCHING

Consider the example query in Figure 1. Suppose we are interested in authors who have published a paper in the past two ICDM conferences and wish to follow their current work. Also assume that the publications database receives a batch of updates after every conference and we seek to query the database after every update for a match. Figure 2 shows the evolution of a matching subgraph in the publications data. It is obvious that a complete match will

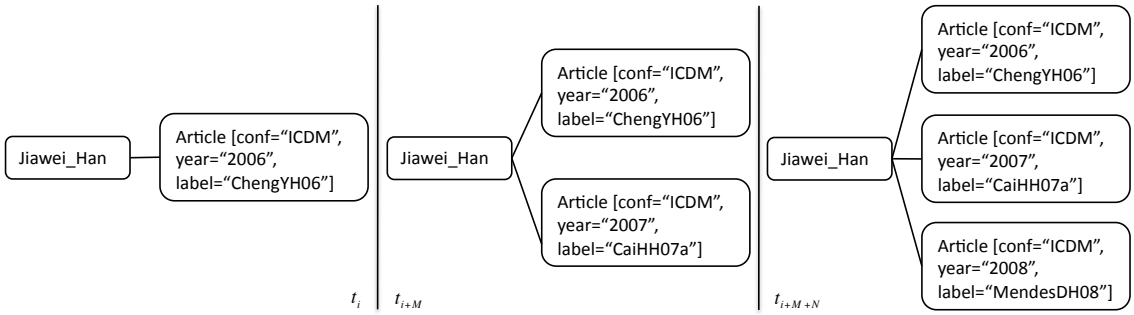

Figure 2: Evolution of a matching subgraph

not be found until time instance $(i + M + N)$ but more importantly, an approach employing an algorithm for subgraph matching in static graphs will search the entire graph beginning from scratch and perform a non-significant amount of work every time before reporting no entries matching the query pattern. An alternate approach is to track partial matches as they appear and augment them as more information becomes available.

Herein lies the first challenge. Most prominent algorithms for pattern matching in static graphs are designed to seek complete matches with the pattern and try to aggressively prune their search space using semantic and structural properties of the query graph. As an example, if searching the database at time i (Figure 2), a traditional algorithm will begin from a vertex of type author and require that it has at least three neighboring vertices with matching attributes (as shown in Figure 1). This check will fail as the graph at time i does not meet this requirement. On the other hand, a simple incremental algorithm expands its search space by matching an edge at a time. When the first edge arrives in the graph that matches with any edge in the query graph, a partial match is created with appropriate mapping information. When the next matching edge arrives, another new partial match is created and the previous partial match is augmented with an additional edge. Such partial match augmentations will continue until a complete match is found. While the discussion of complex boundary conditions are beyond the scope of this paper, it is not hard to see the exponential, unbounded growth in the number of partial matches over time with this strategy.

4. CROSS-CUTTING THEMES

4.1 Temporal Query Constraints

Apparently, there are a number of options one can adapt to improve the performance. If an author's first ICDM publication was in 2007, the above approach will create a partial match when that information arrives. Assuming no out-of-order arrival, this partial match can never yield a complete match in future because our query also requires a matching publication from 2006. Also, all partial matches which represent an author with one or more 2006 ICDM publications but with no 2007 ICDM publication can be pruned as well. In short, these advocate strategies which are *time-aware*. These are precisely the issues discussed in [6] and presented as *ordered-arrival constraints*. However, introduction of an order or exploiting such properties introduces a new set of challenges such as out of order arrival of data. For real applications the query processor may need to be robust to arrival patterns. At this point we are actively working on finding a set of representative temporal queries and studying the performance benefits of such constraints. *Clustered-arrival constraint* is another useful concept in addition to arrival order. It can be used to describe the acceptable interval between the arrival of tuples in a database or edges in a graph. The interval can be specified in terms of time or the number of entities/updates. Implementation of this constraint will require a query processor to prune its set of partial results by removing stale entries. It can be viewed as a broader version of a windowed query which requires satisfaction of a predicate over the subset of data arriving within the specified time window.

4.2 Adaptive Query Processing

Another important aspect of the research on query constraints involved monitoring the data stream itself for realistic estimates of the constraint parameters [6, 16]. In the database context this is accomplished by maintaining synopsis data structures and algorithms that impose minimum overhead. The objective of this monitoring is twofold. The fundamental assumption is that once a query is registered and runs over a long period of time, the characteristics of the data stream can change over time. This fact may stay unknown to the user running the query. For an example, let's assume a query that monitors packets flowing through multiple networks. Because the data packets can arrive out of order, a domain driven approach will account for latencies in the network infrastructure as part of the monitoring process. Further assume that partial matches that do not yield a complete match within a specified time window are excluded from tracking. If the expected value of these latency intervals drop over time then the query processor will be overestimating that period and maintain a larger set of partial results. On the other hand, if the expected interval span increased over time then the query processor will be producing potentially incorrect results. In the graph stream processing context, our foremost objective is to develop strategies for pruning the set of partial matches or being selective about initiating a partial match. There are several metrics for structural or semantic summaries of a graph dataset that are amenable to scalable implementation. The above mentioned works motivate us to expose those properties to a query optimizer or composer module in a generic fashion.

4.3 Multiple Query Processing

For the example describe above, a different strategy may focus on reducing the number of partial matches maintained in memory. The number of matches drop dramatically as publications from longer sequence of years are considered; hence, it may decide to check for a specific subset of the match to arrive before spawning a new partial match. This involves performing a subgraph isomorphism check and can be expensive for even a small substructure. However, if multiple queries share the same substructure, the cost of the detecting the subset can be compensated. A real-life query process-

ing system is expected to have a large number of queries registered in it and their efficient handling is an important criteria for scalability. While scaling up or scaling out is a design choice determined by the application architecture, simply adding more computing resources to match the increase in number of queries is not sufficient or optimal. Exploiting common workload across multiple continuous queries is a recurrent theme both in the database [16, 18] and graph mining [21].

5. CONCLUSIONS AND RESEARCH PLAN

The literature on continuous query processing for relational streams is a rich source of guidance for the emerging area of subgraph pattern matching on streams. As various parallel and distributed computing systems emerge for processing massive scale graphs, they need to be complemented with appropriate algorithmic approaches for enabling the same innovations accomplished with high throughput relational data stream systems. This work serves as a reminder that continuous query processing systems are fundamentally different from conventional query processing systems that follow a command-response model and hence, motivates the rethinking of query processing and optimization strategies. Algorithmic approaches for query processing are obviously different between relational database systems and graph databases; however, the concepts such as ordered-arrival and other temporal constraints are immediately relevant to their graph querying counterparts. Areas as adaptive query processing and multiple query processing can be considered as next-level optimization goals once the first-generation of graph based continuous query systems are developed.

6. REFERENCES

[1] Storm: The hadoop of real-time processing: http://tech.backtype.com.

[2] D. J. Abadi, D. Carney, U. Çetintemel, M. Cherniack, C. Convey, S. Lee, M. Stonebraker, N. Tatbul, and S. Zdonik. Aurora: a new model and architecture for data stream management. *The VLDB Journal*, 12:120–139, August 2003.

[3] A. Arasu, B. Babcock, S. Babu, M. Datar, K. Ito, I. Nishizawa, J. Rosenstein, and J. Widom. Stream: The stanford stream data manager. In *SIGMOD Conference*, page 665, 2003.

[4] A. Arasu, S. Babu, and J. Widom. The cql continuous query language: semantic foundations and query execution. *The VLDB Journal*, 15:121–142, June 2006.

[5] B. Babcock, S. Babu, M. Datar, R. Motwani, and J. Widom. Models and issues in data stream systems. In *Proceedings of the 21st ACM SIGMOD-SIGACT-SIGART symposium on Principles of database systems*, PODS '02, pages 1–16, New York, NY, USA, 2002. ACM.

[6] S. Babu, U. Srivastava, and J. Widom. Exploiting k-constraints to reduce memory overhead in continuous queries over data streams. *ACM Trans. Database Syst.*, 29:545–580, September 2004.

[7] B. Chandramouli, J. Goldstein, and D. Maier. High-performance dynamic pattern matching over disordered streams. *Proc. VLDB Endow.*, 3:220–231, September 2010.

[8] S. Chandrasekaran, O. Cooper, A. Deshpande, M. J. Franklin, J. M. Hellerstein, W. Hong, S. Krishnamurthy, S. R. Madden, F. Reiss, and M. A. Shah. Telegraphcq: continuous dataflow processing. In *Proceedings of the 2003 ACM SIGMOD international conference on Management of data*, SIGMOD '03, pages 668–668, New York, NY, USA, 2003. ACM.

[9] S. Choudhury, L. Holder, G. Chin, and J. Feo. Incremental Pattern Matching for Temporal Queries in Attributed Dynamic Graphs. *PNNL Tech Report, SA-82215*, 2011.

[10] T. Coffman, S. Greenblatt, and S. Marcus. Graph-based technologies for intelligence analysis. *Commun. ACM*, 47:45–47, March 2004.

[11] D. Conte, P. Foggia, C. Sansone, and M. Vento. Thirty years of graph matching in pattern recognition. *International Journal of Pattern Recognition and Artificial Intelligence*, 18(3):265–298, 2004.

[12] W. Fan, J. Li, S. Ma, N. Tang, Y. Wu, and Y. Wu. Graph pattern matching: from intractable to polynomial time. *Proc. VLDB Endow.*, 3:264–275, September 2010.

[13] N. Jain, S. Mishra, A. Srinivasan, J. Gehrke, J. Widom, H. Balakrishnan, U. Çetintemel, M. Cherniack, R. Tibbetts, and S. Zdonik. Towards a streaming sql standard. *Proc. VLDB Endow.*, 1:1379–1390, August 2008.

[14] A. Khan, N. Li, X. Yan, Z. Guan, S. Chakraborty, and S. Tao. Neighborhood based fast graph search in large networks. In *Proceedings of the 2011 international conference on Management of data*, SIGMOD '11, pages 901–912, New York, NY, USA, 2011. ACM.

[15] A. Lumsdaine, D. Gregor, B. Hendrickson, and J. W. Berry. Challenges in parallel graph processing. *Parallel Processing Letters*, 17(1):5–20, 2007.

[16] S. Madden, M. Shah, J. M. Hellerstein, and V. Raman. Continuously adaptive continuous queries over streams. In *Proceedings of the 2002 ACM SIGMOD international conference on Management of data*, SIGMOD '02, pages 49–60, New York, NY, USA, 2002. ACM.

[17] D. Mizell and K. Maschhoff. Early experiences with large-scale cray xmt systems. In *Proceedings of the 2009 IEEE International Symposium on Parallel&Distributed Processing*, pages 1–9, Washington, DC, USA, 2009. IEEE Computer Society.

[18] K. Munagala, U. Srivastava, and J. Widom. Optimization of continuous queries with shared expensive filters. In *Proceedings of the twenty-sixth ACM SIGMOD-SIGACT-SIGART symposium on Principles of database systems*, PODS '07, pages 215–224, New York, NY, USA, 2007. ACM.

[19] D. Peng and F. Dabek. Large-scale incremental processing using distributed transactions and notifications. In *Proceedings of the 9th USENIX conference on Operating systems design and implementation*, OSDI'10, pages 1–15, Berkeley, CA, USA, 2010. USENIX Association.

[20] D. Terry, D. Goldberg, D. Nichols, and B. Oki. Continuous queries over append-only databases. *SIGMOD Rec.*, 21:321–330, June 1992.

[21] S. Zhang, J. Yang, and W. Jin. SAPPER: Subgraph Indexing and Approximate Matching in Large Graphs. *Proceedings of the VLDB Endowment*, 3(1), 2010.

[22] P. Zhao and J. Han. On graph query optimization in large networks. *Proc. VLDB Endow.*, 3:340–351, September 2010.

[23] Q. Zhao, P. Mitra, and B. Chen. Temporal and information flow based event detection from social text streams. In *Proceedings of the 22nd national conference on Artificial intelligence*, pages 1501–1506. AAAI Press, 2007.