Poster: Temporal Pattern Recognition in Large Scale Graphs

Hassan Nazeer Chaudhry
Politecnico di Milano
hassannazeer.chaudhry@polimi.it

Alessandro Margara
Politecnico di Milano
alessandro.margara@polimi.it

Matteo Rossi
Politecnico di Milano
matteo.rossi@polimi.it

ABSTRACT

Many application domains involve monitoring the temporal evolution of large-scale graph data structures. Unfortunately, this task is not well supported by modern programming paradigms and frameworks for large-scale data processing. This paper presents ongoing work on the implementation of FlowGraph, a framework to recognize temporal patterns over properties of large-scale graphs. FlowGraph combines the programming paradigm of traditional graph computation frameworks with the temporal pattern detection capabilities of Complex Event Recognition (CER) systems. In a nutshell, FlowGraph distributes the graph data structure across multiple nodes that also contribute to the computation and store partial results for pattern detection. It exploits temporal properties to defer as much as possible expensive computations, to sustain a high rate of changes.

KEYWORDS

Dynamic graphs processing, pattern recognition, stream processing

ACM Reference format:
Hassan Nazeer Chaudhry, Alessandro Margara, and Matteo Rossi. 2019. Poster: Temporal Pattern Recognition in Large Scale Graphs. In *Proceedings of DEBS '19, Darmstadt, Germany, June 24-28, 2019,* 2 pages.
https://doi.org/10.1234/XXXXXXX.XXXXXXX

1 MOTIVATION

Many application scenarios involve large-scale graph-based data structures that continuously evolve over time. For example, in social networks new posts can refer to existing ones and mention users, which in turn are connected by a "follower" relation that changes over time. Common problems in these scenarios entail capturing the temporal evolution of properties that depend on the structure of the graph. For instance, understanding the evolution of the relations between users in social networks can help customize the interface and improve the user experience.

The above problems are challenging. On the one hand, they demand the capability to perform computations on large-scale graphs: such computations are often iterative in nature and expensive, as exemplified by well known algorithms for path discovery, cluster detection, ranking, etc. On the other hand, they require analyzing the evolution of the graph and its properties, keeping up with a possibly high rate of changes.

Unfortunately, existing frameworks for large-scale data processing do not meet these requirements. Graph processing systems [8] enable scalable distributed graph computations through a programming paradigm known as *think like a vertex* (TLAV) [10], introduced in 2010 with the Pregel system [9]. TLAV exploits a bulk synchronous parallel programming model [6], where the computation is split into supersteps (epochs): at each superstep, a vertex can perform some computation that changes its internal state and/or send messages to other vertices. This paradigm simplifies the distribution of state and computation over multiple processing nodes, but only refers to *static graphs* that do not change over time. Stream processing systems analyze dynamic data as it becomes available, to derive relevant information and enable timely reactions [4]. Modern big data processing platforms such as Apache Spark Streaming [12] and Apache Flink [2] offer steam processing capabilities by implementing functional operators that transform input streams into output streams. A stream processing job is represented as a workflow of such operators, which are then deployed over multiple processing nodes. However, operators are designed to only store the limited state that is strictly needed to compute the desired results and offer limited or no support for updating large-scale data stores, as required in dynamic graph processing.

Despite some initial investigation in the area [5, 7, 11], the problem of defining a programming abstraction and processing framework to analyze the evolution of large-scale graphs remains open. In this paper, we tackle this problem by introducing a novel programming paradigm that integrates the traditional TLAV graph processing model with the temporal pattern detection capabilities of stream processing systems, and in particular of Complex Event Recognition (CER) systems [1]. We show how the paradigm is being implemented in FlowGraph, a distributed processing framework to detect temporal patterns in large-scale graphs. FlowGraph distributes the graph structure across multiple nodes that contribute to the computation and store partial results for pattern detection. It exploits temporal properties within patterns to defer as much as possible the execution of expensive computations, to sustain a high rate of changes.

2 THE FLOWGRAPH FRAMEWORK

This section overviews the FlowGraph framework we propose to detect temporal patterns on large-scale graphs. We first present the data and programming model (Section 2.1) and then the system implementation (Section 2.2).

2.1 Programming model

The FlowGraph data model assumes a graph $G(V, E)$ where V is a set of vertices and E is a set of edges. Both vertices and edges can have associated properties. For instance, in a social network,

DEBS '19, June 24-28, 2019, Darmstadt, Germany

ACM ISBN XXX-X-XXX-XXXX-X/XX/XX…$15.00
https://doi.org/10.1234/XXXXXXX.XXXXXXX

vertices can be users and edges their "follow" relations. Vertices can contain user data as properties. The structure of the graph—as well as the values of properties in vertices and edges—can change over time. We assume that the notifications of changes are received as streams from some external source.

The FlowGraph programming model integrates graph computations with temporal pattern recognition. Computations return relevant information from the state of the graph at a given point in time. These results are conceptually stored inside a key-value store partitioned across the vertices of the graph. Temporal patterns correlate the results of computations at multiple points in time to capture trends of interest in the application domain. Computations are internally encoded as TLAV programs (the established paradigm for large-scale graph computations). We assume a predefined library of computations is available, which can be called by name using the `execute()` operator, but developers can install new computations if needed. We use `select()` operators to identify sub-graphs based on the results of a computation. We can then run aggregations on the properties of the selected sub-graphs.

For instance, the following code snippet assumes that a `k-means` computation computes the k-means clustering algorithms, and stores within each vertex v an entry `(cluster, c)` where `c` is an identifier of the cluster v belongs to. The code snippet first computes `k-means` with `k=10`. Then, it selects each and every cluster, that is, any sub-graph such that all the vertices have the same value (parameter `$x`) for the `cluster` property. Finally, it computes the average value of the `age` attribute within each cluster. The pattern is recognized when the average value for any cluster is greater than 50.

```
graph.execute("k-means",10).select("cluster", $x)
     .avg("age") > 50
```

Patterns can involve temporal predicates. For instance, the following snippet shows a sequence that is satisfied when, within 10 seconds, the size of the largest cluster increases by more then 5.

```
graph.execute("k-means",10).select("cluster", $x)
     .size().max() = $max1
-> (10 sec)
graph.execute("k-means",10).select("cluster", $x)
     .size().max() > $max1 + 5
```

While they are far from being a complete overview of the programming model, the above snippets well exemplify its core concepts: computations that store their results inside vertices, parametric selection, aggregations, temporal operators, and parameter correlations. We are currently finalizing a formal specification of the pattern definition language, which is based on temporal logic and inspired by our previous work on CER [3].

2.2 System implementation

We are currently implementing the programming model above in the prototype FlowGraph framework. FlowGraph is developed in Java on top of the Akka actor system[1]. Similar to modern data processing platforms, the FlowGraph architecture comprises a master node that coordinates many worker nodes. Clients can connect to the master node and submit the patterns of interest together with the code of user-defined computations, if any. Workers store the state of the graph in main memory for improved performance. Specifically, graph vertices are partitioned across worker nodes, together with their outgoing edges. Workers execute computations using the TLAV paradigm, with the master acting as a synchronization point between epochs, and save the results of the computation locally, into a temporal key-value store.

Two key insights enable FlowGraph to optimize storage and processing, with the final goal of maximizing throughput and reducing latency as much as possible. First, old values are deleted from the local key-value stores as soon as they cannot influence the detection of any pattern anymore. Their time of validity is determined by statically analyzing the patterns when they are deployed into the system. Second, patterns are rewritten to defer expensive computations as much as possible. For instance, if a pattern involves checking if a node belongs to a given cluster and later in time it holds a specific value v for a given property p, the cluster computation is started retroactively only when a node is found with value v for property p. Pattern rewriting is based on a cost model currently under development.

[1] https://doc.akka.io/docs/akka/current/general/actor-systems.html

3 CONCLUSIONS

This paper introduces FlowGraph, a framework for temporal pattern recognition on graph data. FlowGraph integrates large-scale graph computations with time-based analysis. By distributing the processing effort and deferring expensive computations as much as possible, FlowGraph has the ambitious goal to detect relevant patterns with low delay in the presence of frequent changes.

REFERENCES

[1] Alexander Artikis, Alessandro Margara, Martin Ugarte, Stijn Vansummeren, and Matthias Weidlich. 2017. Complex Event Recognition Languages: Tutorial. In *Proc. of the Int. Conf. on Dist. and Event-based Sys. (DEBS '17)*. ACM, 7–10.

[2] Paris Carbone, Asterios Katsifodimos, Stephan Ewen, Volker Markl, Seif Haridi, and Kostas Tzoumas. 2015. Apache Flink: Stream and Batch Processing in a Single Engine. *IEEE Data Engineering Bullettin* 38, 4 (2015), 28–38.

[3] Gianpaolo Cugola and Alessandro Margara. 2010. TESLA: A Formally Defined Event Specification Language. In *Proc. of the Int. Conf. on Dist. Event-Based Sys. (DEBS '10)*. ACM, 50–61.

[4] Gianpaolo Cugola and Alessandro Margara. 2012. Processing Flows of Information: From Data Stream to Complex Event Processing. *Comput. Surveys* 44, 3 (2012), 15:1–15:62.

[5] Benjamin Erb, Dominik Meissner, Jakob Pietron, and Frank Kargl. 2017. Chronograph: A Distributed Processing Platform for Online and Batch Computations on Event-sourced Graphs. In *Proceedings of the International Conference on Distributed and Event-based Systems (DEBS '17)*. ACM, 78–87.

[6] Alexandros V. Gerbessiotis and Leslie G. Valiant. 1994. Direct Bulk-synchronous Parallel Algorithms. *J. Parallel and Distrib. Comput.* 22, 2 (1994), 251–267.

[7] Anand Padmanabha Iyer, Li Erran Li, Tathagata Das, and Ion Stoica. 2016. Time-evolving Graph Processing at Scale. In *Proceedings of the International Workshop on Graph Data Management Experiences and Systems (GRADES '16)*. ACM, 5:1–5:6.

[8] V. Kalavri, V. Vlassov, and S. Haridi. 2018. High-Level Programming Abstractions for Distributed Graph Processing. *IEEE Transactions on Knowledge and Data Engineering* 30, 2 (Feb 2018), 305–324. https://doi.org/10.1109/TKDE.2017.2762294

[9] Grzegorz Malewicz, Matthew H. Austern, Aart J.C Bik, James C. Dehnert, Ilan Horn, Naty Leiser, and Grzegorz Czajkowski. 2010. Pregel: A System for Large-scale Graph Processing. In *Proceedings of the International Conference on Management of Data (SIGMOD '10)*. ACM, 135–146.

[10] Robert Ryan McCune, Tim Weninger, and Greg Madey. 2015. Thinking Like a Vertex: A Survey of Vertex-Centric Frameworks for Large-Scale Distributed Graph Processing. *Comput. Surveys* 48, 2 (2015), 25:1–25:39.

[11] Youshan Miao, Wentao Han, Kaiwei Li, Ming Wu, Fan Yang, Lidong Zhou, Vijayan Prabhakaran, Enhong Chen, and Wenguang Chen. 2015. ImmortalGraph: A System for Storage and Analysis of Temporal Graphs. *Transactions on Storage* 11, 3, 14:1–14:34.

[12] Matei Zaharia, Tathagata Das, Haoyuan Li, Timothy Hunter, Scott Shenker, and Ion Stoica. 2013. Discretized Streams: Fault-tolerant Streaming Computation at Scale. In *Proc. of the Symp. on Op. Sys. Princ. (SOSP '13)*. ACM, 423–438.