Research Statement

CHAO ZHANG, University of Waterloo, Canada

My early research focused on relational data processing. I studied how to leverage computed results to expedite subsequent computations of aggregate functions that are fundamental operations in data analytics. I have explored the opportunities of incremental computations and query rewriting using materialized views.

My subsequent research, primarily during my post-doctoral period, focused on graph processing. My work centered on designing index methods tailored to speed up graph query processing. This involved designing efficient data structures to support fundamental graph computation operators such as reachability and connectivity, mirroring the relational database's B+ tree.

My research has contributed to system advancements by extending Apache Spark, Apache Flink, and Oracle PGX. Additionally, it has advanced theoretical aspects related to user-defined aggregate functions and dynamic connectivity in graph theory.

I summarize my research accomplishments to date, and then I will outline my research roadmap for the next five years. My future research will focus on advanced graph data processing issues, including speeding up computations using index methods or modern hardware, and the impact of graph data on retrieval-augmented generation (RAG) pipelines in generative AI systems.

1 RELATIONAL DATA PROCESSING

Aggregate functions play a foundational role in data analytics. My early research focuses on leveraging computed results to expedite aggregate function computations.

1.1 Incremental computations of aggregate functions

Sliding window aggregations are fundamental in analyzing streaming data, providing insights in temporal patterns, and anomaly identification. These computations involve aggregating data within a window of streaming data, continuously moving the window as new data arrives. The window is managed as new data is inserted and expired data is deleted. Incremental computations play a vital role in significantly enhancing the efficiency of these computations. However, existing algorithms often prioritize either achieving high throughput or maintaining low latency, creating a dichotomy that fails to meet the requirement of real-time data processing.

My research [7, 8] introduced the parallel boundary aggregator (PBA). The aim is to achieve high-throughput and low-latency computation simultaneously. PBA employs two types of simple parallel incremental computations to compute partial aggregations, merged to obtain final computations. This approach maximizes the utilization of fine-grained computation results between continuously updated window snapshots. The tailored design to incremental computations enables PBA to require only 3 operations in the worst case and an average of 2 operations to compute aggregate functions within a window. Ultimately, PBA enhances throughput by up to 4× while simultaneously reducing latency when compared to state-of-the-art algorithms.

1.2 Rewriting queries with user-defined aggregate functions

User-Defined Aggregate Functions (UDAFs) serve as interfaces in mainstream data processing systems, enabling the creation of complex aggregate functions beyond those offered by SQL. However, existing UDAF interfaces typically rely on imperative or procedural languages, present drawbacks that cause performance issues and pose challenges to SQL query engines.

My research [13–15] proposed a declarative UDAF framework aiming to simplify UDAF implementation for end-users and resolve the issues for SQL query engines. This framework decomposes declarative UDAFs into fundamental aggregate functions executable with system built-in functions, reducing context switching overhead between SQL execution and UDAF computation. More importantly, it allows query engines to comprehend UDAF semantics, enabling caching and sharing of UDAF results for computing subsequent UDAFs, instead of computing from scratch. We establish theoretical foundations to identify sharing conditions between UDAFs, enhancing cached result utilization. Furthermore, we combine these conditions with standard aggregate query rewriting techniques, expanding rewriting opportunities for queries with non-equivalent UDAFs. Experimental evaluations in various systems, including PostgreSQL, Spark SQL, MonetDB, SQLite, YeSQL, and SingleStore, illustrated the performance improvement of this declarative UDAF framework.

2 GRAPH DATA PROCESSING

My recent research has been primarily centered on the development of index methods aimed at speeding up two fundamental operations in graph processing: *connectivity* and *reachability* queries. My aim is to boost the efficiency of these critical operations over static and dynamic graphs.

2.1 Indexing for connectivity queries

Connectivity queries, pivotal in checking the existence of an undirected path between two nodes, stand as one of the most foundational graph computations. However, devising index methods for connectivity queries in dynamic graph settings poses notable challenges, primarily rooted in efficiently reflecting graph changes into the maintained index. Dealing with connectivity queries in dynamic graphs that include edge insertions and deletions is commonly referred to as *fully dynamic connectivity* (FDC) [4]. Designing an FDC approach that operates in $O(\log n)$ time (even in amortized scenarios) remains a longstanding challenge. The primary challenge arises during edge deletions, causing graph traversals that significantly impact performance.

In my research [11], I focused on designing an index tailored for processing connectivity queries in streaming graphs. In this context, connectivity queries operate within a window of edges, with continuous edge insertions and deletions occurring within that window. Notably, the proposed index circumvented the need for computationally expensive edge deletion operations, thus alleviating the performance bottlenecks typically encountered in maintaining connectivity indexes within dynamic graph settings, *i.e.*, an FDC approach. This index method achieved an amortized time complexity of $O(\log n)$ for both queries and updates in real-world graphs, under the practical assumption of a limited number of connected components. Comprehensive experiments highlighted the efficiency of the index, showing an improvement of up to two orders of magnitude compared to FDC-based approaches.

2.2 Indexing for reachability queries

Reachability queries, which verify the existence of directed paths between two nodes, are fundamental operations for exploring transitive relationships in graphs. In modern graph data models, edges are assigned with labels, aiding in detailed reachability analytics by setting specific path constraints. Termed as *regular path queries* due to their formulation using regular expressions, these queries present challenges in efficient processing within graph data systems.

My research [9] focused on building an index method tailored for reachability queries with recursive label-concatenated path constraints. Mainstream graph data systems typically encounter timeouts while processing these queries [1]. Our index method was built upon the foundational 2-hop labeling data structure [2], incorporating succinct information to facilitate path constraint evaluation. Our index showcased an improvement of six orders of magnitude in query processing, compared to methods based on graph traversals. Moreover, our index revealed intriguing trade-offs between the cost of index construction and the benefits in query processing, particularly when compared to mainstream graph databases. For instance, the efficiency gained in processing only a few queries in Virtuoso justified the initial investment required for building the index.

We also developed a tutorial [10] at SIGMOD 2023 on index methods for reachability queries and extended that to a full survey paper [12] that is now under review, aiming to pave the way for the development of full-fledged index methods in graph database systems.

3 FUTURE RESEARCH

I intend to conduct research on advanced graph data processing. My focus will center on addressing complex and inadequately explored issues in database management systems and generative AI systems. This diverse spectrum of subjects will engage my research group during my early career.

3.1 Index methods in graph query processing

Indexes in graph databases. The new SQL standard, SQL:2023, introduces a new feature supporting property graph queries called SQL/PGQ. This simplifies data querying within graphs compared to conventional methods using table self-joins. This advancement underscores the necessity for modern data systems to efficiently process graph queries. While certain simple cyclic pattern matching queries, such as triangle matching, can be handled using worst-case optimal join algorithms, mainstream data systems struggle with complex graph queries involving navigational queries due to the recursive nature of node and edge traversal. Despite existing index methods tailored to specific classes of navigational queries, their practical implementation faces significant obstacles related to generality and update efficiency – see the discussion in [10, 12]. My research will focus on developing an index capable of efficient updates and processing a wide spectrum of graph queries, significantly speeding up query processing.

Indexes for streaming and temporal graph computations. Beyond traditional graph databases, the increasing significance of streaming graphs and temporal graphs in analyzing temporal entity connections introduces a similar necessity for efficient index methods in processing graph queries. Streaming graphs involve continuously updated windows for query processing. Temporal graphs maintain the entire temporal history of the database and allow time-travel queries over windows of various time intervals. The dynamicity of data in these settings presents challenges in index method design. The overlapping nature of windows offers opportunities for incremental computations, as explored in our recent work focusing on streaming graphs. My research in this direction will aim to develop index methods conducive to incremental updates.

3.2 Modern hardware for graph computations

NVMe SSDs. The latest generation of NVMe SSDs boasts an I/O bandwidth four orders of magnitude higher than traditional hard disk drives (HDDs), accompanied by a three-orders-of-magnitude reduction in random I/O response time. In addition, NVMe SSDs will become more cost-effective than HDDs within the next five years. This substantial leap in NVMe SSDs capabilities is poised to significantly influence the architecture of data processing systems. However, current data processing systems do not fully leverage the potential advantages of NVMe SSDs [3]. My research will focus on harness NVMe SSDs to accelerate data processing, specifically within the domain of large-scale graph data computation. Large graph processing necessitates substantial memory resources. The emergence of NVMe SSDs prompts an exploration into the capabilities of a single machine compared to distributed architectures. Distributed architectures involve graph partitioning and additional communication costs, particularly significant in the domain of large-scale graph data computation.

NVLink and GPUDirect in GPU-based acceleration. Considerable effort has been devoted to GPU-based acceleration in data processing. The performance of such computations is often bounded by the bandwidth of PCIe due to sub-optimal interconnect between CPUs and GPUs. This limitation becomes more critical as GPUs possess restricted memory capacities, often requiring multiple rounds of data transfer. Newer and faster interconnect technologies like NVLink, and advancements like GPUDirect, enabling direct GPU data access bypassing CPUs, present opportunities to mitigate these performance bottlenecks [6]. My research will aim to harness these emerging technologies to improve data processing performance, particularly in scenarios involving large-scale graph computations that require substantial memory resources.

3.3 Graphs for retrieval-augmented generation

Retrieval-augmented generation (RAG) [5] has emerged as a promising approach to addressing critical issues faced by large language models (LLMs), such as knowledge cutoff and hallucinations. Within the RAG framework, pertinent data is retrieved from a vector database and fed into an LLM with given questions. The efficacy of retrieval is affected by the process of chunking data into the vector database. Each chunk is encoded as a vector and indexed for storage. Retrieval relies on a similarity score computed between the vector representing the query and those representing individual chunks. Conventional chunking, performed per text paragraph, poses limitations as individual chunks may diverge in meaning, potentially leading to irrelevant or hallucinated responses from LLMs. To mitigate this, my research will focus on constructing a comprehensive global data representation layered atop chunked data. This will involve a preliminary step of creating a global data representation prior to the retrieval process to facilitate the identification of relevant chunks within the vector database. I believe graphs are an ideal data model for global data representation. Their inherent ability to capture connections among data elements aligns with the goal of refining the relevance of retrieved chunks to a given context.

REFERENCES

[1] Angela Bonifati, Wim Martens, and Thomas Timm. 2019. Navigating the Maze of Wikidata Query Logs. In *Proc. 28th Int. World Wide Web Conf.* 127–138.

[2] Edith Cohen, Eran Halperin, Haim Kaplan, and Uri Zwick. 2002. Reachability and Distance Queries via 2-Hop Labels. In *Proc. 13th Annual ACM-SIAM Symp. on Discrete Algorithms.* 937–946.

[3] Gabriel Haas and Viktor Leis. 2023. What Modern NVMe Storage Can Do, and How to Exploit It: High-Performance I/O for High-Performance Storage Engines. *Proc. VLDB Endowment* 16, 9 (2023), 2090–2102.

[4] Jacob Holm, Kristian de Lichtenberg, and Mikkel Thorup. 2001. Poly-Logarithmic Deterministic Fully-Dynamic Algorithms for Connectivity, Minimum Spanning Tree, 2-Edge, and Biconnectivity. *J. ACM* 48, 4 (2001), 723–760.

[5] Patrick Lewis, Ethan Perez, Aleksandra Piktus, Fabio Petroni, Vladimir Karpukhin, Naman Goyal, Heinrich Küttler, Mike Lewis, Wen-tau Yih, Tim Rocktäschel, Sebastian Riedel, and Douwe Kiela. 2020. Retrieval-Augmented Generation for Knowledge-Intensive NLP Tasks. In *Advances in Neural Information Processing Syst., Proc. 34st Annual Conf. on Neural Information Processing Syst.* Article 793, 16 pages.

[6] Clemens Lutz, Sebastian Breß, Steffen Zeuch, Tilmann Rabl, and Volker Markl. 2020. Pump Up the Volume: Processing Large Data on GPUs with Fast Interconnects. In *Proc. ACM SIGMOD Int. Conf. on Management of Data.* 1633–1649.

[7] Chao Zhang, Reza Akbarinia, and Farouk Toumani. 2021. Efficient Incremental Computation of Aggregations over Sliding Windows. In *Proc. 27th ACM SIGKDD Int. Conf. on Knowledge Discovery and Data Mining.* 2136–2144.

[8] Chao Zhang, Reza Akbarinia, and Farouk Toumani. 2023. Parallelization of Incremental Aggregations over Sliding Windows. Under review.

[9] Chao Zhang, Angela Bonifati, Hugo Kapp, Vlad Ioan Haprian, and Jean-Pierre Lozi. 2023. A Reachability Index for Recursive Label-Concatenated Graph Queries. In *Proc. 39th Int. Conf. on Data Engineering.* 67–81.

[10] Chao Zhang, Angela Bonifati, and M. Tamer Özsu. 2023. An Overview of Reachability Indexes on Graphs. In *Companion of ACM SIGMOD Int. Conf. on Management of Data.* 61–68.

[11] Chao Zhang, Angela Bonifati, and M. Tamer Özsu. 2023. Incremental Sliding Window Connectivity over Streaming Graphs. Under submission.

[12] Chao Zhang, Angela Bonifati, and M. Tamer Özsu. 2023. Indexing Techniques for Graph Reachability Queries. Under review.

[13] Chao Zhang and Farouk Toumani. 2020. Sharing Computations for User-Defined Aggregate Functions. In *Proc. 23rd Int. Conf. on Extending Database Technology.* 241–252.

[14] Chao Zhang and Farouk Toumani. 2023. Sharing Queries with Nonequivalent User-Defined Aggregate Functions. Under review.

[15] Chao Zhang, Farouk Toumani, and Bastien Doreau. 2020. SUDAF: Sharing User-Defined Aggregate Functions. In *Proc. 36th Int. Conf. on Data Engineering*. 1750–1753.