Abstract

With the magnitude of graph-structured data continually increasing, graph processing systems that can scale-out and scale-up are needed to handle extreme-scale datasets. While existing distributed out-of-core solutions have made it possible, they suffer from limited performance due to excessive I/O and communication costs.

We present DFOGraph, a distributed fully-out-of-core graph processing system that applies and assembles multiple techniques to enable I/O- and communication-efficient processing. DFOGraph builds upon two-level partitions with adaptive compressed representations to allow fine-grained selective computation and communication. Our evaluation shows DFOGraph outperforms Chaos and HybridGraph significantly (>12.94× and >10.82×) when scaling out to eight nodes.

1 Introduction

Recent Internet-scale graphs can have hundreds of billions of vertices and trillions of edges [6]. Analysis of these massive graphs can be challenging, and recent works include scale-up and scale-out solutions. Scale-up ones process the graph on a single machine using a disk or an array of disks. Scale-out ones include distributed in-memory and out-of-core systems. Although distributed in-memory systems have excellent performance, they require a large number of machines and incur high costs when processing these large graphs. In contrast, distributed out-of-core systems are more cost-efficient.

We propose DFOGraph, a novel distributed out-of-core graph processing system targeting high-speed NVMe SSDs and networks, featuring:

- Fully out-of-core support. Vertex data may not fit in the memory. Previous system Chaos [8] also lies in this category, while some out-of-core systems like HybridGraph [9] and TurboGraph++ [4] do not.
- Relaxed bandwidth assumption. DFOGraph only requires the network bandwidth comparable to per-node disk throughput to scale out. In contrast, Chaos requires the network to outstrip the aggregate disk bandwidth of the whole cluster.

To achieve the goal, we design DFOGraph as follows:

- The core design is vertex-centric [7] push abstraction and two-level partitioning. Pushing enables various optimizations. Partitions narrow the random access span and make pushing practical for fully-out-of-core cases without touching excessive on-disk pages.
- The design enables effective I/O and communication optimizations. CSR (compressed sparse row) or DCSR (doubly-compressed sparse row) are adaptively chosen

CCS Concepts: • Computing methodologies → Distributed algorithms.

Keywords: Graph Analysis Frameworks, Distributed Systems
for edge representation of each chunk, reducing the space consumption and the I/O cost. Messages are efficiently filtered, and only needed ones are sent on the wire to reduce the network traffic, resulting in relaxed bandwidth assumptions.

- We develop multiple adaptive strategies of communication. Operations related to disk or network are decomposed and pipelined, which mostly hides the extra latency caused by optimizations, and helps better utilize and trade-off among the CPU, network, and I/O.

2 Core Design

The core design includes vertex-centric push abstraction and two-level (inter-node and intra-node) partitioning.

- Inter-node partitioning. Each partition contains vertices of continuous IDs, and we try to make a balanced weight of \((P \times |V| + |E|)\) for each partition, where \(P\), \(V\), and \(E\) denote the number of partitions, the vertex set, and the edge set of the partition, respectively.

- Intra-node batching. In each machine, we split vertices to batches of fixed size. Random accesses to vertex data are limited to the span of one batch, which is critical for fully-out-of-core cases to eliminate random accesses to disks. Each edge chunk stores edges from the same source partition to the same destination batch.

Corresponding to the two levels, DFOGraph passes each message inter-node and dispatches it intra-node to batches that need it. The communication contains four phases:

1. Generating. Each batch interacts with its vertex data and saves the generated messages to disk.
2. Inter-node passing. Each node sends messages into buffers of each peer’s memory.
3. Intra-node dispatching. Each node dispatches the messages to its batches, resulting in a file storing needed messages for each batch.
4. Processing. Each batch does computation and interacts with vertex data, based on its edges and messages.

3 Optimizations

- Adaptive edge representations. Each chunk is stored as DCSR in \(O(|E|)\) space, and optionally CSR if its space does not exceed \(32|E|\). The representation to use is decided during message processing, based on the estimated cost given the number of incoming messages.

- Adaptive message dispatching strategies. Message dispatching is either sequential or parallel based on the CPU load. Dispatching uses an auxiliary graph, storing the destination batches of each source vertex, which also follows the adaptive edge representations. DFOGraph may choose not to dispatch if the incoming messages are fewer than the dispatching cost.

- Message filtering. Before sending messages to a peer, a node eliminates unneeded messages (whose source vertex does not have outgoing edges to the peer). DFOGraph may choose not to filter if its cost is too high.

- Pipelining. We decompose and pipeline operations of disk or network, hiding the latency of expensive steps such as message filtering. The communication is also pipelined – after a batch of messages completes one phase, it enters the next phase if resources are ready. Sending messages to different peers can be parallel as no data race exists, which happens if extra network bandwidth is available to reduce the latency.

4 Experiments

Table 1 shows the five graphs used for evaluation. Table 2 compares DFOGraph, Chaos, and HybridGraph, running four algorithms with eight AWS EC2 i3en.3xlarge instances. DFOGraph significantly outperforms Chaos (>12.94x) and HybridGraph (>10.82x) on the three smaller graphs. For the trillion-edge graph KRON-38 (fully-out-of-core – 2 TB vertex data vs. 746 GB aggregate memory), we only attempted one PR iteration. Chaos did not complete loading and one iteration in 24 hours, while DFOGraph preprocessed it in 6.51 hours and finished one iteration in 7.36 hours.

To evaluate the effect of intra-node batching to narrow the random accessing span, we ran PR on KRON-34 with four nodes. The memory is enough or insufficient\(^1\) to store vertex data. Table 3 shows the comparison between DFOGraph with a modified version without intra-node batching\(^2\). Given insufficient memory, DFOGraph without batching had massive random accesses to the disk and did not finish one iteration in 6 hours, even though the vertex data is only 1/3 larger than the memory. With batching, DFOGraph performed one iteration in 23 minutes. Batching brings only 8% overhead in the semi-out-of-core case, and enables efficient fully-out-of-core processing.

To measure the I/O and communication efficiency, we ran five PR iterations using DFOGraph and Chaos on RMAT-32 with eight nodes, recording the disk and network traffic

---

\(^1\) Extra memory is locked in a sleeping process.

\(^2\) There are fewer edge chunks without the batches, each storing edges from the same source partition and to the same destination partition (rather than the same destination batch). No message dispatching happens, and threads of each node process messages in parallel. Vertex data is memory-mapped and operations on it need to be atomic.

---

| Graph      | \(|V|/10^6\) | \(|E|/10^9\) | Size/GB |
|------------|-------------|-------------|---------|
| twitter-2010 [1, 2] | 41.7 | 1.47 | 10.9 |
| uk-2014 [1, 2] | 787.8 | 47.61 | 354.7 |
| RMAT-32 [3] | 4 295.0 | 68.72 | 1 024.0 |
| KRON-34 [5] | 17 179.9 | 68.72 | 1 024.0 |
| KRON-38 [5] | 274 877.9 | 1 099.51 | 16 384.0 |

Table 1. Graph datasets for experiments.
We would like to thank the anonymous reviewers for their helpful comments and suggestions. This material is based upon work supported by the National Science Fund for Distinguished Young Scholars under Grant No. 61525202. Wenguang Chen is the corresponding author.

## References

[1] Paolo Boldi, Marco Rosa, Massimo Santini, and Sebastiano Vigna. 2011. Layered Label Propagation: A MultiResolution Coordinate-Free Ordering for Compressing Social Networks. In Proceedings of the 20th international conference on World Wide Web, Sadagopan Srinivasan, Krithi Ramamritham, Arun Kumar, M. P. Ravindra, Elisa Bertino, and Ravi Kumar (Eds.). ACM Press, 587–596.

[2] Paulo Boldi and Sebastiano Vigna. 2004. The WebGraph Framework I: Compression Techniques. In Proc. of the Thirteenth International World Wide Web Conference (WWW 2004). ACM Press, Manhattan, USA, 479–494.

[3] Deepayan Chakrabarti, Yiping Zhan, and Christos Faloutsos. 2004. R-MAT: A recursive model for graph mining. In Proceedings of the 2004 SIAM International Conference on Data Mining. SIAM, 442–446.

[4] Seongyun Ko and Wook-Shin Han. 2018. ShenTu: processing multi-trillion edge graphs on millions of cores in seconds. In Proceedings of the International Conference for High Performance Computing, Networking, Storage, and Analysis. IEEE Press, 56.

[5] Jurij Leskovec, Deepayan Chakrabarti, Jon Kleinberg, and Christos Faloutsos. 2005. Realistic, mathematically tractable graph generation and evolution, using kronecker multiplication. In European conference on principles of data mining and knowledge discovery. Springer, 133–145.

[6] Heng Lin, Xiaowei Zhu, Bowen Yu, Xiongchao Tang, Wei Xue, Wenguang Chen, Lufei Zhang, Torsten Hoefler, Xiaosong Ma, Xin Liu, et al. 2018. ShenTu: processing multi-trillion edge graphs on millions of cores in seconds. In Proceedings of the International Conference for High Performance Computing, Networking, Storage, and Analysis. IEEE Press, 56.

[7] Robert Ryan McCune, Tim Weninger, and Greg Maday. 2015. Thinking like a vertex: a survey of vertex-centric frameworks for large-scale distributed graph processing. ACM Computing Surveys (CSUR) 48, 2 (2015), 25.

[8] Amitabha Roy, Laurent Bindschaedler, Jasminka Malicevic, and Willy Zwaenepoel. 2015. Chaos: Scale-out graph processing from secondary storage. In Proceedings of the 25th Symposium on Operating Systems Principles. ACM, 410–424.

[9] Zhigang Wang, Yu Gu, Yubin Bao, Ge Yu, and Jeffrey Xu Yu. 2016. Hybrid pulling/pushing for I/O-efficient distributed and iterative graph computing. In Proceedings of the 2016 International Conference on Management of Data. ACM, 479–494.