MAGMA: Proposing a Massive Historical Graph Management System

Alexandros Spitalas and Kostas Tsichlas

University of Patras, Rion, Patra, Greece.
a.spitalas@upatras.gr,ktsichlas@ceid.upatras.gr

Abstract. In recent years, maintaining the history of graphs has become more and more imperative due to the emergence of related applications in a number of fields like health services, social interactions, and map guidance. Historical graphs focus on being able to store and query the whole evolution of the graph and not just the latest instance. In this paper we have two goals: 1) provide a concise survey of the state-of-art with respect to systems in historical graph management since no such comprehensive discussion exists and 2) propose an architecture for a distributed historical graph management system (named MAGMA - MAssive Graph MAnagement) based on previous research work of the authors.

Keywords: temporal graphs · graph management systems · query engine

1 Introduction

In recent years there is a rapid increase of time-evolving networks that produce a considerable amount of data. Networks, such as citation networks, traffic networks, and social networks are, naturally represented as graphs and they are usually dynamic. For example, in a citation network, new nodes and edges are constantly added due to the publication of new papers. An important challenge that arises in these time-evolving networks is the efficient management of their history in order to be able to reason about its whole evolution and not only about its latest state. This allows us to answer queries such as "what is the average connectivity of author X in the citation network between 2010 and 2015".

There have been quite a lot of systems developed since 2016 for historical graph management. Most of them are distributed, since evolving large graphs an extremely demanding with respect to space usage and query/update time. A rather outdated (since 2016) related survey can be found in [41]. They focus mainly on the models used for temporal graphs and the techniques available to query them. Another recent survey is [2] (2021) which analyzes graph streaming systems, where the differences and similarities between graph streaming systems and historical graph systems are explicitly given. In general, graph streaming systems tend to use snapshots as the stable (latest) instance of the graph, since it may be the case that recent updates have not been registered. However, in principle these snapshots may be stored and allow for historical queries as well. Some graph streaming systems explicitly - although it is controversial to what extent - support historical queries on such (small number) snapshots of their evolution.

Our contribution in this paper is twofold. First, we provide a concise but comprehensive discussion on the systems developed up to today after 2016 that is covered by the survey of [41]. Due to space limitations, we do not discuss extensively these systems. At the same time, we focus mainly on distributed systems making a simple reference to non-distributed ones. To the best of our knowledge, there is no other up-to-date comprehensive reference to such systems. Our second contribution, which required this state-of-the-art review, is the proposal of the high-level architecture of a distributed system for managing time-evolving graphs. The architecture is based on the ideas set by the authors in previous papers [23, 22, 38] as well as by the most recent developments in the area of historical graph management, as laid out in our small survey.

The rest of the paper is structured as follows. In Section 2 we provide a review of historical graph management systems after 2016. In section 3 we discuss the high-level architecture of the system we intend to implement for managing historical graphs. Finally, we conclude in Section 4.

2 A Review of Historical Graph Management Systems

Historical graphs have to utilize multiple dimensions resulting in many possible directions for such a system. Most systems are concerned with the storage and query of the evolution of the attributes as time evolves and some try to utilize the evolution of the topology for better partitioning or for reasons related to efficiency. In Table 2, we provide, without further discussion, some basic characteristics of non-distributed historical graph management systems. Some terminology is in order to understand the following tables:

1. **transaction time vs valid time:** Transaction time represents the time that an event takes place (i.e. the moment that a node is stored or deleted from a network) whereas valid time signifies the time period in which an object was valid (i.e. the time interval that a node existed in a database). In the transaction time setting updates can only occur in an append-like manner (i.e. an update in a field changes the value of the most recently stored value) whereas in the valid time setting updates can refer to any time point.
2. **time as a property vs snapshots:** in a rather crude manner, we get basically two different representations of time-evolving networks: a) snapshots, which correspond to a copy+log method; that is, the network is stored at specific time instances and in between a log is kept with the changes and b) time as a property, which correspond to incorporating the notion of time as another special property of the objects/properties within a network. There are many variations of these two basic representations.
3. **offline vs online vs streaming:** In an offline setting, we get all the history of the graph beforehand. In an online setting, the graph evolves and with it the database, while queries can be made at any time. In a streaming setting, we have an online setting with restrictions as to how much space and time is allowed for each update. In the literature, streaming is not usually related to historical information but more to computational restrictions on the processing of the stream due to its high velocity and massiveness. One can get as a by-product a rudimentary transaction time temporal graph processing system.
4. **Time-dependent and Time-independent algorithms:** If the algorithm on the temporal graph can be applied without time constraints then it is time-independent (e.g., pagerank computation at time instance t). If there are time constraints, then the algorithm is time-dependent (e.g., shortest path that respects time intervals on nodes/edges and the journey is time-consistent).

In Table 2 we show all distributed systems for historical graph management after the year 2016. Since our proposed system falls under this category we are going to discuss briefly some of these systems, which according to our opinion are quite important and have nice properties.

HINODE was the first pure vertex-centric system with respect to the storage model. It was introduced in [23] and supports valid time as well as extensions like

Summarizing the Characteristics of Non-Distributed Temporal Graph Management Systems			
Systems	Memory	Storage Model	Time-related characteristics
InteractionGraph [10]	Main Memory (old graph in disk)	Custom	Transaction time
STVG [28]	Main Memory	Neo4j	valid time, offline, restricted to transit networks
ASPEN [7]	In-Memory/parallel	extends Ligra	Streaming
GraphOne [24]	in-memory NVMe SSD	Custom	Streaming, can't get arbitrary historic views if transaction time is assumed
Auxo [12]	Main and External Memory	Custom	Transaction time
[3]	Main Memory	Custom	Transaction time, Snapshot-based, focus on space savings
[1]	Main Memory	Neo4j	Valid time, In addition to entity evolution it supports schema evolution
TGraph [15]	Main and External Memory	Neo4j	Support ACID Transactions, slow topological updates but fast property updates, Transaction time
VersionTraveller [18]	Main Memory	based on PowerGraph static graph management system	Offline Snapshot-based, Focus on switching between snapshots
NVGraph [27]	Non-Volatile Main Memory and DRAM	Custom	Online Snapshot-based, Transaction time

Table 1. Non-distributed systems for historical graph management.

Summarizing the Characteristics of Distributed Temporal Graph Management Systems		
Systems	Storage Model	Time-related characteristics
Portal [33]	Spark	Offline, time as a property, Valid time
GDBAlive [29]	Cassandra	Transaction time
Graphsurge [37]	Custom	offline snapshots, focus on differential computation across multiple snapshots
TEGRA [17]	Custom	Transaction time, based on persistent trees, incremental computation model, window analytics
GraphTau [16]	Spark	Streaming
Immortalgraph [30]	Custom	Transaction time, Snapshot-based, Focus on locality-aware (w.r.t. time and topology by replication) batch scheduling for computation
HGS [21]	Cassandra	Transaction Time, Sophisticated Snapshot-based
SystemG-MV [40]	IBMs SystemG	Relaxed transaction time
Raphtory [39]	Custom + Cassandra for archiving	Transaction time, streaming
Chronograph [5]	MongoDB	offline, time as a property, Focus on graph traversals
Graphite [9]	Apache Giraph	offline, Time-dependent and time-independent algs
Granite [34]	Based on Graphite	focus on temporal path queries, partition techniques to keep everything in main memory
Tink [26]	Apache Flink	Online, Valid time
Gradoop - TPGM [36, 6, 35]	Apache HBase/ Accumulo	Valid and Transaction time (bitemporal), Fully-fledged system ranging from a graph analytical language to the storage model
Greycat [14]	NoSQL Database + custom	Valid time, No edge attributes
PAST [8]	based on key/value stores (e.g., Cassandra)	Streaming Spatio-temporal graphs, bipartite graphs, only edges with time-points, spatiotemporal-specific query workloads
HINODE [23, 22, 38]	Custom (other versions are based on Cassandra and MongoDB)	Online, time as a property, Valid time (allows more general notions of time), pure vertex-centric storage model

Table 2. Distributed systems for historical graph management.

multiple universes. It was implemented within the G^* system [25] by replacing its storage subsystem. They showed gains in space usage, which is an immediate consequence of the pure vertex-centric approach. They supported local queries (e.g., 2-hop queries) as well as global queries (e.g., clustering coefficient). In addition, this vertex-centric model was also adapted for NoSQL databases by creating two models, SingleTable (ST) and MultipleTable (MT). In the former, all data fit in one table and a row has the data of a Diachronic Node, while in the latter, data are split in different tables. Two implementations were made, one in Cassandra [22] and later one in MongoDB [38] for comparison reasons, while in MongoDB we tried also to take advantage of indexes and iterative computation to reduce memory usage.

Portal In [33] they discuss about interval-based and point-based models preferring the interval-based model with sequenced semantics. As a data model, they talk about TGraph that uses the property graph model while they also discuss about sequenced semantics in a distributed environment (e.g partitioning, time-window operations). In PhD Thesis [32] they propose a Temporal Graph Algebra (TGA) and a temporal graph model (TGraph) supporting TGA In addition, in [31] they propose a declarative language (Portal) based on the previous model and built on top of a distributed system (Apache Spark). Portal has SQL-like syntax following SQL:2011 standard. They also discuss about possible algorithms on temporal graphs among which are node influence over time, graph centrality over time, communities over time, and spread of information. TGraph is a valid time model that extends the property graph model (each edge and vertex is associated with a period of validity), while all relations in Graph must meet 5 criteria (uniqueness of vertices/edges, referential integrity, coalesced, required property, constant edge association). TGA is both snapshot and extended snapshot reducible presenting a new primitive (resolve) while containing operators like trim, map, and aggregation. Portal uses Spark for in-memory representation and processing while it uses Apache Parquet for on-disk data layout using node files and edge files (but it doesn't support an index mechanism). They experimented with different in-memory representations, SnapshotGraph(SG) that stores the graph as individual snapshots, MultiGraph(MG) that stores one single graph by storing one vertex for all periods and one edge for every time period and OneGraph that stores each edge and vertex only once (also exists MGC and OGC). It has distributed locality like Immortalgraph, experimenting with different partitioning methods (the equi-depth partitioning is more efficient in most experiments) but stores materialized node/edges instead of deltas and they also experimented with both structural and temporal locality, concluding that temporal locality is more efficient (among other reasons due to the lack of sufficient discrimination in the temporal ranges of the datasets).

ImmortalGraph [30] is a parallel in-memory storage and computation system for multicore machines and distributed settings designed for historical graphs. It focuses more on locality optimizations, both in saving the data and in the execution of the queries using locality-aware batch scheduling (LABS). They make a clear distinction and a very nice discussion between the time-centric layout and the structure-centric layout. It supports parallel temporal graph mining using iterative computations while they prefer those computations to be in memory. ImmortalGraph supports both global and local queries at a point in time or a time window. Data are stored in snapshot groups with the use either of edge files or vertex files, depending on the application. A snapshot group organizes together snapshots of a time interval by storing the first one and the changes that happened to the rest. This can be stored either with the use of time locality by grouping activities associated with a vertex (and a vertex index) or with the use of structure locality by storing together neighboring vertex (and a time index). Instead of choosing between the possible trade-off from structure and time locality, they replicate the needed data and decide which technique to use according to the type of query and how far is the starting point from the start of the snapshot group. LABS favors partition-parallelism from snapshot-parallelism, so they prefer batch operations of vertex/edges achieving better locality and less inter-core communication. They also experimented with iterative graph mining and iterative computations. In the former they reconstruct the needed snapshots in memory favoring time locality (and they compare both push, pull, and stream techniques), while in the latter they compute the first snapshot and the later $N-1$ snapshots in batch (achieving better locality). They also implemented both low-level and high-level query interfaces, the latter used for iterative computations. An earlier implementation of ImmortalGraph is Chronos [13] with the main difference being that it only focuses on time locality. Finally, they provide a low-level as well as a high-level programming interface (APIs) that in fact define their analytics engine. They also experiment on Pagerank, diameter, SSSP, connected components, maximal independent sets, and sparse-matrix vector multiplication.

Historical Graph Store (HGS) [21] is a cloud parallel node-centric distributed system for managing and analyzing historical graphs. HGS consists of two major components, Temporal Graph Index (TGI) that manages the storage of the graph in a distributed Cassandra environment, and Temporal Graph Analysis Framework (TAF) that is a spark-based library for analyzing the graph in a cluster environment. TGI combines Partitioned Eventlists, which stores atomic changes, with Derived Partitioned Snapshots, which is a tree structure where each parent is the intersection of children deltas (used for better structure locality storing neighborhoods), both of them are partitioned, while they are also combined with Version Chain to maintain pointers to all references of nodes in chronological order. TGI divides the graph into time spans (like snapshot groups of ImmortalGraph) with micro-deltas which are stored as key-value pairs contiguously into horizontal partitions at every time span. In that way, it can execute in parallel every query to many Query Processors and aggregate the result to Query Manager or to client. It can work both on hash-based and locality-aware partitioning by projecting a time range (time-span) of the graph in a static graph. TAF supports both point in time queries and time-window, some of the supported queries are subgraph retrieval with filtering, aggregations, pattern matching, and queries about the evolution of the graph. An earlier implementation of TGI is DeltaGraph [20] which focuses on snapshot retrieval

ChronoGraph [5] is a temporal property graph database built by extending Tinkerpop and its graph traversal language Gremlin so as to support temporal queries. It stores the temporal graph in persistent storage (MongoDB), and then loads the graph in-memory and traverses it. Their innovation is not in the storage model but in how they support traversal queries efficiently on top of it. It exploits parallelism, the temporal support of Tinkerpop to increase efficiency, and lazy evaluations to reduce memory footprints of traversals. Its main focus is on temporal graph traversals but can also return snapshots of the graph. They distinguish point-based events and period-based events because of their semantics and their architectural needs. They use aggregation to convert point-based events to period-based events so as not to have two different semantics in order to improve time efficiency in query execution. They achieve this by using a threshold as the max time interval that might exist between time instants so as to group them. A graph is composed of a static graph, a time-instant property graph, and a time-period property graph. They also use event logic, where an event might be either a vertex or an edge, on a period or a time instant. They applied temporal implementation of BFS, SSSP, and DFS, while they don't recommend DFS on their system because of Gremlins recursive logic. One more thing they discuss is that when you store the temporal graph in snapshots there will be some loss of information because a snapshot might contain data of an hour, day e.t.c according to the needs of the problem, while when you store them using time interval, you have a more accurate representation of the graph. An extension of Chronograph by using time-centric computation for traversals is given in [4].

Tink [26] is an open-source parallel distributed temporal graph analytics library built on top of the Dataset API of Apache Flink and uses Gelly as a language. It extends the temporal property graph-model focusing on keeping intervals instead of time-points by saving nodes as tuples. It depends on Flink to use parallelism, optimizations, fault tolerance, and lazy-loading and supports iterative processing. It also uses functions from Flink like filtering, mapping, joining, and grouping. Most algorithms use Gelly's Signal/Collect (scatter-gather) model which executes computations in a vertex-centric way. It also provides temporal analytics metrics and algorithms. For the latter, they implemented shortest path earliest arrival time and shortest path fastest path while for temporal metrics they provide temporal betweenness and temporal closeness.

Gradoop (TPGM) TPGM [36] [6] [35] is an extension of Gradoop's EPGM model (model for static graph processing, presented in a series of papers from 2015, e.g., see [19]) to support temporal analytics on evolving property graphs (or collection of graphs) that can be used through Java API or with KNIME. Gradoop is an open-source parallel distributed dataflow framework that runs on shared-nothing clusters and uses GRALA as a declarative analytical language and Temporal-GDL as a query language. Gradoop supports Apache HBase, and Apache Accumulo to provide storage capabilities on top of HDFS, while other databases can also be used with some extra work. TPGM supports bitemporal time by adding to vertex, and edges as well as to graph the logical attributes for start and end time for both valid and transaction time (but it allows to not use some of them). While TPGM provides an abstraction, Apache Flink is used for handling the execution process in a lazy way and it provides several libraries. GRALA provides operators both for single graphs and graph collections, it supports retrieval of snapshots, transformations of attributes or properties, subgraph extraction, the difference of two snapshots, time-dependent graph grouping, temporal pattern matching, and others. For some more complex algorithms, it also supports iterative execution using Apache Flink's Gelly library.

Lastly, they have implemented a set of operations for their analytics engine and have implemented them in Flink - by using Flink Gelly. For further investigation, it should be mentioned that they provide an extensive description of their architecture while they also provide a *Lessons Learned* section that contains valuable information with respect to their design choices.

SystemG-MV In [40] they propose an OLTP-oriented distributed temporal property graph database (dynamically evolving temporal graphs). It is built on top of IBM's SystemG, which is a distributed graph database using LMDB (B-tree based key-value store). Data are stored in tables with key/value pairs allowing to query part of the graph efficiently without retrieving whole snapshots. Different tables exist for vertices, edges, and properties, while it supports updates only on present/future timestamps like transaction-time models. Therefore, changing previous values of the graph is not allowed explicitly, but it is possible to change past events by using low-level methods. In this model, they save two timestamps for the creation/deletion of vertices/edges but while they don't allow edges to be recreated with the same id, although multiple edges can exist between a pair of vertices. For vertices, they keep the deleted vertices in a different table, while for properties they keep it simplified by keeping only one timestamp for the update as the rest can be calculated. Alongside the historic tables, they keep one table with the current state of the graph for more efficient queries.

GraphOne [24] is an in-memory data store with a durability guarantee on external non-volatile memory NVMe SSD, while it was solely implemented in C++. Its objective is to be able to perform both real-time analytics or diverse data access while synchronous updates are applied to the database. To achieve that, GraphOne uses a hybrid model which is composed of a circular edge log and an adjacency store. The adjacency store has a multi-versioned degree array and an adjacency list with chained edges, which is used to permanently store the data taking regard to snapshots. On the other hand, edge log is used to temporary store the incoming data as edges so as to later move them in parallel to the adjacency store and improve the ingestion time. In brief, an epoch in GraphOne is consisted of 4 stages logging, archiving, durable, and compaction. At logging phases, records are inserted in the edge log at their arrival order, when the inserted edges reach the archiving threshold the multi-threaded archiving phase starts in parallel with the logging phase. At the start of the archiving phase, it shards non-archived edges to multiple local buffers so as to keep the data ordering intact, then the edges are being archived in parallel to the adjacency store, while also new degree nodes are allocated. In short, in the durable phase data are being appended to a file, while in the compaction phase deleted data are being removed. One thing that needs to be noticed is that GraphOne despite that is designed to store evolving graphs, it is not designed for getting arbitrary historic views from the adjacency store.

TEGRA [17] is a distributed system with a compact in-memory representation (using their own storage model) both for graph and intermediate state. Its main focus is on time window analytics for historical graphs, but it can also be used for live analytics as the data are ingested in the database. An interesting feature is the ICE computational model that takes advantage of the intermediate state of computations saving it, so as to use it in the same or different queries. Computations are being made only in subgraphs affected by updates at each iteration. This has some overhead on finding the correct state and also the extra entities that should be included in the query when there is large number of updates at each iteration or while trying to use ICE on different queries. Tegra also uses TimeLapse, an API for high-level abstraction which also allows what-if questions that change the graph creating different histories, suited for data analytics purposes. The storage model behind TEGRA is DGSI, which uses persistent data structures to maintain previous versions of data when modified. It uses persistent adaptive radix trees to store edges and nodes separately with path copying. It uses simple partitioning strategies to distribute the graph to nodes. Each node has two pART for nodes and edges respectively. Log files are being used to store updates between snapshots, which are stored in turn in the two pARTs. The branch and commit primitives are really interesting as well as the GAS (Gather - Apply - Scatter) model [11]. It allows also changing any version thus leading to a branched history (like a tree - full persistence). Lastly, TEGRA also uses an LRU policy to periodically remove versions that have not been accessed for a long time.

STVG [28] is a prototype framework that focuses on fast-evolving graphs. It is built on top of Neo4j and supports both point and time-window queries while its main use is to analyze evolutionary transit networks. It is based on the whole-graph model for representing the graph, which is composed of subgraphs that facilitate the conceptual modeling of the connectivity between entities and the time-graph of Neo4j that is responsible for keeping track of time evolution. Subgraphs are connected to the time-graph to keep track of the evolution of the whole-graph, while nodes belonging to different subgraphs are linked with complementary connectivity edges. Since this framework is used for evolutionary transit networks it is demanded that the graph needs to be connected while edges can't recur over time. Projected graphs are used to materialize and retrieve the graph both at a time-window or a sliding window. They have implemented also graph metrics used to analyze a transit network, graph density, network diameter, and average path length having in mind their specific application. In general, this framework has some good ideas but it is tailored for transit networks.

Graphite [9] is a distributed system for managing historical graphs (fully evolved and using valid time) by using an interval-centric computing model (ICM) built over Apache Giraph. They assume data are given in ascending time order and any vertex can exist only once for a contiguous time-interval. It also has the ability to execute both time-independent and time-dependant historical queries (temporal queries on a time-window), while they tried to create a unifying abstraction that scales to both and ease algorithm design and detach user logic using ICM and time-warp operator. ICM uses Bulk Synchronous Parallel (BSP) execution for every active vertex of a query until it converges. They use two stages of logic, compute and scatter, where compute does the computations needed for a vertex, and scatter transfers it with messages to neighbor vertexes as needed. Time-warp operator happens at the alternating compute scatter steps to help sharing of calls and messages across intervals. A key aspect of it, is that it groups input guaranteeing correctness of grouping and no duplication, while it returns as minimal as possible triples. They also designed and constructed a plethora of time independent (TI) and time dependent (TD) algorithms for their system. with a very detailed evaluation

Granite [34] is a distributed engine for storing and analyzing temporal property graphs (supports temporal path queries) made on top of and as a sequel to Graphite focusing on path queries. It is made an assumption for infrequent updates and frequent queries. They extend the previous model by adding a temporal aggregation operator, indexing, query planning and optimization, while they prefer to relax ICM so as to work beyond time respecting algorithms. Granite handles both static temporal graphs and dynamic temporal graphs while it uses interval-centric features only in the latter. An interesting point is that to optimize path queries they split them and execute them concurrently, while they also keep statistics about the active nodes at each time point so as to optimize the query planning. While Graphite makes hash partitioning at query execution, Granite first partitions every entity according to its type and later it performs a topological partition to its independent group of entities of the same type and splits them into workers using the round-robin technique. They also use a result tree so as not to send duplicate paths across the system (some parts of the path might be the same). Lastly, they propose a query language for path queries.

NVGRAPH A rather interesting system from a hardware perspective. NVGRAPH [27] is an in-memory data structure focused on exploiting the different advantages of NVMM and DRAM, combining them into a C++ library implementation. The major issue they try to tackle in NVMM is providing crash consistency while they argue that simply using NVMM without considering its issues is a sub-optimal solution. They focus on creating an architecture that uses both DRAM and NVMM to hide the issues of NVMM while they are exploiting its advantages. NVGraph stores the graph as a series of continuous snapshots by storing the first snapshot and deltas for the next snapshots. They also implemented 4 algorithms for evaluation Pagerank, BFS, influence maximization, and rumor source detection.

3 Architecture of MAGMA

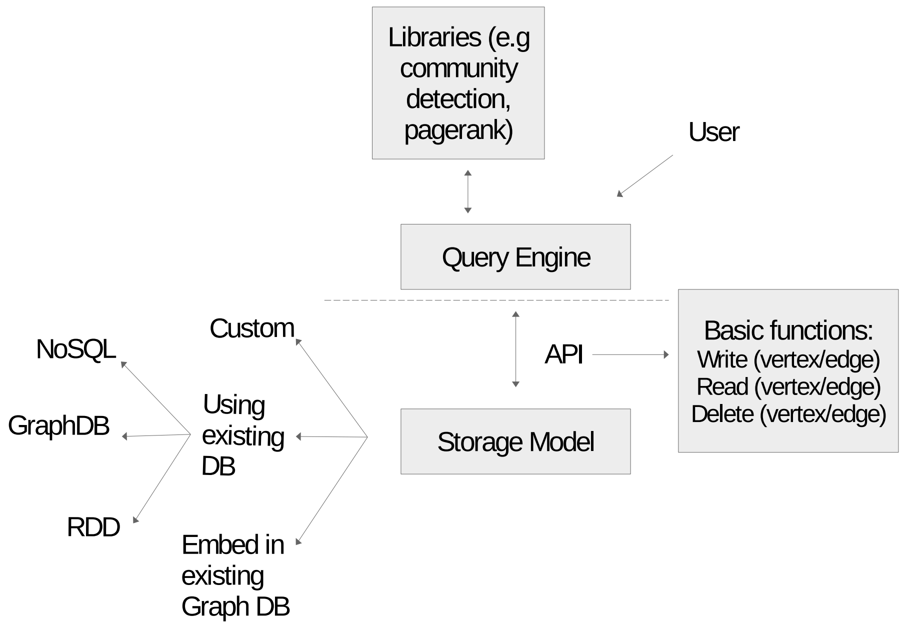
Libraries (e.g community detection, pagerank)
User
Query Engine
API
Basic functions: Write (vertex/edge) Read (vertex/edge) Delete (vertex/edge)
Storage Model
Custom
Using existing DB
Embed in existing Graph DB
NoSQL
GraphDB
RDD

Fig. 1. A view of MAGMA with the possible storage directions.

In this section, we describe the general characteristics of the proposed historical graph management and processing system (MAGMA), the possible directions we could take implementing it as well as the possible obstacles we need to overcome.

An immediate observation from the previous systems is that each one of them focuses on different aspects of historical graph management, resulting in a different appropriate solution for each application. This is because the management and processing of historical graphs span multiple design dimensions forbidding the existence of one system to rule them all. Our approach is towards creating a purely vertex-centric and storage optimal (asymptotically) distributed system called MAGMA with the ability to update/query efficiently the history and apply graph algorithms on arbitrary time periods rather than on specified snapshots. Following HiNode, MAGMA will be more efficient in local than global queries due to its vertex-centric structure. However, we also wish to efficiently execute global queries (e.g., pagerank) by exploiting our vertex-centric architecture and implementing modern techniques (e.g., thinking like a vertex) for efficient and effective parallel computation. Another important aspect that needs to be addressed in a later stage of the development of MAGMA, is the system's API. In particular, we need to design the system in a way that guarantees its simplicity with respect to use, its efficiency, its scalability, its flexibility with respect to its functionality, and its compatibility with existing libraries (for static or temporal graphs).

The key part of the system is the efficient and effective vertex-centric storage of the graph. A diachronic node contains the whole history of a particular node in the sense that it stores all changes and their time intervals related to this node, such as a change in an incoming edge or a change in a property of the node. To this end, we employ three fundamental operations in order to update and query the diachronic nodes: write, read and delete. All three operations are applied on diachronic nodes that contain all relevant information (edges, properties, etc.). More complex updates and query operations can be built on these fundamental operations that will serve mainly the online management and processing of the historical graph.

Regarding the storage model, we have narrowed our options into either creating a custom database for storing the historical graph into servers or by extending an existing database and applying our model to them. In any case, we will always stick to the pure vertex-centric approach proposed in HiNode and adapt it appropriately to fit the design choice of the storage model. In the case of creating a custom database, we have complete freedom with respect to designing the storage model to fit HiNode, but on the other hand, it will require considerably more effort for the implementation as well as to ensure compatibility with existing libraries. On the other hand one could use an existing database, either a NoSQL database like Cassandra and MongoDB or a Graph database (e.g., GraphX and SystemG). In this case, it is easier to build the system and take advantage of the optimizations and functionality that already exist within this database (e.g., fault-tolerance and partitioning), but there is less freedom in applying the storage model of HiNode. Another option, in this case, is to extend an existing graph database (e.g., GraphX) to support natively the management and processing of historical graphs based on a pure vertex-centric approach. This is a harder task, but it has the merit of sharing and using existing libraries within this particular graph database. In addition, the visibility of such a solution will be much higher across the community.

Since MAGMA is a distributed system, the partitioning strategy is of paramount importance for the efficiency of the system. Most systems use either a simple hash-based partition or a chronological or topological partitioning. In our case, the topological partitioning is more natural but we also need to take into account the temporal evolution of the graph. In topological partitioning, we want to place in the same machine, nodes that are connected or that are relatively close to each other. One problem we might encounter with topological partitioning is that in different timestamps, the distance between nodes changes, and as a result, different partitions may be more appropriate in different time instances. This is problematic in our case since a diachronic node contains all the history of the node and thus naturally all history is stored in a single machine. Two possible solutions for this issue are either by using different metrics for partitioning combining the whole history of the graph or by dividing parts of a node to different machines. Another possible solution, which could also be combined with the previous one, is the duplication of some nodes across machines. However, in this case, care should be taken with respect to space usage.

Another critical part of the system is the query engine and the libraries that will be available. Regarding the libraries, we intend to implement algorithms on temporal graphs like temporal shortest path (journeys) or community detection and evolution while also supporting algorithms for static snapshots. This can be achieved either by using the abstraction provided from the API or by exploiting the system's architecture and creating them from scratch. For the former task, we first want to create a query engine able to handle more demanding tasks that supports parallelism and provides the user with an easy-to-use API. To do so, our processing unit needs to apply one of the following approaches: "thinking like an edge" (TLAE), "thinking like a vertex" (TLEV), "thinking like a neighborhood" (TLAN), "thinking like a subgraph" (TLAS) or "thinking like an interval" (TLAI). We need to further investigate these approaches and decide which one would be more efficient in our system, although we can deduce straightforwardly that some of these will probably not fit our vertex-centric architecture. On the other hand, TLEV techniques seem as the most promising at the moment, in order to take advantage of Hinode's vertex-centric structure, while TLAN or TLAS approaches could also fit our model depending on the partition strategy used. At a later stage, these approaches will be used for iterative computations.

4 Conclusions

In this paper, we provide a small review of contemporary historical graph management systems and propose an architecture for such a system based on our previous research work. We intend to extend the very preliminary results contained in this paper as follows: 1) A survey on systems for historical graph management. This survey will cover all historical graph management systems and will provide researchers as well as developers information as to the pros and cons of these systems in order to help them choose correctly. 2) The development of a system (called MAGMA) for managing and processing historical graphs. The high-level architecture of this system and basic options for its implementation are described in this paper.

References

1. Andriamampianina, L., Ravat, F., Song, J., Vallès-Parlangeau, N.: A generic modelling to capture the temporal evolution in graphs. In: 16e journées EDA : Business Intelligence & Big Data (EDA 2020). vol. RNTI-B-16, pp. 19–32. Lyon, France (Aug 2020), https://hal.archives-ouvertes.fr/hal-03109670
2. Besta, M., Fischer, M., Kalavri, V., Kapralov, M., Hoefler, T.: Practice of streaming processing of dynamic graphs: Concepts, models, and systems (2021)
3. Bok, K., Kim, G., Lim, J., Yoo, J.: Historical graph management in dynamic environments. Electronics **9**(6) (2020). https://doi.org/10.3390/electronics9060895
4. Byun, J.: Enabling time-centric computation for efficient temporal graph traversals from multiple sources. IEEE Transactions on Knowledge and Data Engineering pp. 1–1 (2020). https://doi.org/10.1109/TKDE.2020.3005672
5. Byun, J., Woo, S., Kim, D.: Chronograph: Enabling temporal graph traversals for efficient information diffusion analysis over time. IEEE Transactions on Knowledge and Data Engineering **32**(3), 424–437 (2020). https://doi.org/10.1109/TKDE.2019.2891565
6. Christ, L., Gomez, K., Rahm, E., Peukert, E.: Distributed graph pattern matching on evolving graphs (2020)
7. Dhulipala, L., Blelloch, G.E., Shun, J.: Low-latency graph streaming using compressed purely-functional trees. In: Proceedings of the 40th ACM SIGPLAN Conference on Programming Language Design and Implementation. p. 918–934. PLDI 2019, Association for Computing Machinery, New York, NY, USA (2019)
8. Ding, M., Yang, M., Chen, S.: Storing and querying large-scale spatio-temporal graphs with high-throughput edge insertions. arXiv preprint arXiv:1904.09610 (2019)
9. Gandhi, S., Simmhan, Y.: An interval-centric model for distributed computing over temporal graphs. In: 2020 IEEE 36th International Conference on Data Engineering (ICDE). pp. 1129–1140 (2020). https://doi.org/10.1109/ICDE48307.2020.00102
10. Gedik, B., Bordawekar, R.: Disk-based management of interaction graphs. IEEE Transactions on Knowledge & Data Engineering **26**(11), 2689–2702 (nov 2014). https://doi.org/10.1109/TKDE.2013.2297930
11. Gonzalez, J.E., Low, Y., Gu, H., Bickson, D., Guestrin, C.: Powergraph: Distributed graph-parallel computation on natural graphs. p. 17–30. OSDI'12, USENIX Association (2012)
12. Han, W., Li, K., Chen, S., Chen, W.: Auxo: a temporal graph management system. Big Data Mining and Analytics **2**(1), 58–71 (2019). https://doi.org/10.26599/BDMA.2018.9020030
13. Han, W., Miao, Y., Li, K., Wu, M., Yang, F., Zhou, L., Prabhakaran, V., Chen, W., Chen, E.: Chronos: A graph engine for temporal graph analysis. In: Proceedings of the Ninth European Conference on Computer Systems. EuroSys '14, Association for Computing Machinery, New York, NY, USA (2014). https://doi.org/10.1145/2592798.2592799, https://doi.org/10.1145/2592798.2592799
14. Hartmann, T., Fouquet, F., Jimenez, M., Rouvoy, R., Le Traon, Y.: Analyzing complex data in motion at scale with temporal graphs (07 2017). https://doi.org/10.18293/SEKE2017-048
15. Huang, H., Song, J., Lin, X., Ma, S., Huai, J.: Tgraph: A temporal graph data management system. In: Proceedings of the 25th ACM

International on Conference on Information and Knowledge Management. p. 2469–2472. CIKM '16, Association for Computing Machinery, New York, NY, USA (2016). https://doi.org/10.1145/2983323.2983335, https://doi.org/10.1145/2983323.2983335
16. Iyer, A.P., Li, L.E., Das, T., Stoica, I.: Time-evolving graph processing at scale. In: Proceedings of the fourth international workshop on graph data management experiences and systems. pp. 1–6 (2016)
17. Iyer, A.P., Pu, Q., Patel, K., Gonzalez, J.E., Stoica, I.: TEGRA: Efficient ad-hoc analytics on evolving graphs. In: 18th USENIX Symposium on Networked Systems Design and Implementation (NSDI 21). pp. 337–355. USENIX Association (Apr 2021), https://www.usenix.org/conference/nsdi21/presentation/iyer
18. Ju, X., Williams, D., Jamjoom, H., Shin, K.G.: Version traveler: Fast and memory-efficient version switching in graph processing systems. In: 2016 USENIX Annual Technical Conference (USENIX-ATC 16). pp. 523–536 (2016)
19. Junghanns, M., Petermann, A., Teichmann, N., Gómez, K., Rahm, E.: Analyzing extended property graphs with apache flink. In: Proceedings of the 1st ACM SIGMOD Workshop on Network Data Analytics. NDA '16, Association for Computing Machinery, New York, NY, USA (2016). https://doi.org/10.1145/2980523.2980527, https://doi.org/10.1145/2980523.2980527
20. Khurana, U., Deshpande, A.: Efficient snapshot retrieval over historical graph data. In: 2013 IEEE 29th International Conference on Data Engineering (ICDE). pp. 997–1008 (2013). https://doi.org/10.1109/ICDE.2013.6544892
21. Khurana, U., Deshpande, A.: Storing and analyzing historical graph data at scale. In: Pitoura, E., Maabout, S., Koutrika, G., Marian, A., Tanca, L., Manolescu, I., Stefanidis, K. (eds.) Proceedings of the 19th International Conference on Extending Database Technology, EDBT 2016, Bordeaux, France, March 15-16, 2016, Bordeaux, France, March 15-16, 2016. pp. 65–76. OpenProceedings.org (2016). https://doi.org/10.5441/002/edbt.2016.09, https://doi.org/10.5441/002/edbt.2016.09
22. Kosmatopoulos, A., Gounaris, A., Tsichlas, K.: Hinode: implementing a vertex-centric modelling approach to maintaining historical graph data. Computing **101**(12), 1885–1908 (2019). https://doi.org/10.1007/s00607-019-00715-6, https://doi.org/10.1007/s00607-019-00715-6
23. Kosmatopoulos, A., Tsichlas, K., Gounaris, A., Sioutas, S., Pitoura, E.: Hinode: an asymptotically space-optimal storage model for historical queries on graphs. Distributed Parallel Databases **35**(3-4), 249–285 (2017). https://doi.org/10.1007/s10619-017-7207-z, https://doi.org/10.1007/s10619-017-7207-z
24. Kumar, P., Huang, H.H.: G¡span class="smallcaps smallercapital"¿raph¡/span¿o¡span class="smallcaps smallercapital"¿ne¡/span¿: A data store for real-time analytics on evolving graphs. ACM Trans. Storage **15**(4) (Jan 2020). https://doi.org/10.1145/3364180, https://doi.org/10.1145/3364180
25. Labouseur, A.G., Birnbaum, J., Olsen, P.W., Spillane, S.R., Vijayan, J., Hwang, J., Han, W.: The g* graph database: efficiently managing large distributed dynamic graphs. Distributed Parallel Databases **33**(4), 479–514 (2015). https://doi.org/10.1007/s10619-014-7140-3, https://doi.org/10.1007/s10619-014-7140-3
26. Lightenberg, W., Pei, Y., Fletcher, G., Pechenizkiy, M.: Tink: A temporal graph analytics library for apache flink. In: Companion Proceedings of the The Web Conference 2018. pp. 71–72 (2018)

27. Lim, S., Coy, T., Lu, Z., Ren, B., Zhang, X.: Nvgraph: Enforcing crash consistency of evolving network analytics in nvmm systems. IEEE Transactions on Parallel and Distributed Systems **31**(6), 1255–1269 (2020). https://doi.org/10.1109/TPDS.2020.2965452
28. Maduako, I., Wachowicz, M., Hanson, T.: Stvg: an evolutionary graph framework for analyzing fast-evolving networks. Journal of Big Data **6**(1), 1–24 (2019)
29. Massri, M., Raipin Parvedy, P., Meye, P.: Gdbalive: a temporal graph database built on top of a columnar data store. Journal of Advances in Information Technology **12** (09 2020). https://doi.org/10.12720/jait.12.3.169-178
30. Miao, Y., Han, W., Li, K., Wu, M., Yang, F., Zhou, L., Prabhakaran, V., Chen, E., Chen, W.: Immortalgraph: A system for storage and analysis of temporal graphs. ACM Trans. Storage **11**(3) (jul 2015). https://doi.org/10.1145/2700302, https://doi.org/10.1145/2700302
31. Moffitt, V., Stoyanovich, J.: Portal: A query language for evolving graphs (02 2016)
32. Moffitt, V.Z.: Framework for Querying and Analysis of Evolving Graphs. Ph.D. thesis (07 2017). https://doi.org/10.13140/RG.2.2.16079.64166, https://www.proquest.com/docview/1946186055?pq-origsite=gscholar&fromopenview=true
33. Moffitt, V.Z., Stoyanovich, J.: Towards sequenced semantics for evolving graphs. In: EDBT. pp. 446–449 (2017)
34. Ramesh, S., Baranawal, A., Simmhan, Y.: Granite: A distributed engine for scalable path queries over temporal property graphs. Journal of Parallel and Distributed Computing **151**, 94–111 (2021). https://doi.org/https://doi.org/10.1016/j.jpdc.2021.02.004, https://www.sciencedirect.com/science/article/pii/S0743731521000253
35. Rost, C., Gomez, K., Täschner, M., Fritzsche, P., Schons, L., Christ, L., Adameit, T., Junghanns, M., Rahm, E.: Distributed temporal graph analytics with gradoop. The VLDB Journal (May 2021). https://doi.org/10./s00778-021-00667-4, https://doi.org/10.1007/s00778-021-00667-4
36. Rost, C., Thor, A., Rahm, E.: Analyzing temporal graphs with gradoop. Datenbank-Spektrum **19**(3), 199–208 (2019)
37. Sahu, S., Salihoglu, S.: Graphsurge: Graph analytics on view collections using differential computation. In: Proceedings of the 2021 International Conference on Management of Data. pp. 1518–1530 (2021)
38. Spitalas, A., Gounaris, A., Tsichlas, K., Kosmatopoulos, A.: Investigation of database models for evolving graphs. In: Combi, C., Eder, J., Reynolds, M. (eds.) 28th International Symposium on Temporal Representation and Reasoning, TIME 2021, September 27-29, 2021, Klagenfurt, Austria. LIPIcs, vol. 206, pp. 6:1–6:13. Schloss Dagstuhl - Leibniz-Zentrum für Informatik (2021). https://doi.org/10.4230/LIPIcs.TIME.2021.6, https://doi.org/10.4230/LIPIcs.TIME.2021.6
39. Steer, B., Cuadrado, F., Clegg, R.: Raphtory: Streaming analysis of distributed temporal graphs. Future Generation Computer Systems **102**, 453–464 (2020). https://doi.org/https://doi.org/10.1016/j.future.2019.08.022, https://www.sciencedirect.com/science/article/pii/S0167739X19301621
40. Vijitbenjaronk, W.D., Lee, J., Suzumura, T., Tanase, G.: Scalable time-versioning support for property graph databases. In: 2017 IEEE International Conference on Big Data (Big Data). pp. 1580–1589 (2017). https://doi.org/10.1109/BigData.2017.8258092

41. Zaki, A., Attia, M., Hegazy, D., Amin, S.: Comprehensive survey on dynamic graph models. International Journal of Advanced Computer Science and Applications **7**(2), 573–582 (2016)

This research was supported by the Hellenic Foundation for Research and Innovation (H.F.R.I.) under the “2nd Call for H.F.R.I. Research Projects to support Faculty Members & Researchers” (Project Number: 3480).