Graph Navigation Query in an Edge Streaming Database

Tarek Aoukar† Jun Miyazaki‡

Department of Computer Science, School of Computing, Tokyo Institute of Technology, 2-12-1 Ookayama, Meguro-ku, Tokyo 152-8550, Japan

E-mail: †aoukar@lsc.c.titech.ac.jp, ‡miyazaki@lsc.c.titech.ac.jp

Abstract: In this paper, we present a new solution in graph database systems that takes advantage of two different types of graph databases (navigation query oriented, global graph processing oriented) to obtain balanced performance on two different workloads simultaneously with low overhead. In recent years, internet users have been generating data at a highly accelerating rate, and in various formats, urging the need to develop various types of database systems. Traditional relational databases no longer are the most effective solutions for all types of data storage and processing, for example: Web graphs, SNS, and Transportation networks data are better suited for graph databases due to the nature of the data and processing behavior (n-hop neighbors, PageRank, WCC). Using the grid format for a graph as presented in GridGraph and combining it with graph edge linkage like Neo4j, our solution can achieve good performance regardless of the workload on it. We show some performance evaluation of the proposed system as well as the system architecture.

Keywords: Database Core Technology, Graph Database, Data Structures

1. Introduction

In recent years, the Internet users have been generating data at a highly accelerated rate, and in various formats, opening the gate to various types of databases to be involved as storage and processing solutions. Traditional relational databases no longer are the most effective solutions for all types of data storage and processing. For example, Web graphs, SNS, and transportation networks data are better suited for graph databases due to the nature of the data and processing behaviour.

Graph databases have gained a lot of attention lately and can be categorized based on their system architecture to: a) distributed systems: as a natural solution for the ever growing data size distributed systems such as Pregel[1], PowerGraph[2], Chaos[3], and b) out-of-core systems such as GraphChi[4], X-Stream[10] and others[7,15]. Distributed systems are a natural solution for the ever growing data size, and have the ability for very large scale graph processing powered by their scale-out architecture. On the other hand, out-of-core systems can avoid the difficulties of distributed systems such as synchronization, load balancing, and clustering.

Graph databases can also be categorized based on their focus into one of two categories: 1) Navigation query oriented ones such as Neo4j[9] and DGraph[19], and 2) Global graph processing (edge streaming) oriented ones such as GraphChi[4] & GridGraph[7]. The former focuses on navigation queries such as searching n-hop neighbors, and aims to obtain high performance by optimizing the data structure and data layout used by the data storage and processing engine, while the latter focuses on analysis on the graph as a whole for running algorithms such as PageRank, WCC, SpMV, etc, by optimizing their data structure and data layout for sequential access.

Neo4j[9] is an open source, fully ACID, transactional property graph database where properties are represented as key-value pairs. Data is stored on a disk in the form of records with 2-level caching to improve performance, on which a record only holds the data to the first relationship while in object cache it holds all its relationships classified by types. Its performance of navigation queries is multiple times higher than that of MySQL even in small datasets.

In this paper, we propose a system that aims to balance between global graph processing and navigation queries by exploiting the data structures used by navigation query oriented databases to reduce the amount of accessed data, and the data layout from global graph processing databases to maximize the sequential access to a hard disk.

The rest of the paper describes some of the existing works that inspire our proposed method, followed by the data structures and algorithms behind our database processing model and query execution. We report experimental results and conclude the paper.

2. Related Work

Recently, a number of systems have been proposed for graph processing[5,6,13,14,8]. We draw attention to some of the notable works.

2.1. Distributed systems

Pregel[1], which is a distributed abstraction for graph-based computation, follows the vertex centric computation model where programs are expressed in iterations of passing messages between vertices and synchronizing at superstep intervals. Pregel is a simple framework for graph computation that is both scalable and fault tolerant hidden behind a simple API.

PowerGraph[2] uses the Gather, Apply, Scatter (GAS) computation model. It focuses to solve the problem of natural graphs which follows the highly skewed power-law degree distribution by factoring computation over edges instead of vertices, so as to increase the parallelism in carried computations.

TigerGraph[20] is a native graph database heavily optimized and designed for massive parallel graph analysis. It contains automatic partitioning of graph and a hash index is used to determine which server data belong to. TigerGraph equips a query language, called GSQL, which is friendly to both bulk-synchronous-programming and map-reduce users.

2.2. Single machine systems

X-Stream is introduced as a scale-up graph processing system for both in memory and out-of-core graphs on a single machine using the GAS model. However, it adopts edge centric implementation to stream unordered edges, instead of random access with states stored in vertices.

Another notable single machine graph processing system is GridGraph[7], which divides graph nodes into 1D partitions and is used to build a 2D edge grid in which each cell contains directed edges from a source partition to a destination partition. It scales well with memory and disk bandwidth and outperforms other systems

like X-Stream and GraphChi.

2.3. Flat datacenter technique

Chaos[3] exploits the streaming partitions provided by X-Stream[10] for sequential access and parallelizes it across machines in a small cluster based on the observation that storage bandwidth is much lower than network bandwidth in a small cluster. In this case, the preprocessing of a graph is trivial. It splits graph data such as vertices, edges, and any intermediate data, and each data fragment is stored to randomly selected storage devices of the cluster, following a uniform distribution. It also allows work-stealing for load balancing. Due to these features, it scales out from 1 to 32 machines only with 1.6X increase in runtime.

3. Architecture of the Proposed System

In this section, we describe the data structures, data layout, and processing model of our system. Our system uses LevelDB[16] as underlying key-value storage which has been developed by Google. LevelDB uses log structured writing, and provides an easy and robust API for fast prototyping. It is also easy to replace with an optimized data access layer at later stages of our system.

3.1. Data Structures

The candidate data structures we used are similar to ones used in Neo4j. The three main data structures are as follows: 1) graph node, 2) graph edge, and 3) edge linkage.

The graph node stores a relationship pointer that points to an edge of the graph in which the node is either a source or a destination, and a property block pointer. The graph edge holds a pointer from a node source node to a destination node, as well as a property block pointer. The edge linkage contains useful navigation pointers which are used as a helper for navigation queries to reduce the amount of streaming needed. Figure 1 below illustrates data structures and how they are connected.

No record stores its own ID. Instead, the ID is the actual offset of the record in the database, and each record holds a validity bit that is used for fast record deletion without overwriting the record immediately. Property block pointers in each record can be used to store 4 bytes of arbitrary data, such as the weight of an edge, if the user data is small.

Records are grouped in disk pages (a byte array container), and each record manipulates its own data directly into the associated disk page to facilitate easier data retention mechanism alternative to detecting record data update and copying data over.

3.2. Data Layout

A disk page is organized in the same manner as Partition Attributes Across (PAX)[17] where fields are grouped together, which improves performance in certain operations such as finding edges with a certain source or destination in a page due to cache access.

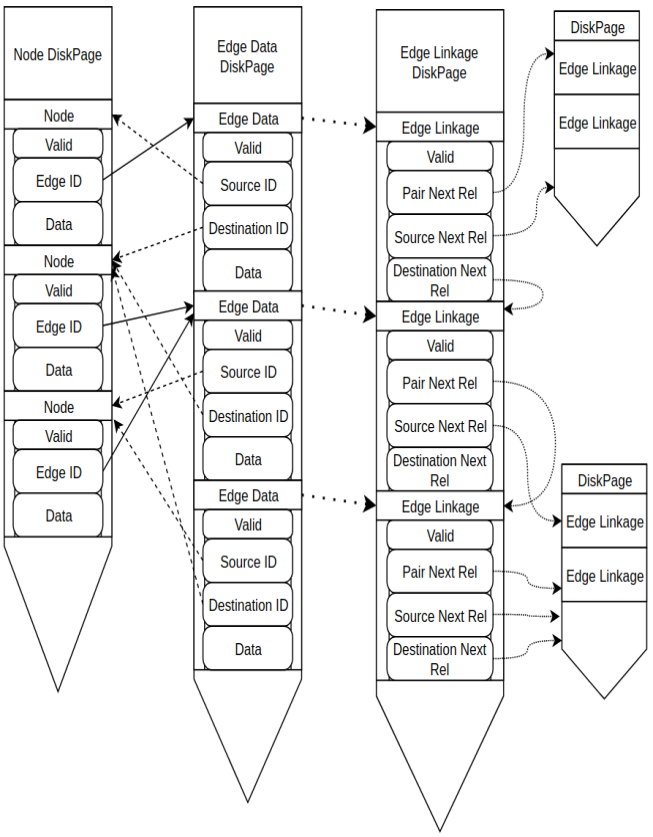

Figure 1: Data structures interconnections

As in GridGraph, the nodes of the graph are split into P partitions, and edges are split into $P \times P$ grid where a cell in position (i,j) holds the edges with source nodes in partition i and destination nodes in partition j. This is built under the assumption that each partition can totally fit into memory with enough free space for auxiliary data. However, it is the user's responsibility to find and provide the number of partitions, which means that prior knowledge about the number of nodes in the graph is required. Otherwise, the user may define the number of partitions. However, too small number of partitions results in too large partitions which do not fit in memory. It also limits the possibility of evolving the graph since adding nodes may increase the partition size to the maximum memory size and it no longer can fit in memory. Our approach, on the other hand, requires the definition of memory limit per partition and dynamically creates partitions if needed. Edge grid data is stored in a sequential manner according to destination and source partitions allowing to sequentially stream all edges with the same destination partition, and therefore, allowing immediate update of node values. This can help some global graph algorithms to converge faster.

In a similar fashion to edge data grid, the edge linkage data is also stored in the grid format and only loaded when required by graph navigation queries to reduce the I/O and memory usage.

Each cell in the grid can uniquely be identified by the higher 8 bytes representing 4 bytes for destination and 4 bytes for source. On loading edge cells into memory when required by navigation queries, edge cells are split into sequential pages. This allows to load uniquely identified edge data pages without streaming the whole cells.

3.3. Pre-processing

The pre-processing stage takes an input graph file in the edge list format (source_node destination_node) and converts it by the following process:

1. The main thread reads edges sequentially from the file and dispatch the edge to a responsible processing thread according to source and destination partitions

2. Worker threads calculate the cell to which the edge belongs and write it to an available disk page. The number of worker threads is limited by the multi-threading hardware support.

In addition, we implement a buffering mechanism to improve performance by reducing disk accesses.

3.4. Processing Model

The processing model is different depending on the types of queries. Therefore, we explain how queries are handled for each type of the load.

The overview of our system architecture is shown in Figure 2 below.

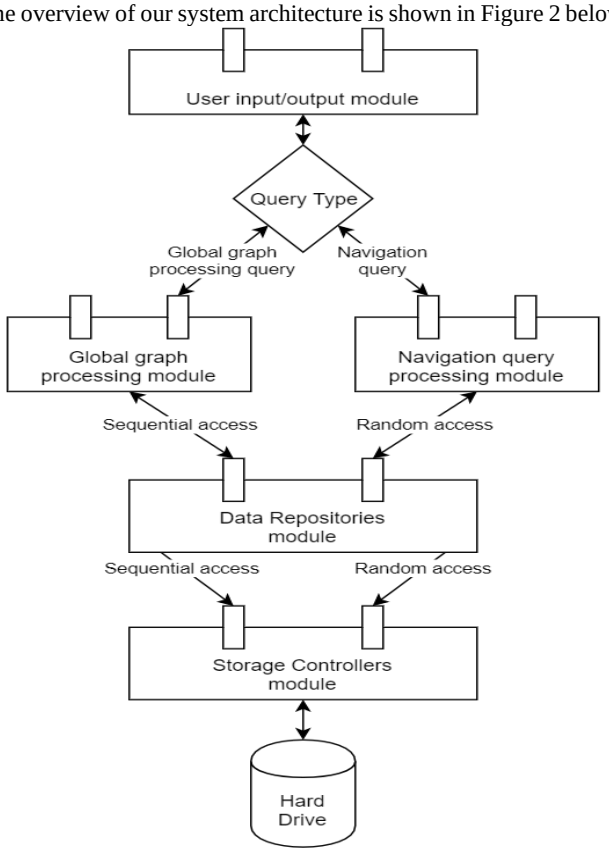

Figure 2: System architecture overview

3.4.1. Global Graph Query

For global graph algorithms, such as PageRank, WCC, SpMV, etc. the Stream-Apply computation model is used. The I/O cost for a single iteration is a single sequential read over the edge data, a single sequential write and P^2 random reads over the graph nodes data (only write modified pages).

The programming interface is robust and simple as shown in the following algorithm.

Algorithm Global graph algorithm query

```
function Execute()
  while(shouldIterate())
    for each node destination page      ◄ dp
      for each node source page         ◄ sp
        for each edge in cell(dp,sp)
          processEdge(edge, source,
        destination,accumulator)
    for each node in dp
      updateNode(node, accumulator)
```

The functions shouldIterate(), processEdge(), and updateNode() are defined by a user, while iterating and saving changes are controlled by the system. For advanced users, customized initialization and finalization functions can be overwritten as well as providing a custom accumulator for more complicated processing. As an example of global graph algorithms, an implementation for the PageRank algorithm is shown below. Note that for simplicity, a PageRank value and an out degree value are stored in the node property pointer and the edge property pointer, respectively.

Algorithm PageRank algorithm

```
function shouldIterate()
    if (diff / v <= threshold)
      return false
    diff = 0
    return true

function processEdge(edge, src, dst, acc)
    acc += src.GetProperty() / edge.GetProperty()

function updateNode(node, acc)
    float new_pr = 1 - d + d * acc
    diff += node.GetPropertyId() - new_pr
    node.SetPropertyId(new_pr);
```

3.4.2. Navigation Query

With graph databases focusing on global graph processing, navigation queries have poor performance. They require scanning over the whole set of edges multiple times because they cannot identify the positions of the required edges. Navigation oriented graph databases, on the other hand, store pointers to link edges forming a linked list data structure. Our system separates the linkage pointers from the edge data and stores them in a cell of the grid similar to edge data that allows one-to-one mapping between the two cells. During navigation query, this linkage data is loaded into memory, and passed to the navigation query engine. Then, the navigation query engine uses linkage data to decide which edges contribute to the query, and loads their pages accordingly, so that the sequential scan problem can be avoided. Moreover, it is also possible to load and navigate multiple edge linkage data pages before deciding which edges are needed by the query. This approach has the advantages of 1) reducing I/O compared to global graph processing engines, 2) exploiting sequential access and data locality provided by the data layout in grid format. Sequential access is not provided by the graph databases for navigation query where edges of the same graph node pair may end up scattered across the whole

database file requiring to seek back and forth to follow the pointers.

4. Experimental Evaluation

We evaluated our system to show the viability of its performance with real world social graphs on a commodity PC.

4.1. Datasets

We chose a common and available dataset, the Twitter-2010[18] dataset, which contains over 41.7 million users connected by 1.5 billion relations with 26.2GB in total.

4.2. Experiment Setup

The experiments were conducted on a PC running 64bit Ubuntu 18.04.3 LTS equipped with an Intel® Core™ i7 CPU running at 4GHz, 16GB of main memory and 2TB of a HDD (7200 RPM). Note that in most of our experiments memory consumption maintained a maximum of 6GB.

4.3. Experiment Results

4.3.1. Pre-Processing

For the pre-processing, we have used 6GB of memory, this can be manipulated by four parameters: 1) the number of nodes per disk page, 2) the number of edges per disk page, 3) the number of node pages to cache in memory, and 4) the number of edge pages to cache in memory. Table-1 shows the results of the experiment we carried along with the chosen parameters.

Nodes/ page	Edges/ page	Node page/ batch	Edge page/ batch	time(s)
1048576	32768	60	1750	3665
1048576	65536	60	3500	3570

Table 1: pre-processing results

In the current configuration, we set the node page size to 21MB, the edge page size to 29KB, the cache size for node pages to 60, and that for edge pages to 7000. It took 48 minutes to pre-process twitter-2010 dataset of which size is more than 26GB. The result data was 21GB, which is nearly 20% smaller than the original database. We need to optimize the parameters, and evaluate the performance in detail.

4.3.2. PageRank

PageRank is one of the most well-known and used global graph processing algorithms. Our initial experiment with PageRank was carried on a copy of the database 2here initial PR values for all nodes are set to 1, each node page contained 1048576 nodes, each edge page contained 65536 edges. The number of threads changed either a single thread or 8 threads. In each run, it took 16 iterations to converge the final PR. It took 35469 seconds averaging 2216 seconds per iteration with a single thread, while resulting in 12320 seconds averaging 770 seconds per iteration with 8 threads. Those results can be further improved by implementing a scheduler to skip the converged page.

4.3.3. NodeDegree

We implemented the NodeDegree calculation using 2 different approaches. The first approach calculates the degree of the nodes in each partition by parallel iteration over the related source and destination pairs, in terms of the edges grid, this implementation operates on a fixed cell and parallel compute its edges' degrees, before moving to next cell.

The second approach calculates the degree of different cells in parallel. Both implementations achieved similar performance when running with 1- 4 threads. However, with 8, 16 threads, the second implementation has achieved better performance, this can be related to the fact that the first approach requires thread-safe counting since all threads are operating over the same cell, this was implemented with atomic integers. The second approach, however, has a benefit that each thread operates on a different cell, since it has its own counters.

The following Figure3 shows our experimental results.

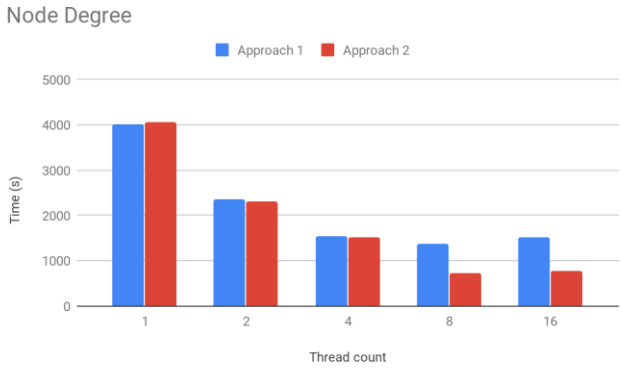

Figure 3: Node Degree results

4.3.4. N-Hop

N-Hop query was selected as a representative of navigation queries. We run this query while varying n from 1 to 5. We notice that the performance improves as the number of threads increases until a certain limit after which the improvement becomes small. Also, because twitter data is a highly connected graph, the edges connecting to new nodes become scattered all over the edges grid and covers nearly every cell after a certain number of hops. In our experiment, we found that the edges cover over 90% of the cells in the third hop (2064187 neighbors), and 100% of cells in the 4th hop (24408042 neighbors). The following charts show performance when n=4 while changing thread count (Figure 4), as well as on fixed

thread count of 8 while changing n (Figure 5).

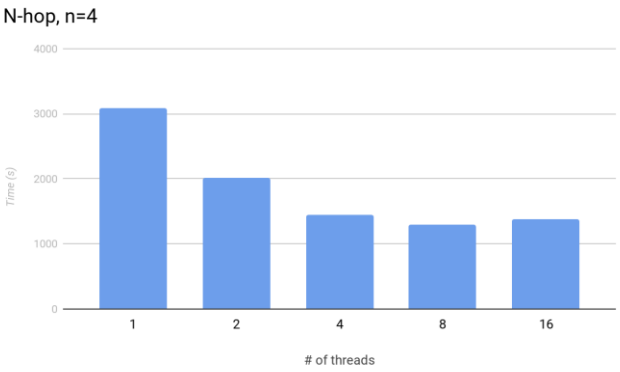

Figure 4: N-hop, n=4

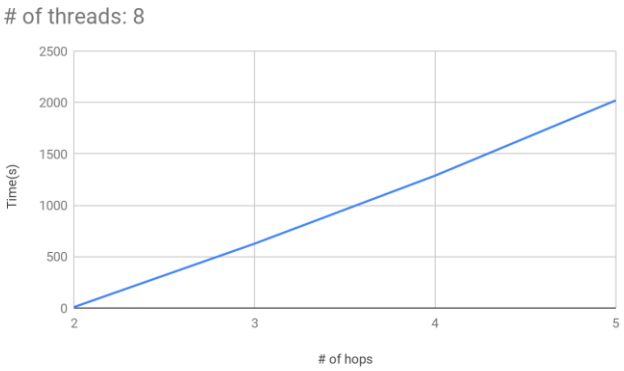

Figure 5: N-hop, thread count=8

5. Conclusion

In this paper, we proposed a new graph database systems that can efficiently process both navigation queries and global graph processing. Graph processing has these two main workloads with different characteristics. The global graph algorithms require sequential access with less cache when dealing with large graphs, especially, at early iterations. Our solution exploits the sequential access bandwidth in a similar way to a cache-oblivious data structures [11, 12]. On the other hand, the graph navigation queries cause random access problems. If edge streaming systems like X-Stream and GridGraph are used for such queries, they require an excessive amount of I/O to perform. Therefore, our solution implements additional edge pointers which become useful to help locating the required edges.

Balancing both workloads simultaneously can be achieved by adapting data structure with edge linkage data similar to Neo4j in a data layout that exploits sequential access similar to GridGraph. The preliminary experimental results indicated the potential advantages of our system.

In our future work, we plan to optimize the pre-processor along with query engines with schedulers, and study the effect of the various memory parameters on the system as a whole, followed by comparison to other out-of-core and distributed systems along with analysis. We also want to implement our navigation query engine and compare to Neo4j and possible other out-of-core edge streaming systems to measure the balanced performance for two types of graph processing.

Acknowledgements

This work was partly supported by JSPS KAKENHI Grant Numbers 18H03242, 18H03342, 19H01138.

References

[1] 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 (SIGMOD '10). Association for Computing Machinery, New York, NY, USA,135–146.DOI:https://doi.org/10.1145/1807167.1807184

[2] Joseph E. Gonzalez and Yucheng Low and Haijie Gu and Danny Bickson and Carlos Guestrin. 2012. PowerGraph: Distributed Graph-Parallel Computation on Natural Graphs. Presented as part of the 10th USENIX Symposium on Operating Systems Design and Implementation (OSDI 12) USENIX. 17--30.

[3] 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 (SOSP '15). Association for Computing Machinery, New York, NY, USA, 410–424. DOI:https://doi.org/10.1145/2815400.2815408

[4] Aapo Kyrola and Guy Blelloch and Carlos Guestrin. 2012. GraphChi: Large-Scale Graph Computation on Just a PC.Presented as part of the 10th USENIX Symposium on Operating Systems Design and Implementation (OSDI 12) USENIX. 31--46.

[5] GONZALEZ, J. E., XIN, R. S., DAVE, A., CRANKSHAW, D., FRANKLIN, M. J., AND STOICA, I. GraphX: Graph processing in a distributed dataflow framework. In Proceedings of the Conference on Operating Systems Design and Implementation (2014), USENIX Association, pp. 599–613.

[6] KHAYYAT, Z., AWARA, K., ALONAZI, A., JAMJOOM, H., WILLIAMS, D., AND KALNIS, P. Mizan: A system for dynamic load balancing in large-scale graph processing. In Proceedings of the European Conference on Computer Systems (2013), ACM, pp. 169–182.

[7] ZHU, X., HAN, W., AND CHEN, W. GridGraph: Large-scale graph processing on a single machine using 2-level hierarchical partitioning. In Proceedings of the Usenix Annual Technical Conference (2015), USENIX Association, pp. 375–386.

[8] CHEN, R., SHI, J., CHEN, Y., AND CHEN, H. PowerLyra: Differentiated graph computation and partitioning on skewed graphs. In Proceedings of the European Conference on Computer Systems (2015), ACM, pp. 1:1–1:15.

[9] H. Huang and Z. Dong, "Research on architecture and query performance based on distributed graph database neo4j," in Consumer Electronics, Communications and Networks (CECNet), 2013 3rd International Conference on, Nov 2013, pp. 533–536.

[10] 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 (SOSP '13). Association for Computing Machinery, New York, NY, USA, 472–488. DOI:https://doi.org/10.1145/2517349.2522740

[11] M. A. Bender, G. S. Brodal, R. Fagerberg, R. Jacob, and E. Vicari. Optimal sparse matrix dense vector multiplication in the I/O-model. In Proceedings of the ACM symposium on Parallel Algorithms and Architectures, pages 61–70. ACM, 2007.

[12] M. Frigo, C. E. Leiserson, H. Prokop, and S. Ramachandran. Cache-Oblivious algorithms. In Proceedings of the Annual symposium on Foundations of Computer Science, pages 285–298. IEEE Computer Society, 1999.

[13] W. Han, K. Li, S. Chen and W. Chen, "Auxo: a temporal graph management system," in *Big Data Mining and Analytics*, vol. 2, no. 1, pp. 58-71, March 2019. doi: 10.26599/BDMA.2018.9020030

[14] Zhu, X., Chen, W., Zheng, W., & Ma, X. (2016). Gemini: A computation-centric distributed graph processing system. In 12th {USENIX} Symposium on Operating Systems Design and Implementation ({OSDI} 16) (pp. 301-316).

[15] Maass, S., Min, C., Kashyap, S., Kang, W., Kumar, M., & Kim, T. (2017, April). Mosaic: Processing a trillion-edge graph on a single machine. In Proceedings of the Twelfth European Conference on Computer Systems (pp. 527-543). ACM.

[16] S. Ghemawat et al. LevelDB. https://code.google.com/p/leveldb/.

[17] Anastassia Ailamaki, David J. DeWitt, and Mark D. Hill. 2002. Data page layouts for relational databases on deep memory hierarchies. The VLDB Journal 11, 3 (November 2002), 198–215. DOI:https://doi.org/10.1007/s00778-002-0074-9

[18] Kwak, H., Lee, C., Park, H., & Moon, S. (2010, April). What is Twitter, a social network or a news media?. In Proceedings of the 19th international conference on World wide web (pp. 591-600). AcM.

[19] Manish R Jain et al. DGraph https://github.com/dgraph-io/dgraph

[20] Deutsch, A., Xu, Y., Wu, M. and Lee, V., 2019. TigerGraph: A Native MPP Graph Database. arXiv preprint arXiv:1901.08248.