Position Paper: Bitemporal Dynamic Graph Analytics

Hassan Halawa
University of British Columbia
hhalawa@ece.ubc.ca

Matei Ripeanu
University of British Columbia
matei@ece.ubc.ca

ABSTRACT

Most of today's graph analytics systems model *static* graphs and do not support business use cases that require the ability to: *(i)* query the *dynamic* graph data for a time-evolving system, *(ii)* carry out investigations on its historical evolution, and *(iii)* audit past business decisions made with potentially stale or incorrect data. This position paper presents our vision for *bi-temporal dynamic graph analytics*, and sketches a design for a system that efficiently supports these requirements.

ACM Reference Format:
Hassan Halawa and Matei Ripeanu. 2021. Position Paper: Bitemporal Dynamic Graph Analytics. In *4th Joint International Workshop on Graph Data Management Experiences & Systems (GRADES) and Network Data Analytics (NDA) (GRADES-NDA'21), June 20–25, 2021, Virtual Event, China.* ACM, New York, NY, USA, 12 pages. https://doi.org/10.1145/3461837.3464514

1 INTRODUCTION

The body of work related to graph analytics is prodigious [1, 2, 6, 17, 18, 23, 34, 35, 38, 53, 56–61, 64, 67, 70, 72, 77–83, 87, 89–97]. A part of the design space, however, has been scarcely explored: systems that are able to both accurately model a graph's evolution over time and support current state, historical, and audit queries. We contend that a multitude of real-world usage scenarios drive the need to design systems that cover this space, and discuss a few of them in Section 2.

As the real systems modeled by graphs continuously evolve with time, a few factors introduce a large amount of complexity: *(i)* the need to support queries over dynamic data, *(ii)* the need to explicitly model the temporal evolution of the system (i.e., the graph topology and vertex/edge properties), *(iii)* the fact that the information pertaining to the evolution of the system may arrive out of order and/or with arbitrary delays, and *(iv)* the need to support complex business use cases (e.g., auditing).

The relatively few existing systems designed specifically to model dynamic and/or temporal graphs do not appropriately address the aforementioned requirements (see Sections 3 and 7). In particular, such systems, do not efficiently support the business use cases that require the ability to: *(i)* query the *dynamic* graph data being collected, *(ii)* carry out investigations on historical data, and *(iii)* audit past business decisions made with potentially stale or incorrect data.

Permission to make digital or hard copies of all or part of this work for personal or classroom use is granted without fee provided that copies are not made or distributed for profit or commercial advantage and that copies bear this notice and the full citation on the first page. Copyrights for components of this work owned by others than the author(s) must be honored. Abstracting with credit is permitted. To copy otherwise, or republish, to post on servers or to redistribute to lists, requires prior specific permission and/or a fee. Request permissions from permissions@acm.org.
GRADES-NDA'21, June 20–25, 2021, Virtual Event, China
© 2021 Copyright held by the owner/author(s). Publication rights licensed to ACM.
ACM ISBN 978-1-4503-8477-3/21/06…$15.00
https://doi.org/10.1145/3461837.3464514

Design Goals. Our goal is to explore the feasibility of, and sketch a design for, a graph analytics system that supports the above three functionalities. At its core, our design employs *bi-temporal* modeling [43] - the system explicitly models all events along two time axes: *(i)* the time graph events happen (referred to as the *valid* time), and *(ii)* the time the system learns about the events (referred to as the *transaction* time).

When facing out of order and arbitrarily delayed events, explicit bi-temporal modeling enables several capabilities which - jointly - can not be supported by systems that are not bi-temporal (Section 3). These capabilities include the ability to query: *(i)* the current state given what the system knows at the time of the query, *(ii)* any past state given what the system knows at the time of the query (i.e., historical queries), and *(iii)* any past state given what the system knew at some arbitrary point in the past (i.e., audit queries). We stress that we aim to serve all these queries online, that is while the system ingests new data, and without pausing the system for pre-processing.

Challenges. A system targeting dynamic graphs with explicit support for bi-temporal modeling must be able to find a good balance when addressing several challenges: *(i)* space efficiency given that any ingested events must be kept indefinitely (i.e., for audit purposes and historical investigations, data can not be deleted or overwritten), *(ii)* ingestion performance given that the modeled system may be highly dynamic (i.e., numerous independent data sources, high ingest rates), *(iii)* query performance given the need to support the aforementioned diverse query types online (i.e., current state, historical, and audit), and *(iv)* maintaining consistency given that events may frequently arrive out of order and with potentially arbitrary delays.

Core Data Structures. Some of these challenges are addressed by a judicious choice of supporting data structures (Section 5). We observe that only two core data structures are sufficient to support a bi-temporal graph: a *bi-temporal value* to model the evolution of a specific attribute of an existing entity (e.g., vertex, or edge), and a *bi-temporal set* to model the evolution of a set of entities (e.g., a graph's vertices, or a vertex outward edges) over time. For the bi-temporal set data structure we find inspiration in a fundamental problem in computational geometry [20] (i.e., Segment Stabbing). To support efficient historical investigations and auditing we implement both data structures as *persistent*[1] [22]. This: *(i)* provides the required properties (i.e., immutability, copy-on-write), *(ii)* enables efficient access to all previous versions of the data structures (i.e., efficient historical querying), and *(iii)* supports concurrent queries without requiring the use of expensive coarse-grained locking mechanisms (i.e., a global mutex).

[1] In contrast to *persistent* data structures, ordinary data structures are *ephemeral*: changes to the data structure destroy the old version leaving only the new one. Also, note that *persistent* data structures can be made *durable*, that is, saved to disk or some other form of non-volatile storage to recover from crashes or power failures.

Contributions. In this position paper we: *(i)* make the case for our proposed bi-temporal dynamic graph analytics system and discuss real-world application use case scenarios (Section 2), *(ii)* outline our vision for a graph analytics system that explicitly models the evolution through time of dynamic graphs (Section 3), *(iii)* discuss its feasibility as well as the potential design choices that can be made in this space (Sections 4, and 5), and *(iv)* sketch the supporting data structures (i.e., bi-temporal values, and sets) and show how these can be used to synthesize the system's underlying bi-temporal dynamic graph data structure (Section 5). *To the best of our knowledge, we are the first*[2] *to sketch a design for a truly bitemporal dynamic graph analytics system.*

2 USE CASE SCENARIOS

Key Shared Properties. Before presenting use-cases we summarize the generic application-level properties that drive the need for the system we propose. A good match for our system are applications that:

(i) model a time-evolving real-wold system as a graph of interlinked entities,
(ii) require interactive use while ingesting new events (i.e., the ability to support 'online' queries),
(iii) require explicitly modeling the temporal evolution of the data (e.g., as mandated by regulators, or to support historical investigation),
(iv) operate in environments where the information pertaining to the evolution of the system may arrive with arbitrary delays and/or out of order, and
(v) require, in addition to basic current state and historical queries, support for more complex business use cases (e.g. forensics and audit queries).

The use-cases presented below follow a presentation structure that roughy mirrors the above five properties.

Infrastructure Monitoring and Planning. Power transmission grids can be modeled as a vast, complex, and dynamic graph of redundant transmission paths - between power suppliers and consumers - which the utility provider needs to continuously monitor. Transmission lines can be taken down for maintenance and equipment may simply fail. Moreover, additional dynamicity is introduced by intermittent generators such as from renewable energy sources (e.g., wind, solar) as well as from electric vehicles which - when plugged-in - can act as either consumers or as temporary power suppliers. Supporting the system operators to predict the impact of line outages and to identify the most vulnerable and critical links in the power system, as well as promptly suggesting corrective actions in the event of a transmission line failure is critical to minimize blackouts and equipment damage. This, however, is predicated on efficient support for processing the dynamic graph used to model the power grid [33, 54].

Additionally, supporting historical investigation and auditing (e.g., answering queries like: "What was the state of the grid at some past time T?', "Was a prior decision correct based on the operators' view of the state of the grid at time T?") is essential to analyze and improve operational decisions and relies on modeling the temporal evolution of the system [68]. These investigations need to be supported in a context where sensing information about the state of the grid may arrive with delays or out of order, not only because the communication infrastructure may suffer outages itself, but also because of the various human-factors involved (e.g., operational errors, equipment taken down for maintenance, etc.).

Regulatory Compliance in the Crypto-Currency Market. As crypto-currencies gained increasingly widespread acceptance, the regulatory requirements on market intermediaries (e.g., exchanges, lenders that accept crypto-currency as collateral) have become increasingly rigorous and now include anti money-laundering and 'know-your-customer' (KYC) provisions [29]. At a high level, regulation places the burden of proof on market intermediaries and asks them to demonstrate that they have taken appropriate measures to *"mitigate the risks identified through the implementation of controls and measures tailored to these risks"* [31]. This is particularly complex in the crypto-currency space for two reasons: *(i)* many crypto-currencies have been designed for pseudo-anonymity (transactions between so-called 'wallets' are public on the blockchain, yet the association between wallets and real-world identities remains hidden), and *(ii)* this space has seen wide illicit use [25] and attempts to obfuscate the original source of funds have become increasingly more sophisticated (e.g., using smart contracts [44], 'washing' funds through chains of hundreds of fake transactions - that is, transactions between wallets controlled by the same entity, and/or moving funds across multiple crypto-systems).

In this context, a number of services that augment the transaction graph with additional information have emerged (e.g., Chainalysis [15], and Elliptic [26]). We provide an oversimplified view here: the public transaction graph only includes wallets, transactions between wallets, and their metadata (e.g., valid time and value). New entities and edges are added to this graph: *(i)* pseudo-identities to model the knowledge that the same real-world identity (i.e., person or organization) controls multiple wallets, and *(ii)* actual real-world identities when this information is known. Additionally, new attributes are added to label wallets involved in illegal (e.g., ransomware [39]) and potentially illegal (e.g., markets on the dark-web [25]) activities. Similar labels may be added to transactions involving risky activities (e.g., mixers, smart contracts involving gambling) [14]. Note that this information inherently arrives with delays: for example, while a wallet has been used for ransomware on a specific date (i.e, valid time), this can be discovered and transmitted to the system much later (i.e., transaction time). By mining the enhanced graph, a market participant, can comply with KYC and anti money-laundering regulations, taking measures to monitor and mitigate risks, and, if information is preserved, has the ability to prove to a third party (e.g., to the regulator) that a specific decision was correct given the information available at the time the decision was taken.

Financial Services. The banking and financial services domain operates in many legal jurisdictions with differing laws and regulations regarding data retention, reporting, and auditing (and thus shares many of the challenges identified for the crypto-currency

[2] The Gradoop project [45] has recently added support for bitemporal modeling [76]. As it employs a "bigtable" like data structure [7, 16] at its core, it is implicitly adaptable to support bitemporal modeling by adding transaction time and valid time attributes to all records. However, this design is not driven by nor specialized for bitemporality. In contrast, we design explicitly for bitemporality and propose novel data structures to support it.

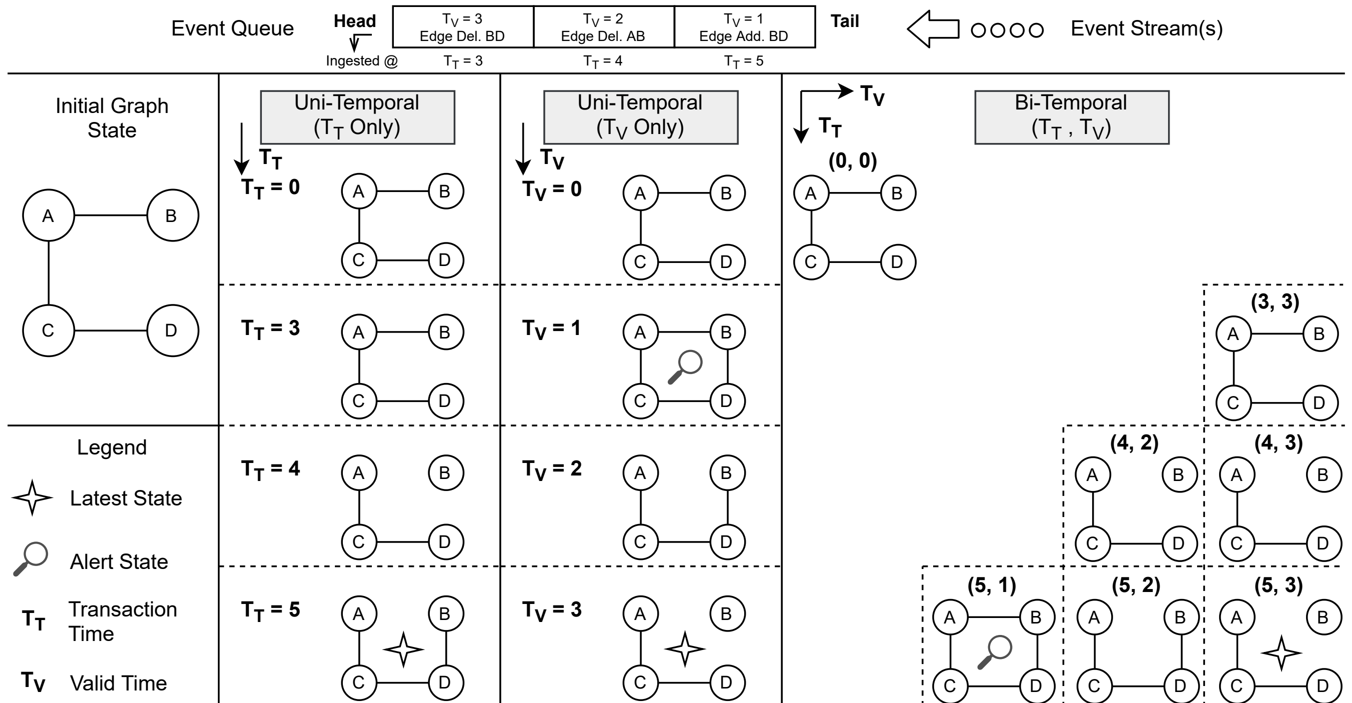

Figure 1: Contrasting the ability of different temporal modeling approaches to completely and correctly track the states the system goes through over time. Note that: *(i)* the final states recorded by each approach, marked with a star, are not the same (i.e., the final state for the Uni-Temporal (T_T) approach is incorrect!), *(ii)* the "alert" state (i.e., when a cycle is formed - marked with a magnifying glass) is not detected correctly in the Uni-Temporal (T_T) case, *(iii)* the Uni-Temporal (T_V) approach is assuming an idealized way of handling out of order events: storing all received events and then replaying them back in the correct order), *(iv)* the Bi-Temporal approach correctly exposes all the states that the system goes through, thus it supports audit queries (while none of the other models can do this), *(v)* the last row in the Bi-Temporal case (e.g., $(5, T_V)$) is equivalent to Uni-Temporal (T_V), and *(vi)* the empty cells in the Bi-temporal Model are intentionally left blank, the query result in any of these cells is the same as the only other state shown within the enclosing region (i.e., the $(0, 0)$ state). Section 3 provides an overview of the temporal modeling approaches, and presents this example scenario in more detail.

regulatory compliance use case mentioned above). A common use case, however, is fraud detection where it is necessary for the system to be capable of: *(i)* ingesting a large volume of transaction related events, enriching them via numerous secondary data sources, and running light-weight checks for detecting fraud [74], *(ii)* efficient mining of historical data for situations where it is necessary to carry out more heavy-weight fraud detection analyses or to generate any data required to train machine learning models, and *(iii)* explicit support for both historical as well as audit queries. In Section 3 we use a mock-up version of this application scenario and dive more deeply into how different ways of modeling the temporal evolution of the dynamic graph data impacts the capabilities of the analytics system and even its ability to operate correctly.

3 BACKGROUND: MODELING TEMPORAL EVOLUTION

A *temporal model* refers to whether/how a system tracks the temporal evolution of a dynamic system, and what query capabilities it inherently provides. Jensen et al. [43] provide a comprehensive glossary of temporal data modeling concepts. The following temporal modeling approaches are of interest:

- *Non-Temporal.* Temporal evolution of the data is not explicitly tracked by the graph analytics system: ingested events are used to update the current state of the graph then discarded. As such, there is only one queryable system state at any given point in time: the current state. A number of existing dynamic graph processing systems fall in this category: [23, 53, 62, 67, 79, 81, 82, 89, 90, 93]. Given that this approach would not meet our requirements to support historical and audit queries we do not discuss it further.
- *Uni-Temporal.* Time is represented by a single axis representing either the time an event has occurred in the real-world or the time the event has been presented to the system. This approach, adopted by [41, 84], provides limited observability and can lead to incorrect conclusions (particularly in the case of out of order events). The example we present below highlights the limitations of this approach.
- *Bi-Temporal.* Employs two axes to represent time [3]: *(i) valid time*: the time an event occurred in the real-world, and *(ii) transaction time*: the time an event is ingested by the system. This separation

[3] While we chose to follow the original terminology proposed by Jensen et al. [43], other terminologies coexist such as: *(i)* event time and processing time [5], *(ii)* event time and ingestion time [13], or *(iii)* application time and system time in SQL:2011 [51].

between "what occurred at time T" vs. "what the system knew at time T" enables not only historical investigations but, more importantly, auditing and forensics.

Example (Fig. 1) We focus on one of the scenarios from Section 2 and present a mock-up example (Fig. 1) to explain how different ways of modeling time impact the ability to support historical and audit queries, and highlight the correctness problems that appear if bi-temporal modeling is not used. This example is motivated by the financial domain where detecting cycles is a building block for complex fraud detection heuristics [74].

Ingestion Pipeline. We assume that incoming events are: *(i)* consumed from multiple independent sources, and *(ii)* timestamped at their respective source with a corresponding Valid Time T_V prior to ingestion into the system. For the purpose of our example, we assume an idealized process of ingesting events: *(i)* assigning all incoming events a Transaction Time T_T in the order of their arrival and storing them in an event buffer, and *(ii)* handing events from this buffer to the system in the order assumed by the specific scenario we explore.

Initial State. In all cases we assume the same initial graph state and event queue (shown at the left and top of Fig. 1 respectively).

Uni-Temporal T_T only. First, we discuss a uni-temporal model where only the transaction time is maintained: the system has no notion of the application-level valid time and only tracks history via the transaction time. Such a system would: *(i)* ingest events in their order of arrival and discard any available application-level timestamps (i.e., the T_V timestamps), *(ii)* employ monotonically increasing transaction timestamps T_T to uniquely identify each historical state, and *(iii)* be able to query those historical states (using the transaction times) in addition to maintaining access to the latest state.

This approach, however, can not be used for historical and audit queries as processing out of order messages in the order of their arrival leads to correctness problems. For our example, with this approach it is not possible to: *(i)* detect the state of interest (i.e., the cycle formation), and, more generally, *(ii)* correctly track the temporal evolution of the system (as highlighted by the incorrect final state).

Uni-Temporal T_V only. Second, we discuss a uni-temporal model that tracks the evolution of the system using the application-level valid time T_V. Conceptually speaking, this is the actual temporal evolution of the system assuming all events arrive in-order. If it were possible to implement it, such a system: *(i)* could be used to query historical states (using the valid time) while also maintaining access to the latest state, and *(ii)* would correctly record all (and only) the states valid from the application-level perspective, including the state where the fraud ring is formed. This is the situation Fig. 1 presents.

In practice, however, this is not possible as it is not feasible to process events in the order of their valid time T_V (at least not for the systems we describe in Section 2). This is for two reasons: first, strong, unrealistic assumptions would be needed to enable in-order event processing. Such assumptions include bounds on event delays, or sequential IDs for events - none of which are possible for the use-cases we present. Second, one would need to relax the requirement to process the latest arriving events and delay their processing until there are guarantees that these events can be processed in the correct order. For our scenarios event delays can easily range from hours to weeks or even more, and delaying processing this long is unreasonable.

The previous paragraph argues that a system that processes events in the order of their valid time T_V is not feasible. The alternative is to process the events in the order of their arrival, and use their valid time T_V to support queries. While this approach can support well current state and historical queries, it fails for audit queries (due to the lack of transaction timestamps). The key issue is that recording only the valid time T_V renders the system unable to record all the states that were observed by the system operators at runtime. For example, in Fig. 1, the recorded history shows that a state of interest occurred; yet, it does not show that the system operator was not able to observe this state.

Bi-Temporal. In contrast, a bi-temporal system tracks time along two axes: the system-level transaction time T_T, and the application-level valid time T_V. For each ingested event the system attaches a monotonically increasing T_T timestamp in addition to the already included T_V timestamp, this allows it to correctly: *(i)* track the temporal evolution of the system (i.e., no invalid states unlike the uni-temporal T_T approach), *(ii)* identify the states of interest (i.e., the fraud ring), and *(iii)* track all states the system goes through for auditing purposes. For example the system can show that, in the actual evolution of the system (i.e., the last row for the Bi-Temporal approach in Fig. 1), there was a state of interest (i.e., a cycle was formed); yet, at any time, an operator querying only the latest state of the system in real-time (i.e., the right-most column) would not have detected this cycle.

4 SYSTEM OVERVIEW

Overview. At a high-level, three main operations are carried out by the system: *(i) ingestion* where the system indexes each incoming event by a tuple of timestamps (T_V, T_T) and ingests it, *(ii) querying* where the system assembles a (possibly virtual) snapshot of the underlying graph at some user-specified query point (T_V, T_T), and *(iii) computation* where the system runs a user-defined compute function on the obtained snapshot.

4.1 Ingestion

Events. At a high-level, an event signifies a change to either the topology (i.e., the structure) or the properties (i.e., the attributes) of the graph tracked by the system such as adding/deleting: *(i)* a vertex, *(ii)* an edge between two vertices, *(iii)* a vertex property, or *(iv)* an edge property. Incoming events are assumed to be timestamped with an application-specific valid time T_V at their original data source. The system processes events in the order of their arrival and assigns them a transaction time T_T. We assume that T_V and T_T are physical timestamps obtained through loosely synchronized clocks.

Logical Invariants. The system maintains the following invariants on the events being ingested at all times: *(i)* the *transaction* time T_T assigned to incoming events must be monotonically increasing, *(ii)* any new event being ingested must have a *valid* time T_V that is less than or equal to the current *transaction* time T_T (i.e., $T_V <= T_T$)

since an event is first generated somewhere and then ingested, and *(iii)* internally, data associated with already ingested events will neither be mutated by the system nor by any subsequent incoming event (i.e., history is never erased / overwritten by the system - only new information about the past is added).

4.2 Querying

***Point-in-time* Query Semantics.** To support the application scenarios mentioned so far, the system supports *point-in-time queries* that is, querying the state of the underlying graph at some user-specified *point* (T_V, T_T) in time:

- *current state queries:* [4] what is the current state of the graph (i.e., what is the state of the graph with all events included up to $T_T = T_V = now$)?
- *historical queries:* what was the state of the graph at some time T_V in the past, given what the system knows now (i.e., given what the system knows at $T_T = now$, what is the state of the graph including all events included up to $T_V < now$)?
- *audit queries:* what was the state of the graph at some time T_V in the past, based on what the system knew at that time or at an earlier time T_T, $T_V \leq T_T < now$? This allows auditing as it can reconstruct any past view a system operator might have observed at T_T.

A Snapshot Perspective. One can view a query as the process of assembling a graph snapshot from the underlying temporal data at the user-specified query point (T_V, T_T) from the entire set of ingested events. A snapshot may be: *(i)* materialized: that is, eagerly constructed on-demand at the time of the query, or *(ii)* virtual: that is lazily built based on the requests from the computation layer. Each of these approaches can be combined with caching to avoid recalculating snapshots or parts of them.

4.3 Computation

Overview. A user may specify a (graph) computation to be executed by the system on a snapshot at (T_V, T_T). Two high-level options are available:

- *External computation.* The snapshot is eagerly constructed and it is shipped to some external graph analytics system that the user has access to, and
- *Internal computation.* In this case the computation is executed internally by the runtime on a virtual snapshot. The snapshot is lazily built in-memory based on individual requests made by the computation runtime to the low-level supporting data structures. The computation runtime is responsible for: *(i)* scheduling the execution of the computation, *(ii)* lazily building the virtual snapshot, *(iii)* managing other temporary data needed for running the computation, and *(iv)* reporting the final result to the user upon successful execution.

The trade-off here is between gaining the ability to leverage specialized and/or highly-optimized existing graph analytics systems for heavy computations (for the external computation option) vs. avoiding expensive serialization/de-serialization of the derived snapshots for lighter computations (the internal computation option).

[4]We consider trigger/alarm queries [79] beyond the scope of the current work.

High-Level Programming Model. For the rest of this section, we sketch one option for the high-level programming model, API, and query execution model for the internal computation option. It is worth noting that, while, we focus on a *Vertex-Centric* (VC) *Bulk-Synchronous-Parallel* (BSP) programming model [88], supporting other high-level programming models can be imagined in a similar way leveraging the underlying low-level bi-temporal data structures presented in Section 5. For example, support for high-level declarative query languages (e.g., Cypher [69], GSQL [36], Gremlin [42], or the upcoming ISO standard GQL [24]) can be naturally implemented as long as they can be applied on a point-in-time snapshot.

For our system, as a proof-of-concept, we aim to first support a *Vertex-Centric* (VC) *Bulk-Synchronous-Parallel* (BSP) programming model [88] as it: *(i)* is expressive and used by many existing graph analytics frameworks [6, 60], *(ii)* makes it possible to write graph algorithms that can be parallelized over large multi-core Non-Uniform Memory Access (NUMA) machines, and *(iii)* is relatively simple to implement/use.

Two key modifications to the design of an existing BSP engine are needed to adopt it within our system: first, information about the desired point-in-time (T_V, T_T) queried by the user needs to be carried down when interrogating the low-level data structures that hold the graph state. Second, the computation runtime would need to lazily assemble and cache the virtual snapshot (to avoid duplicate requests to the low-level data-structures).

5 SUPPORTING DATA STRUCTURES

Overview. To enable bi-temporal modeling of a graph data structure and to support point-in-time querying semantics only two low-level data structures are sufficient:

- A *bi-temporal value* to model the evolution of a specific attribute of an existing vertex/edge.
- A *bi-temporal set* to model the evolution of the vertices of a graph or the evolution of the neighbour set of a vertex.

The rest of this section presents the key choices that have driven the design of these data structures, sketches their internal organization and use (Table 1), and presents a simple example illustrating how they can be used to track the bi-temporal evolution of a vertex outward edge set in Fig. 2 (for the same scenario previously shown in Fig. 1). Table 2 summarizes the operations supported by the two data structures as well as their corresponding runtime/space complexities. We note that our design does not make any assumptions about the order of incoming events; it may be possible to design similar data structures with better properties if one makes additional assumptions - e.g., that most events arrive in (T_V) order, and only a small fraction of them arrive out of order.

Key Choices. The following key choices drive the design of the bi-temporal value and bi-temporal set data structures:

- *Prioritization of Design Goals.* Given the business requirements and application scenarios we previously outlined, we made the following prioritization decisions that drive the choice of the underlying data structures we use (as well as the overall architecture of the system): *(i)* the system should be primarily optimized for fast event *ingestion* given the high expected rate of incoming

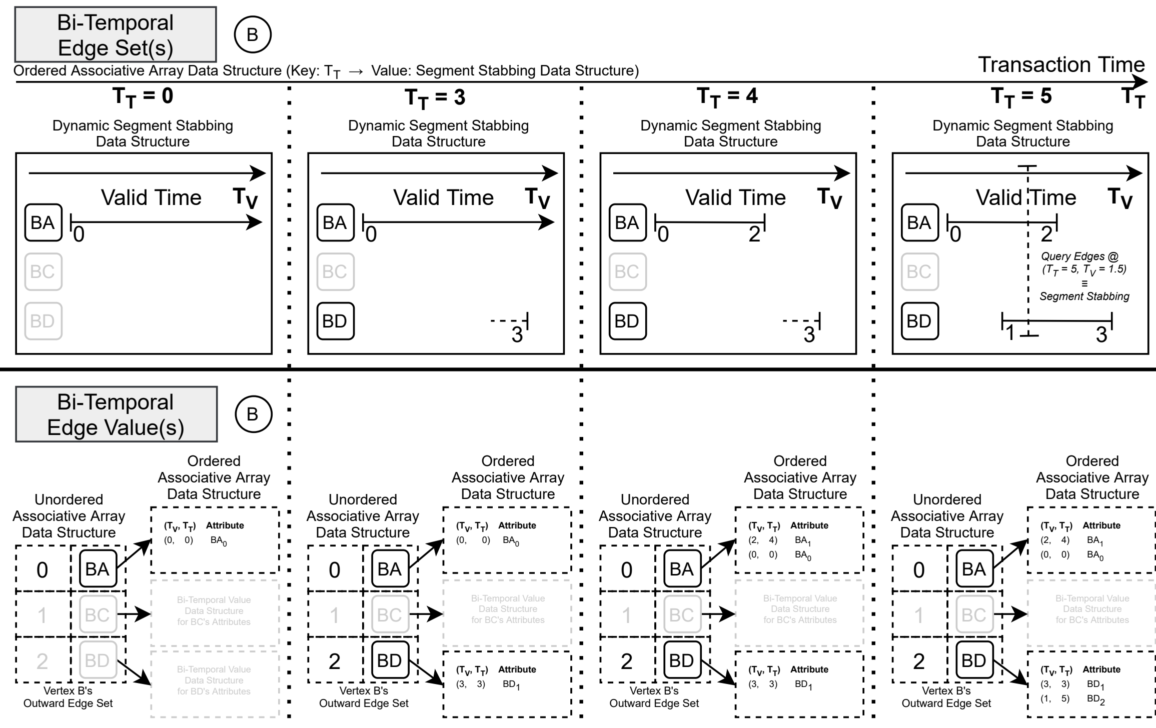

Figure 2: A visualization of the supporting data structures described in Section 5 as well as an illustration of their evolution over time to track a dynamic system. This illustration follows the scenario shown in Fig. 1 and focuses only on vertex B (i.e., the vertex with the most events in the scenario). The figure shows how (a subset) of the supporting internal data structures change to track the temporal evolution of the outgoing set of edges (i.e., using a bi-temporal edge set) as well as the outgoing edge attributes (i.e., using bi-temporal values for the edge attributes) assuming each edge add/delete event also updates the corresponding edge attributes. The figure also visually highlights what a query looks like for the bi-temporal set (i.e., segment stabbing at $T_V = 1.5$). Table 1 presents a simplified overview of query operations.

Table 1: Simplified summary of how the proposed data structures are used for each type of query (i.e., current state, historical, or audit). Best read in conjunction with the visual illustration in Fig. 2, and the corresponding descriptions in Section. 5.

Data Structure	Query Type	Usage
Bi-Temporal Value	Current State	*(i)* Go down the lexicographical 2D (T_V, T_T) ordering,
	Historical	*(ii)* Return the first entry with $T_V <= Query_T_V \wedge T_T <= Query_T_T$
	Audit	⇒ This returns the value of the attribute at $(Query_T_V, Query_T_T)$ (if one exists)
Bi-Temporal Set	Current State	*(i)* Go down the 1D (T_T) ordering, and get the segment stabbing data structure at $T_T <= Query_T_T$,
	Historical	*(ii)* Segment stab at $Query_T_V$, and iterate over the encountered intervals to get pointers to elements
	Audit	⇒ This returns the set of elements alive at $(Query_T_V, Query_T_T)$

Table 2: Summary of the supported operations, and runtime/space complexities for the Bi-Temporal Value and Set data structures described in Section. 5. Note that: *(i)* it is expected that $L <= C$ hence the simplification for the Bi-Temporal Set data structure complexities, and *(ii)* the *iterateElements* operation is an *output-sensitive algorithm* (i.e., its runtime complexity depends on the size of the output: K).
Notation. C: The total number of commits to the data structure (i.e., the total number of tracked historical/audit states). K: The number of reported results (i.e., the total number of elements that exist at (T_V, T_T)). L: The number of element lifetime intervals (i.e., segments) stored in the dynamic segment stabbing data structure. S: The number of times the bi-temporal value was set.

Data Structure	Operation	Average-Case Runtime Complexity	Worst-Case Runtime Complexity	Space Complexity
Bi-Temporal Value	setValue(T_V, T_T, Value)	$O(\log S)$		$O(S \log S)$
	getValue(T_V, T_T)	$O(\log S)$	$O(S)$	
Bi-Temporal Set	addElement(T_V, T_T, Element)	$O(\log C + \log L)$ $\approx O(\log C)$		$O(C \log C + L \log L)$ $\approx O(C \log C)$
	removeElement(T_V, T_T, Element)			
	containsElement(T_V, T_T, Element)			
	iterateElements(T_V, T_T)	$O(\log C + K \log L)$		

events, *(ii)* data structure selection and optimization should be biased for performance (i.e., runtime complexity) rather than memory efficiency (i.e., space complexity) - we aim for *logarithmic* runtime complexity and *quasilinear* space complexity, and *(iii)* query operations on the current state (i.e., $T_T = T_V = now$) should be the fastest supported type of query, followed by historical queries (i.e., $T_V < T_T = now$), and audit queries (i.e., $T_V \leq T_T < now$).

- *Persistent Data Structures.* We make extensive use of *persistent* data structures [22] which: *(i)* provide the key properties needed (i.e., immutability, and copy-on-write), *(ii)* enable efficient access to all previous versions of the data structures (i.e., support efficient historical querying), *(iii)* support concurrent queries (i.e., multiple readers) without requiring the use of expensive coarse-grained locking mechanisms (i.e., a global mutex), and *(iv)* meet our aforementioned design goals for *logarithmic* runtime and *quasilinear* space complexity respectively (as shown in Table. 2).

Supporting Data Structure: Bi-Temporal Value. This data structure enables efficient bi-temporal set/get operations on *a single value* (i.e., provides simple value semantics). Two operations are supported: *(i) setValue*(T_V, T_T, Value), and *(ii) getValue*(T_V, T_T).

A bi-temporal value is represented by a persistent ordered associative array data structure which maintains all historical values ordered descendingly by the valid time first and then by the transaction time (in order to break ties when the valid times are equal). Any set/get operation on this bi-temporal value can now be envisioned as simply going down this lexicographical ordering to either insert a new item or read an existing one (i.e., regardless of the query type). Assuming that a persistent self-balancing binary tree data structure is used to implement the ordered associative array data structure, then the runtime complexity of the set/get operations is, on average, *logarithmic* in the number of historical values.

Supporting Data Structure: Bi-Temporal Set. This data structure enables efficient bi-temporal set/get operations on *a set of elements* at a specified (T_V, T_T) notably: *(i)* adding an element to the set, *(ii)* removing an element from the set, *(iii)* determining membership of an element in the set, and *(iv)* iterating over all elements in the set. The key use for this data structure is to efficiently identify the vertices (or the outward edges of a vertex) that exist at a specific time (T_V, T_T) as these are essential for the query operations carried out by the system (Section 4).

Straw-man Design. We could synthesize a basic bi-temporal set data structure by combining an unordered associative array data structure with the aforementioned bi-temporal value data structure[5]: by creating a mapping between an element ID (e.g., an edge ID in case of an outward edge set, or vertex ID in case of a vertex set) and a bi-temporal boolean value, if the bi-temporal value is true at a specific (T_V, T_T) then at that point in time the element exists. The advantage of this approach is that it is simple to implement, and efficiently supports the first three operations we require from a bi-temporal set data structure. The major drawback is that there is no way to *efficiently* support the final key operation - iterating over all elements at a specific (T_V, T_T): the only way to do this is to iterate over every element and for each element checking its presence at (T_V, T_T) using the corresponding bi-temporal boolean value. The runtime complexity of the iteration will thus be *linear* in the total number of elements ever observed by the set over its entire history as opposed to just the number of elements present at a specific (T_V, T_T) thereby making it unsuitable for highly dynamic systems (e.g., systems with high churn).

[5] This is how we initially approached the design.

Final Design. To efficiently support all four operations we reformulated the problem we are trying to solve as: given a set of element lifetime intervals, where each interval is identified by $[T_V\ Start, T_V\ End]$, and a query point T_V, what are all the elements that have a lifetime interval that intersects this query point? (Fig. 2 illustrates this view: in this case the intervals would track the existence of vertices in the whole graph vertex set, or edges in the outward edge set of a vertex).

This formulation turns out to be a dynamic version of a fundamental problem in computational geometry known as - one-dimensional - *Segment Stabbing* [20] which has many published solutions including one with a *logarithmic* runtime complexity and a *quasilinear* space complexity [66][6]. As most of the advanced solutions to this problem are complex and rely on ephemeral data structures, we instead implemented a *persistent* Interval Tree based on the dynamic but ephemeral solution presented in in [19] (at the cost of somewhat higher complexity for some queries: more specifically *iterateElements()* has a runtime complexity of $O(logC + KlogL)$ instead of $O(LogC + LogL + K)$).

Finally, each time the persistent interval tree is updated (i.e., when a new event is ingested), a pointer to the updated tree is stored into an ordered associative array with the transaction time T_T as the key. This design is possible because updates to the persistent interval tree data structure efficiently create a new version of the structure (i.e., copy-on-write) while preserving all previous versions (i.e., immutability).

Composite Data Structure: The Bi-Temporal Graph. The aforementioned *Bi-Temporal Value*, and *Bi-Temporal Set* data structures can be composed to create a *Bi-Temporal Graph* data structure to track the bi-temporal evolution of an entire graph, and to support a vertex-centric computational model. Generally, when building the graph data structure, at any point there is a need to track the bi-temporal evolution of a set of elements or a single element then a Bi-Temporal Set or Value can be used respectively.

Sketching the composite *Bi-Temporal Graph* data structure from the top-down: *(i)* a *Bi-Temporal Graph* combines a Bi-Temporal Set of *Vertices* with an unordered associative array to access those vertices by their vertex ID, *(ii)* a *Vertex* combines a machine word-size integer vertex ID, an *Outward Edge Set*, and an unordered associative array mapping vertex attribute names to their Bi-Temporal Values, *(iii)* an *Outward Edge Set* combines a Bi-Temporal Set of *Edges* with an unordered associative array to access those edges by their edge ID, and *(iv)* an *Edge* combines a machine word-size integer edge ID, source/destination vertex IDs, and an unordered associative array mapping edge attribute names to their Bi-Temporal Values. *It is important to note that this Bi-Temporal Graph data structure is also persistent, immutable, and copy-on-write since all its primitive components share those properties.*

[6] More specifically, the solution presented in [66] is for the more general dynamic two-dimension orthogonal range and line segment intersection reporting problem.

6 DISCUSSION

We continue by exploring several interrelated topics.

Why propose a design based on persistent data structures instead of one based on conventional ephemeral data structures? Our proposed design has unconventional properties (i.e., it is immutable and copy-on-write), and makes extensive use of persistent data structures which may be not widely known. Here we touch upon an alternative design approach based on ephemeral data structures, discuss the implications of such a design, and contrast it with our proposed design based on persistent data structures.

To maintain information about a collection of items in an ephemeral data structure, at best, the runtime complexity for read/write operations would be $O(1)$ and the space complexity would be $O(N)$. If new information arrives at the system, then the data structure can only be updated in-place by overwriting any previously stored values. While this offers excellent performance characteristics and low memory overhead, this behavior makes it unsuited for historical and audit queries which require: *(i)* storing not just the latest values but all historical ones as well, and *(ii)* providing guarantees that all stored values have never been - accidentally or intentionally - overwritten.

To adapt this design based on ephemeral data structures to be able to meet the aforementioned requirements for historical and audit queries, the solution is to disallow in-place updates and never overwrite previously stored values. A naive way to do this is to generate a completely new copy of the data structure each time it is updated which implies a $O(N)$ runtime complexity for write operations.

A more practical way, however, is to employ persistent data structures [22] which not only offer the properties we require by construction (i.e., immutability and copy-on-write) but also are much more efficient (e.g., $O(\log N)$ runtime complexity for read/write operations) than the comparable straw-man design based on ephemeral data structures we just outlined. We described in more detail how we constructed our supporting data structures earlier (in Section. 5), and summarized their runtime / space complexities in Table. 2. *To the best of our knowledge, we are the first to propose a bi-temporal graph analytics system composed entirely of persistent data structures.*

What are some of the trade-offs for the proposed bitemporal graph analytics system, and its design? There are a few intrinsic costs for our proposed bitemporal graph analytics system - specifically to support current state, historical, and audit queries: *(i)* space and time overheads: potentially significant costs to pay for each new piece of ingested data while guaranteeing that all historical data is preserved, and *(ii)* limiting the optimization options: as we aim to support current state queries, this limits the possibility to process the data to optimize its layout (i.e., either to enable efficient querying or storage). These trade-offs make bitemporal systems in general overall potentially slower compared to existing non-bitemporal graph analytics systems.

Additionally, there are a few additional trade-offs which stem from deliberate decisions we made for this initial design sketch including: *(i)* poor memory locality due to heavy use of persistent data structures which are copy-on-write by design, *(ii)* not taking into account domain-specific optimizations to improve performance for real-world graphs (i.e., unstructured, sparse, scale-free) as part of the initial design sketch, and *(iii)* providing a single implementation for the bitemporal set and bitemporal value data structures in Section. 5 with the expectation that they cover a wide range of workloads (i.e., in contrast to specializing those data structures to the characteristics of a single workload which can net significant performance advantages).

What are some enabling technologies that might help mitigate the aforementioned trade-offs? To mitigate the aforementioned trade-offs, as well as the challenges related to the potentially massive working set size outlined in Section 1, recent advances can be leveraged: *(i)* single-node DRAM memory capacity which has continued to grow rapidly with RAM capacity in the scale of tens of terabytes per node now not being uncommon, *(ii)* processor extensions, such as Intel's 5-Level Paging [40], which are laying the groundwork for a significant increase in the maximum addressable memory space (from 256 terabytes to 128 petabytes in Intel's case) based on the current non-volatile memory capacity growth trends, and *(iii)* byte-addressable, low-latency, and high-bandwidth Storage Class Memory (SCM), such as Intel's Optane based on 3D Cross Point Technology, which are starting to gain increased adoption as a new tier of Non-Volatile Memory (NVM) with the aim to be a cheaper and higher capacity replacement for conventional DRAM. We believe that these advances can be leveraged to make either scale-up (i.e., single-node) system designs, such as the one we plan for our prototype, or scale-out (i.e., multiple-node) designs viable despite the aforementioned trade-offs and challenges.

What are some open questions based on our preliminary work so far? While this project is still in its early stages, we have uncovered a few open questions that we intend to explore as part of our work on this project. We outline a few of those below.

First, prior work in this area has converged on scale-out approaches to handle massive data sets. However the advent of large Storage-Class Memory (SCM), which bridges the gap between DRAM and storage in terms of latency, bandwidth, capacity and cost, raises the question of whether a scale-up approach would now become not just feasible but also superior (e.g., in terms of performance/energy per dollar).

Second, the use of Non-Volatile Memory (NVM) could potentially both enable high-performance and lower the cost of maintaining data durability in spite of failures (e.g., crashes); but what optimizations can be carried out on the stored data layout in order to improve performance and resilience by leveraging the non-volatile nature of the memory?

Third, for our envisioned use cases where the system is expected to be continuously operational, the expectation is that both high availability and reliability must be maintained. On the one hand, to increase availability, online replication is a standard technique one can apply. On the other hand, to increase reliability, durability of the already ingested data becomes a critical concern. It still remains to be seen whether our design - which employs persistent data structures that are immutable and copy-on-write - combined with the use of NVM would be sufficient for a durable scale-up design.

Fourth, a key use case that we envision for the proposed bitemporal dynamic graph analytics systems is for auditing and forensics. One question that stems from this is whether the audit log be made tamper-evident (or tamper-proof) using ideas inspired by techniques employed by blockchains, or the use of specialized devices such as write-once storage.

Finally, given the unique properties attributed to the proposed system and the data that it manages (e.g., transaction times are monotonically increasing, node IDs are assigned by the system, and the data is immutable and only grows over time), what techniques inspired by those used in log-based data stores can be used to further increase performance and efficiency?

What are other query models for evolving graphs beyond point-in-time queries? Section. 7 focuses on the most common approaches to represent and query evolving graphs. One drawback common to the frequently used approach which conceptually relies on point-in-time querying is that it essentially restricts the set of operations that can be expressed [65]. One such query that can not be expressed in a point-in-time model is finding all nodes whose temporal evolution follows a specified pattern of interest (e.g., finding power links with a high number of intermittent failures over time for the infrastructure monitoring and planning use case outlined in Section. 2). In this case, the pattern of interest does not occur at any particular point-in-time.

To support these types of queries, novel query models have been proposed (e.g., Temporal Algebra [65] and its extensions [4]). These, however, are only designed to model the unitemporal evolution of a graph. It is not apparent: *(i)* how easily the aforementioned models and supporting infrastructure can be extended to account for bitemporal modeling, or *(ii)* how effectively our proposed data structures can support the operators that enable the temporal evolution query capabilities suggested by [4, 65].

7 RELATED WORK

While the amount of prior work in this area is vast, space constrains us to a limited coverage. We covered the space for temporal modeling and querying in Section 3, and limited our survey to prior work in the graph analytics space. We do note, however, that there is a large body of work that focuses on temporal modeling in the area of relational databases, including some that support bi-temporal modeling [49, 51, 52].

The rest of this section focuses on categorizing existing graph analytics systems based on their ability to ingest new information about a time-evolving real-world system they attempt to model.

Our Terminology: Offline vs. Online (Streaming and Dynamic) Graph Analytics. Analytics systems which support querying of time-evolving graphs have only recently started to gain attention. Consequently, the terminology is still fluid with terms used by different groups yet with different meanings [10, 63, 71]. For clarity we describe the terminology we use below. First, we separate between *Offline* and *Online* systems. Offline systems have no notion of new incoming updates at runtime, and operate solely on a graph's historical evolution that is known in advance. In contrast *Online* systems operate on a stream of incoming events, and continuously update the underlying graph state (i.e., history is not known in advance). Secondly, we separate between two types of *Online* systems: *Streaming* and *Dynamic*. The major difference between them is the restriction placed on the working set that can be actively maintained by the systems at runtime, and consequently the queries that can be supported: *Streaming* systems operate with limits on the runtime state size while *Dynamic* systems do not.

Offline. An offline system operates on a graph whose entire history is known in advance, and does not allow new updates after this history is ingested. This enables offline systems to: *(i)* heavily pre-process the input graph history, *(ii)* implement advanced memory-layout and partitioning optimizations, and *(iii)* schedule query computations such that their access patterns match the data locality. Systems in this space, which are often referred to in the literature as *historical*, *temporal*, or *time-evolving* graph analytics systems, include: [12, 28, 32, 37, 38, 50, 55, 64, 75, 85, 86]. All of these systems employ uni-temporal modeling to track the evolution of the graph over time, and none of them use bi-temporal modeling. Additionally, offline systems in general, are not suited for our use cases where: *(i)* the graph history is not known in advance, and *(ii)* updates to - as well as queries on - the graph state arrive continuously at runtime.

Online Streaming. An online streaming system operates with the restriction that only a limited amount of information about the graph and its evolution can be kept (typically $V \times log(V)$ or lower [63]) as the whole graph is assumed to be too large to be stored. The immediate consequence is that such a system will be unable to serve historical or audit queries, thus past experience in this area - including prior work such as [3, 8, 9, 11, 21, 27, 30, 46–48, 73] - has limited bearing on the design space we explore. Many of the systems in this space are non-temporal due to the limit on the working set size. However, it is also worth noting that some of the prior work in this space makes the observation that event time and processing/ingestion time must be treated differently [5, 13].

Online Dynamic. An online dynamic system operates without restrictions on the amount of information that can be retained: possibly the entirety of the graph state as well as its evolution. Scale-up, scale-out, and out-of-core processing techniques are all considered viable approaches depending on the graph scale. Many proposed solutions exist in this space with varying supported query capabilities [18, 23, 41, 53, 62, 67, 79, 81, 82, 84, 89, 90, 93]. Section 3 covered how prior work in this space modeled time, and contrasted three temporal modeling approaches (i.e., Non-, Uni-, and Bi- Temporal). We stress that most of the systems we surveyed are non-temporal (i.e., do not explicitly model time, and, as a consequence, can only support current state queries), a few of them are uni-temporal, and none[2] of them implements a bi-temporal data model (i.e., essential to support audit queries). Our proposed system fits in this space, and supports all three query types: current state, historical, and audit.

8 CONCLUSIONS

Summary. Graphs are a key data structure for a wide range of application domains. We contend that existing graph analytics systems which are commonly *static* and *non-temporal*, however, are not able to support real-world use cases which require accurately modeling a *dynamic* graph's evolution over time while concurrently supporting current state, historical and audit queries.

Contributions. *To the best of our knowledge, we are the first to sketch a design for a dynamic graph analytics system that uses bi-temporal data modeling at its core.* We outline the need for such a system through a discussion of real-world use cases and business requirements across several domains, and contend that existing graph analytics systems are not an applicable substitute.

Ongoing Work. We have prototyped the Bi-Temporal Set and Value data structures presented in this paper, and are now working on developing a complete proof of concept implementation for the proposed bi-temporal dynamic graph analytics system presented.

9 ACKNOWLEDGMENTS

This project was sponsored in part by a generous gift fund from Huawei Toronto Heterogeneous Compiler Lab in Canada. We also thank the anonymous reviewers of GRADES-NDA'21 for their insightful comments and detailed feedback which helped us improve the quality of this paper.

REFERENCES

[1] T. K. Aasawat, T. Reza, and M. Ripeanu. 2018. Scale-Free Graph Processing on a NUMA Machine. In *2018 IEEE/ACM 8th Workshop on Irregular Applications: Architectures and Algorithms (IA3)*. Institute of Electrical and Electronics Engineers, New York, NY, USA, 28–36. https://doi.org/10.1109/IA3.2018.00011

[2] Virat Agarwal, Fabrizio Petrini, Davide Pasetto, and David A. Bader. 2010. Scalable Graph Exploration on Multicore Processors. In *Proceedings of the 2010 ACM/IEEE International Conference for High Performance Computing, Networking, Storage and Analysis (SC '10)*. IEEE Computer Society, USA, 1–11. https://doi.org/10.1109/SC.2010.46

[3] Gagan Aggarwal, Mayur Datar, Sridhar Rajagopalan, and Matthias Ruhl. 2004. On the streaming model augmented with a sorting primitive. In *45th Annual IEEE Symposium on Foundations of Computer Science*. IEEE, Institute of Electrical and Electronics Engineers, New York, NY, USA, 540–549.

[4] Amir Aghasadeghi, Vera Zaychik Moffitt, Sebastian Schelter, and Julia Stoyanovich. 2020. Zooming Out on an Evolving Graph. In *EDBT*. OpenProceedings, Konstanz, Germany, 25–36.

[5] Tyler Akidau, Robert Bradshaw, Craig Chambers, Slava Chernyak, Rafael J. Fernández-Moctezuma, Reuven Lax, Sam McVeety, Daniel Mills, Frances Perry, Eric Schmidt, and Sam Whittle. 2015. The Dataflow Model: A Practical Approach to Balancing Correctness, Latency, and Cost in Massive-Scale, Unbounded, out-of-Order Data Processing. *Proc. VLDB Endow.* 8, 12 (Aug. 2015), 1792–1803. https://doi.org/10.14778/2824032.2824076

[6] Apache Giraph. 2012. *Main Project Web Page.* Apache. https://giraph.apache.org/

[7] Apache HBase. 2010. *Main Project Web Page.* Apache. https://hbase.apache.org/

[8] Sepehr Assadi, Sanjeev Khanna, and Yang Li. 2017. On Estimating Maximum Matching Size in Graph Streams. In *Proceedings of the Twenty-Eighth Annual ACM-SIAM Symposium on Discrete Algorithms* (Barcelona, Spain) *(SODA '17)*. Society for Industrial and Applied Mathematics, USA, 1723–1742.

[9] Sepelir Assadi, Sanjeev Khanna, Yang Li, and Grigory Yaroslavtsev. 2016. Maximum Matchings in Dynamic Graph Streams and the Simultaneous Communication Model. In *Proceedings of the Twenty-Seventh Annual ACM-SIAM Symposium on Discrete Algorithms* (Arlington, Virginia) *(SODA '16)*. Society for Industrial and Applied Mathematics, USA, 1345–1364.

[10] Maciej Besta, Marc Fischer, Vasiliki Kalavri, Michael Kapralov, and Torsten Hoefler. 2019. Practice of Streaming Streaming of Dynamic Graphs: Concepts, Models, and Systems. arXiv:1912.12740 [cs.DC]

[11] Sayan Bhattacharya, Monika Henzinger, Danupon Nanongkai, and Charalampos Tsourakakis. 2015. Space- and Time-Efficient Algorithm for Maintaining Dense Subgraphs on One-Pass Dynamic Streams. In *Proceedings of the Forty-Seventh Annual ACM Symposium on Theory of Computing* (Portland, Oregon, USA) *(STOC '15)*. Association for Computing Machinery, New York, NY, USA, 173–182. https://doi.org/10.1145/2746539.2746592

[12] Jaewook Byun, Sungpil Woo, and Daeyoung Kim. 2019. Chronograph: Enabling temporal graph traversals for efficient information diffusion analysis over time. *IEEE Transactions on Knowledge and Data Engineering* 32, 3 (2019), 424–437.

[13] Paris Carbone, Marios Fragkoulis, Vasiliki Kalavri, and Asterios Katsifodimos. 2020. Beyond Analytics: The Evolution of Stream Processing Systems. In *Proceedings of the 2020 ACM SIGMOD International Conference on Management of Data*. Association for Computing Machinery, New York, NY, USA, 2651–2658. https://doi.org/10.1145/3318464.3383131

[14] Chainalysis. 2021. *The 2021 Crypto Crime Report.* Technical Report. Chainalysis. https://go.chainalysis.com/2021-Crypto-Crime-Report.html

[15] Chainalysis. 2021. *The Blockchain Analysis Company.* Chainalysis. https://www.chainalysis.com

[16] Fay Chang, Jeffrey Dean, Sanjay Ghemawat, Wilson C. Hsieh, Deborah A. Wallach, Mike Burrows, Tushar Chandra, Andrew Fikes, and Robert E. Gruber. 2008. Bigtable: A Distributed Storage System for Structured Data. *ACM Trans. Comput. Syst.* 26, 2, Article 4 (June 2008), 26 pages. https://doi.org/10.1145/1365815.1365816

[17] Rong Chen, Jiaxin Shi, Yanzhe Chen, and Haibo Chen. 2015. PowerLyra: Differentiated Graph Computation and Partitioning on Skewed Graphs. In *Proceedings of the Tenth European Conference on Computer Systems* (Bordeaux, France) *(EuroSys '15)*. Association for Computing Machinery, New York, NY, USA, Article 1, 15 pages. https://doi.org/10.1145/2741948.2741970

[18] Raymond Cheng, Ji Hong, Aapo Kyrola, Youshan Miao, Xuetian Weng, Ming Wu, Fan Yang, Lidong Zhou, Feng Zhao, and Enhong Chen. 2012. Kineograph: Taking the Pulse of a Fast-Changing and Connected World. In *Proceedings of the 7th ACM European Conference on Computer Systems* (Bern, Switzerland) *(EuroSys '12)*. Association for Computing Machinery, New York, NY, USA, 85–98. https://doi.org/10.1145/2168836.2168846

[19] Thomas H. Cormen, Charles E. Leiserson, Ronald L. Rivest, and Clifford Stein. 2009. *Introduction to Algorithms, Third Edition* (3rd ed.). The MIT Press, Cambridge, MA, USA, 348–355.

[20] Mark de Berg, Otfried Cheong, Marc van Kreveld, and Mark Overmars. 2008. *Computational Geometry: Algorithms and Applications* (3rd ed.). Springer-Verlag, Berlin, Heidelberg, Germany, 237.

[21] Camil Demetrescu, Irene Finocchi, and Andrea Ribichini. 2010. Trading off Space for Passes in Graph Streaming Problems. *ACM Trans. Algorithms* 6, 1, Article 6 (Dec. 2010), 17 pages. https://doi.org/10.1145/1644015.1644021

[22] J R Driscoll, N Sarnak, D D Sleator, and R E Tarjan. 1986. Making Data Structures Persistent. In *Proceedings of the Eighteenth Annual ACM Symposium on Theory of Computing* (Berkeley, California, USA) *(STOC '86)*. Association for Computing Machinery, New York, NY, USA, 109–121. https://doi.org/10.1145/12130.12142

[23] D. Ediger, R. McColl, J. Riedy, and D. A. Bader. 2012. STINGER: High performance data structure for streaming graphs. In *2012 IEEE Conference on High Performance Extreme Computing*. Institute of Electrical and Electronics Engineers, New York, NY, USA, 1–5. https://doi.org/10.1109/HPEC.2012.6408680

[24] GQL Editors. 2019. *GQL Early Working Draft V2.2.* Technical Report. ISO. https://isotc.iso.org/livelink/livelink?func=ll&objId=20836584&objAction=Open

[25] A ElBahrawy, L Alessandretti, L Rusnac, D Goldsmith, A Teytelboym, and A Baronchelli. 2020. Collective dynamics of dark web marketplaces. *Scientific Reports* 10, 2020 (2020).

[26] Elliptic. 2021. *Bringing Compliance to Cryptoassets.* Elliptic. https://www.elliptic.co

[27] Hossein Esfandiari, Mohammad T Hajiaghayi, Vahid Liaghat, Morteza Monemizadeh, and Krzysztof Onak. 2018. Streaming Algorithms for Estimating the Matching Size in Planar Graphs and Beyond. *ACM Trans. Algorithms* 14, 4, Article 48 (Aug. 2018), 23 pages. https://doi.org/10.1145/3230819

[28] Arash Fard, Amir Abdolrashidi, Lakshmish Ramaswamy, and John A Miller. 2012. Towards efficient query processing on massive time-evolving graphs. In *8th International Conference on Collaborative Computing: Networking, Applications and Worksharing (CollaborateCom)*. IEEE, Institute of Electrical and Electronics Engineers, New York, NY, USA, 567–574.

[29] FATF. 2020. *International Standards on Combating Money Laundering and the Financing of Terrorism and Proliferation: The FATF Recommendations 2012, as updated through October 2020.* Financial Action Task Force. http://www.fatf-gafi.org/media/fatf/documents/recommendations/pdfs/fatf%20recommendations%202012.pdf

[30] Joan Feigenbaum, Sampath Kannan, Andrew McGregor, Siddharth Suri, and Jian Zhang. 2005. On graph problems in a semi-streaming model. *Theoretical Computer Science* 348, 2-3 (2005), 207–216.

[31] FINTRAC. 2021. *Risk Assessment Guidance, as updated through March 2021.* Financial Transactions and Reports Analysis Centre of Canada (FINTRAC). https://www.fintrac-canafe.gc.ca/guidance-directives/compliance-conformite/rba/rba-eng

[32] Francois Fouquet, Thomas Hartmann, Sébastien Mosser, and Maxime Cordy. 2018. Enabling Lock-Free Concurrent Workers over Temporal Graphs Composed of Multiple Time-Series. In *Proceedings of the 33rd Annual ACM Symposium on Applied Computing* (Pau, France) *(SAC '18)*. Association for Computing Machinery, New York, NY, USA, 1054–1061. https://doi.org/10.1145/3167132.3167255

[33] M. H. Gavgani and S. Eftekharnejad. 2017. A graph model for enhancing situational awareness in power systems. In *2017 19th International Conference on Intelligent System Application to Power Systems (ISAP)*. Institute of Electrical and Electronics Engineers, New York, NY, USA, 1–6. https://doi.org/10.1109/ISAP.2017.8071427

[34] Abdullah Gharaibeh, Lauro Beltrão Costa, Elizeu Santos-Neto, and Matei Ripeanu. 2012. A Yoke of Oxen and a Thousand Chickens for Heavy Lifting

Graph Processing. In *Proceedings of the 21st International Conference on Parallel Architectures and Compilation Techniques* (Minneapolis, Minnesota, USA) *(PACT '12)*. Association for Computing Machinery, New York, NY, USA, 345–354. https://doi.org/10.1145/2370816.2370866

[35] Joseph E. Gonzalez, Yucheng Low, Haijie Gu, Danny Bickson, and Carlos Guestrin. 2012. PowerGraph: Distributed Graph-Parallel Computation on Natural Graphs. In *Proceedings of the 10th USENIX Conference on Operating Systems Design and Implementation* (Hollywood, CA, USA) *(OSDI'12)*. USENIX Association, USA, 17–30.

[36] TigerGraph Graph QL Working Group. 2018. *Seamless Syntactic and Semantic Integration of Query Primitives over Relational and Graph Data in GSQL*. Technical Report. TigerGraph. https://cdn2.hubspot.net/hubfs/4114546/IntegrationQuery%20PrimitivesGSQL.pdf

[37] Wentao Han, Kaiwei Li, Shimin Chen, and Wenguang Chen. 2018. Auxo: a temporal graph management system. *Big Data Mining and Analytics* 2, 1 (2018), 58–71.

[38] Wentao Han, Youshan Miao, Kaiwei Li, Ming Wu, Fan Yang, Lidong Zhou, Vijayan Prabhakaran, Wenguang Chen, and Enhong Chen. 2014. Chronos: A Graph Engine for Temporal Graph Analysis. In *Proceedings of the Ninth European Conference on Computer Systems* (Amsterdam, The Netherlands) *(EuroSys '14)*. Association for Computing Machinery, New York, NY, USA, Article 1, 14 pages. https://doi.org/10.1145/2592798.2592799

[39] D. Y. Huang, M. M. Aliapoulios, V. G. Li, L. Invernizzi, E. Bursztein, K. McRoberts, J. Levin, K. Levchenko, A. C. Snoeren, and D. McCoy. 2018. Tracking Ransomware End-to-end. In *2018 IEEE Symposium on Security and Privacy (SP)*. Institute of Electrical and Electronics Engineers, New York, NY, USA, 618–631. https://doi.org/10.1109/SP.2018.00047

[40] Intel. 2017. *5-Level Paging and 5-Level EPT*. Technical Report. Intel. https://software.intel.com/sites/default/files/managed/2b/80/5-level_paging_white_paper.pdf

[41] Anand Padmanabha Iyer, Qifan Pu, Kishan Patel, Joseph E. Gonzalez, and Ion Stoica. 2021. TEGRA: Efficient Ad-Hoc Analytics on Evolving Graphs. In *18th USENIX Symposium on Networked Systems Design and Implementation (NSDI 21)*. USENIX Association, USA, 12.

[42] JanusGraph. 2021. *Gremlin Query Language*. JanusGraph. https://docs.janusgraph.org/basics/gremlin/

[43] C. S. Jensen, J. Clifford, S. K. Gadia, A. Segev, and Richard Thomas Snodgrass. 1992. A Glossary of Temporal Database Concepts. *SIGMOD Rec.* 21, 3 (Sept. 1992), 35–43. https://doi.org/10.1145/140979.140996

[44] Ari Juels, Ahmed Kosba, and Elaine Shi. 2016. The Ring of Gyges: Investigating the Future of Criminal Smart Contracts. In *Proceedings of the 2016 ACM SIGSAC Conference on Computer and Communications Security* (Vienna, Austria) *(CCS '16)*. Association for Computing Machinery, New York, NY, USA, 283–295. https://doi.org/10.1145/2976749.2978362

[45] Martin Junghanns, André Petermann, Kevin Gómez, and Erhard Rahm. 2015. GRADOOP: Scalable Graph Data Management and Analytics with Hadoop. arXiv:1506.00548 [cs.DB]

[46] Michael Kapralov, Sanjeev Khanna, and Madhu Sudan. 2014. Approximating Matching Size from Random Streams. In *Proceedings of the Twenty-Fifth Annual ACM-SIAM Symposium on Discrete Algorithms* (Portland, Oregon) *(SODA '14)*. Society for Industrial and Applied Mathematics, USA, 734–751.

[47] Michael Kapralov, Slobodan Mitrović, Ashkan Norouzi-Fard, and Jakab Tardos. 2020. Space Efficient Approximation to Maximum Matching Size from Uniform Edge Samples. In *Proceedings of the Thirty-First Annual ACM-SIAM Symposium on Discrete Algorithms* (Salt Lake City, Utah) *(SODA '20)*. Society for Industrial and Applied Mathematics, USA, 1753–1772.

[48] Michael Kapralov, Aida Mousavifar, Cameron Musco, Christopher Musco, Navid Nouri, Aaron Sidford, and Jakab Tardos. 2020. Fast and Space Efficient Spectral Sparsification in Dynamic Streams. In *Proceedings of the Thirty-First Annual ACM-SIAM Symposium on Discrete Algorithms* (Salt Lake City, Utah) *(SODA '20)*. Society for Industrial and Applied Mathematics, USA, 1814–1833.

[49] Martin Kaufmann, Peter M. Fischer, Norman May, Chang Ge, Anil K. Goel, and Donald Kossmann. 2015. Bi-temporal Timeline Index: A data structure for Processing Queries on bi-temporal data. In *2015 IEEE 31st International Conference on Data Engineering*. Institute of Electrical and Electronics Engineers, New York, NY, USA, 471–482. https://doi.org/10.1109/ICDE.2015.7113307

[50] Udayan Khurana and Amol Deshpande. 2013. Efficient snapshot retrieval over historical graph data. In *2013 IEEE 29th International Conference on Data Engineering (ICDE)*. IEEE, Institute of Electrical and Electronics Engineers, New York, NY, USA, 997–1008.

[51] Krishna Kulkarni and Jan-Eike Michels. 2012. Temporal Features in SQL:2011. *SIGMOD Rec.* 41, 3 (Oct. 2012), 34–43. https://doi.org/10.1145/2380776.2380786

[52] Anil Kumar, Vassilis J. Tsotras, and Christos Faloutsos. 1995. Access Methods for Bi-Temporal Databases. In *Recent Advances in Temporal Databases*, James Clifford and Alexander Tuzhilin (Eds.). Springer London, London, 235–254.

[53] Aapo Kyrola, Guy Blelloch, and Carlos Guestrin. 2012. GraphChi: Large-Scale Graph Computation on Just a PC. In *Proceedings of the 10th USENIX Conference on Operating Systems Design and Implementation* (Hollywood, CA, USA) *(OSDI'12)*. USENIX Association, USA, 31–46.

[54] Shimiao Li, Amritanshu Pandey, Bryan Hooi, Christos Faloutsos, and Larry Pileggi. 2021. Dynamic Graph-Based Anomaly Detection in the Electrical Grid. arXiv:2012.15006 [cs.LG]

[55] Wouter Lightenberg, Yulong Pei, George Fletcher, and Mykola Pechenizkiy. 2018. Tink: A Temporal Graph Analytics Library for Apache Flink. In *Companion Proceedings of the The Web Conference 2018* (Lyon, France) *(WWW '18)*. International World Wide Web Conferences Steering Committee, Republic and Canton of Geneva, CHE, 71–72. https://doi.org/10.1145/3184558.3186934

[56] Ren-Shuo Liu, De-Yu Shen, Chia-Lin Yang, Shun-Chih Yu, and Cheng-Yuan Michael Wang. 2014. NVM Duet: Unified Working Memory and Persistent Store Architecture. In *Proceedings of the 19th International Conference on Architectural Support for Programming Languages and Operating Systems* (Salt Lake City, Utah, USA) *(ASPLOS '14)*. Association for Computing Machinery, New York, NY, USA, 455–470. https://doi.org/10.1145/2541940.2541957

[57] Yucheng Low, Danny Bickson, Joseph Gonzalez, Carlos Guestrin, Aapo Kyrola, and Joseph M. Hellerstein. 2012. Distributed GraphLab: A Framework for Machine Learning and Data Mining in the Cloud. *Proc. VLDB Endow.* 5, 8 (April 2012), 716–727. https://doi.org/10.14778/2212351.2212354

[58] Lingxiao Ma, Zhi Yang, Han Chen, Jilong Xue, and Yafei Dai. 2017. Garaph: Efficient GPU-Accelerated Graph Processing on a Single Machine with Balanced Replication. In *Proceedings of the 2017 USENIX Conference on Usenix Annual Technical Conference* (Santa Clara, CA, USA) *(USENIX ATC '17)*. USENIX Association, USA, 195–207.

[59] Steffen Maass, Changwoo Min, Sanidhya Kashyap, Woonhak Kang, Mohan Kumar, and Taesoo Kim. 2017. Mosaic: Processing a Trillion-Edge Graph on a Single Machine. In *Proceedings of the Twelfth European Conference on Computer Systems* (Belgrade, Serbia) *(EuroSys '17)*. Association for Computing Machinery, New York, NY, USA, 527–543. https://doi.org/10.1145/3064176.3064191

[60] 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 2010 ACM SIGMOD International Conference on Management of Data* (Indianapolis, Indiana, USA) *(SIGMOD '10)*. Association for Computing Machinery, New York, NY, USA, 135–146. https://doi.org/10.1145/1807167.1807184

[61] Jasmina Malicevic, Baptiste Lepers, and Willy Zwaenepoel. 2017. Everything you always wanted to know about multicore graph processing but were afraid to ask. In *2017 USENIX Annual Technical Conference (USENIX ATC 17)*. USENIX Association, Santa Clara, CA, 631–643.

[62] Mugilan Mariappan and Keval Vora. 2019. GraphBolt: Dependency-Driven Synchronous Processing of Streaming Graphs. In *Proceedings of the Fourteenth EuroSys Conference 2019* (Dresden, Germany) *(EuroSys '19)*. Association for Computing Machinery, New York, NY, USA, Article 25, 16 pages. https://doi.org/10.1145/3302424.3303974

[63] Andrew McGregor. 2014. Graph Stream Algorithms: A Survey. *SIGMOD Rec.* 43, 1 (May 2014), 9–20. https://doi.org/10.1145/2627692.2627694

[64] 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. *ACM Trans. Storage* 11, 3, Article 14 (July 2015), 34 pages. https://doi.org/10.1145/2700302

[65] Vera Zaychik Moffitt and Julia Stoyanovich. 2017. Temporal Graph Algebra. In *Proceedings of The 16th International Symposium on Database Programming Languages* (Munich, Germany) *(DBPL '17)*. Association for Computing Machinery, New York, NY, USA, Article 10, 12 pages. https://doi.org/10.1145/3122831.3122838

[66] Christian Worm Mortensen. 2003. Fully-Dynamic Two Dimensional Orthogonal Range and Line Segment Intersection Reporting in Logarithmic Time. In *Proceedings of the Fourteenth Annual ACM-SIAM Symposium on Discrete Algorithms* (Baltimore, Maryland) *(SODA '03)*. Society for Industrial and Applied Mathematics, USA, 618–627.

[67] Derek G. Murray, Frank McSherry, Rebecca Isaacs, Michael Isard, Paul Barham, and Martín Abadi. 2013. Naiad: A Timely Dataflow System. In *Proceedings of the Twenty-Fourth ACM Symposium on Operating Systems Principles* (Farminton, Pennsylvania) *(SOSP '13)*. Association for Computing Machinery, New York, NY, USA, 439–455. https://doi.org/10.1145/2517349.2522738

[68] Upama Nakarmi, Mahshid Rahnamay Naeini, Md Jakir Hossain, and Md Abul Hasnat. 2020. Interaction Graphs for Cascading Failure Analysis in Power Grids: A Survey. *Energies* 13, 9 (2020), 1–25. https://doi.org/10.3390/en13092219

[69] Neo4j. 2021. *Cypher Query Language*. Neo4j. https://neo4j.com/developer/cypher/

[70] Donald Nguyen, Andrew Lenharth, and Keshav Pingali. 2013. A Lightweight Infrastructure for Graph Analytics. In *Proceedings of the Twenty-Fourth ACM Symposium on Operating Systems Principles* (Farminton, Pennsylvania) *(SOSP '13)*. Association for Computing Machinery, New York, NY, USA, 456–471. https://doi.org/10.1145/2517349.2522739

[71] M. Tamer Ozsu. 2019. *Graph Processing: A Panoramic View and Some Open Problems*. 2019 International Conference on Very Large Data Bases, VLDB'19. https://vldb2019.github.io/files/VLDB19-keynote-1-slides.pdf

[72] Roger Pearce, Maya Gokhale, and Nancy M. Amato. 2010. Multithreaded Asynchronous Graph Traversal for In-Memory and Semi-External Memory. In *Proceedings of the 2010 ACM/IEEE International Conference for High Performance Computing, Networking, Storage and Analysis (SC '10)*. IEEE Computer Society, USA, 1–11. https://doi.org/10.1109/SC.2010.34

[73] Pan Peng and Christian Sohler. 2018. Estimating Graph Parameters from Random Order Streams. In *Proceedings of the Twenty-Ninth Annual ACM-SIAM Symposium on Discrete Algorithms* (New Orleans, Louisiana) *(SODA '18)*. Society for Industrial and Applied Mathematics, USA, 2449–2466.

[74] Xiafei Qiu, Wubin Cen, Zhengping Qian, You Peng, Ying Zhang, Xuemin Lin, and Jingren Zhou. 2018. Real-Time Constrained Cycle Detection in Large Dynamic Graphs. *Proc. VLDB Endow.* 11, 12 (Aug. 2018), 1876–1888. https://doi.org/10.14778/3229863.3229874

[75] Chenghui Ren, Eric Lo, Ben Kao, Xinjie Zhu, and Reynold Cheng. 2011. On Querying Historical Evolving Graph Sequences. *Proc. VLDB Endow.* 4, 11 (Aug. 2011), 726–737. https://doi.org/10.14778/3402707.3402713

[76] Christopher Rost, Andreas Thor, and Erhard Rahm. 2019. Analyzing Temporal Graphs with Gradoop. *Datenbank-Spektrum* 19, 3 (2019), 199–208.

[77] Amitabha Roy, Laurent Bindschaedler, Jasmina Malicevic, and Willy Zwaenepoel. 2015. Chaos: Scale-out Graph Processing from Secondary Storage. In *Proceedings of the 25th Symposium on Operating Systems Principles* (Monterey, California) *(SOSP '15)*. Association for Computing Machinery, New York, NY, USA, 410–424. https://doi.org/10.1145/2815400.2815408

[78] Amitabha Roy, Ivo Mihailovic, and Willy Zwaenepoel. 2013. X-Stream: Edge-Centric Graph Processing Using Streaming Partitions. In *Proceedings of the Twenty-Fourth ACM Symposium on Operating Systems Principles* (Farminton, Pennsylvania) *(SOSP '13)*. Association for Computing Machinery, New York, NY, USA, 472–488. https://doi.org/10.1145/2517349.2522740

[79] S. Sallinen, R. Pearce, and M. Ripeanu. 2019. Incremental Graph Processing for On-line Analytics. In *2019 IEEE International Parallel and Distributed Processing Symposium (IPDPS)*. Institute of Electrical and Electronics Engineers, New York, NY, USA, 1007–1018. https://doi.org/10.1109/IPDPS.2019.00108

[80] M. Shantharam, K. Iwabuchi, P. Cicotti, L. Carrington, M. Gokhale, and R. Pearce. 2017. Performance Evaluation of Scale-Free Graph Algorithms in Low Latency Non-volatile Memory. In *2017 IEEE International Parallel and Distributed Processing Symposium Workshops (IPDPSW)*. Institute of Electrical and Electronics Engineers, New York, NY, USA, 1021–1028.

[81] Bin Shao, Haixun Wang, and Yatao Li. 2013. Trinity: A Distributed Graph Engine on a Memory Cloud. In *Proceedings of the 2013 ACM SIGMOD International Conference on Management of Data* (New York, New York, USA) *(SIGMOD '13)*. Association for Computing Machinery, New York, NY, USA, 505–516. https://doi.org/10.1145/2463676.2467799

[82] Xiaogang Shi, Bin Cui, Yingxia Shao, and Yunhai Tong. 2016. Tornado: A System For Real-Time Iterative Analysis Over Evolving Data. In *Proceedings of the 2016 International Conference on Management of Data* (San Francisco, California, USA) *(SIGMOD '16)*. Association for Computing Machinery, New York, NY, USA, 417–430. https://doi.org/10.1145/2882903.2882950

[83] Julian Shun and Guy E. Blelloch. 2013. Ligra: A Lightweight Graph Processing Framework for Shared Memory. In *Proceedings of the 18th ACM SIGPLAN Symposium on Principles and Practice of Parallel Programming* (Shenzhen, China) *(PPoPP '13)*. Association for Computing Machinery, New York, NY, USA, 135–146. https://doi.org/10.1145/2442516.2442530

[84] Benjamin Steer, Felix Cuadrado, and Richard Clegg. 2020. Raphtory: Streaming analysis of distributed temporal graphs. *Future Generation Computer Systems* 102 (2020), 453–464. https://doi.org/10.1016/j.future.2019.08.022

[85] Matthias Steinbauer and Gabriele Anderst-Kotsis. 2016. DynamoGraph: A Distributed System for Large-Scale, Temporal Graph Processing, Its Implementation and First Observations. In *Proceedings of the 25th International Conference Companion on World Wide Web* (Montréal, Québec, Canada) *(WWW '16 Companion)*. International World Wide Web Conferences Steering Committee, Republic and Canton of Geneva, CHE, 861–866. https://doi.org/10.1145/2872518.2889293

[86] Manuel Then, Timo Kersten, Stephan Günnemann, Alfons Kemper, and Thomas Neumann. 2017. Automatic Algorithm Transformation for Efficient Multi-Snapshot Analytics on Temporal Graphs. *Proc. VLDB Endow.* 10, 8 (April 2017), 877–888. https://doi.org/10.14778/3090163.3090166

[87] Yuanyuan Tian, Andrey Balmin, Severin Andreas Corsten, Shirish Tatikonda, and John McPherson. 2013. From "Think like a Vertex" to "Think like a Graph". *Proc. VLDB Endow.* 7, 3 (Nov. 2013), 193–204. https://doi.org/10.14778/2732232.2732238

[88] Leslie G. Valiant. 1990. A Bridging Model for Parallel Computation. *Commun. ACM* 33, 8 (Aug. 1990), 103–111. https://doi.org/10.1145/79173.79181

[89] Keval Vora, Rajiv Gupta, and Guoqing Xu. 2017. KickStarter: Fast and Accurate Computations on Streaming Graphs via Trimmed Approximations. In *Proceedings of the Twenty-Second International Conference on Architectural Support for Programming Languages and Operating Systems* (Xi'an, China) *(ASPLOS '17)*. Association for Computing Machinery, New York, NY, USA, 237–251. https://doi.org/10.1145/3037697.3037748

[90] Charith Wickramaarachchi, Alok Kumbhare, Marc Frincu, Charalampos Chelmis, and Viktor K. Prasanna. 2015. Real-Time Analytics for Fast Evolving Social Graphs. In *Proceedings of the 15th IEEE/ACM International Symposium on Cluster, Cloud, and Grid Computing* (Shenzhen, China) *(CCGRID '15)*. IEEE Press, New York, NY, USA, 829–834. https://doi.org/10.1109/CCGrid.2015.162

[91] Da Yan, James Cheng, Yi Lu, and Wilfred Ng. 2014. Blogel: A Block-Centric Framework for Distributed Computation on Real-World Graphs. *Proc. VLDB Endow.* 7, 14 (Oct. 2014), 1981–1992. https://doi.org/10.14778/2733085.2733103

[92] Da Yan, James Cheng, M. Tamer Özsu, Fan Yang, Yi Lu, John C. S. Lui, Qizhen Zhang, and Wilfred Ng. 2016. A General-Purpose Query-Centric Framework for Querying Big Graphs. *Proc. VLDB Endow.* 9, 7 (March 2016), 564–575. https://doi.org/10.14778/2904483.2904488

[93] G. Yehuda, D. Keren, and I. Akaria. 2017. Monitoring Properties of Large, Distributed, Dynamic Graphs. In *2017 IEEE International Parallel and Distributed Processing Symposium (IPDPS)*. Institute of Electrical and Electronics Engineers, New York, NY, USA, 2–11. https://doi.org/10.1109/IPDPS.2017.123

[94] Kaiyuan Zhang, Rong Chen, and Haibo Chen. 2015. NUMA-Aware Graph-Structured Analytics. In *Proceedings of the 20th ACM SIGPLAN Symposium on Principles and Practice of Parallel Programming* (San Francisco, CA, USA) *(PPoPP 2015)*. Association for Computing Machinery, New York, NY, USA, 183–193. https://doi.org/10.1145/2688500.2688507

[95] Da Zheng, Disa Mhembere, Randal Burns, Joshua Vogelstein, Carey E. Priebe, and Alexander S. Szalay. 2015. FlashGraph: Processing Billion-Node Graphs on an Array of Commodity SSDs. In *13th USENIX Conference on File and Storage Technologies (FAST 15)*. USENIX Association, Santa Clara, CA, 45–58.

[96] Xiaowei Zhu, Wenguang Chen, Weimin Zheng, and Xiaosong Ma. 2016. Gemini: A Computation-Centric Distributed Graph Processing System. In *Proceedings of the 12th USENIX Conference on Operating Systems Design and Implementation* (Savannah, GA, USA) *(OSDI'16)*. USENIX Association, USA, 301–316.

[97] Xiaowei Zhu, Wentao Han, and Wenguang Chen. 2015. GridGraph: Large-Scale Graph Processing on a Single Machine Using 2-Level Hierarchical Partitioning. In *2015 USENIX Annual Technical Conference (USENIX ATC 15)*. USENIX Association, Santa Clara, CA, 375–386.