A Packed-Memory Data Structure for Distributed Graph Streaming

Rohit Agarwal[a], Alec James[a], Joshua You[a], Aydin Buluc[a, b]
[a]UC Berkeley, [b]Lawrence Berkeley National Laboratory

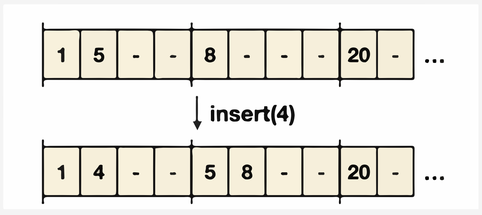
Motivation

- Extremely large sparse dynamic graph workloads are common for many modern database problems.
- Thus, it is necessary to have graph data structures that have high update and query throughput while also operating in a distributed manner, where many machines can access a shared structure.
- Graph algorithms like breadth-first search use locality of edges incident to the same vertex to maximize performance.

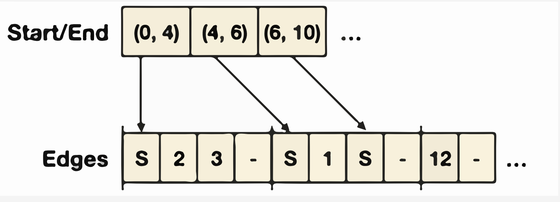
Can we maintain a sparse graph data structure, distributed amongst several servers, that can remain consistent in the face of concurrent queries, updates, and common graph processing algorithms?

Background

A Packed-Memory Array (PMA) [1] is a fast set data structure which supports polylogarithmic time and cache-friendly database operations, such as updates, point queries, and range queries.



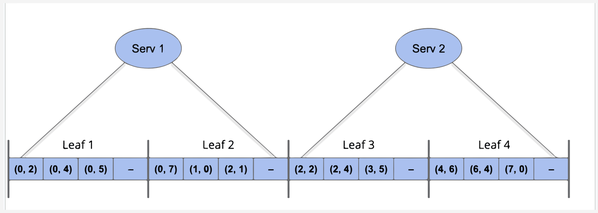
A Packed-Compressed Sparse Row Matrix (PCSR) [2] utilizes the cache locality of the PMA to execute fast graph search algorithms. PCSRs were previously parallelized in shared memory model as PPCSRs [3].



Data Structure Design

Our data structure, the DistPCSR, answers queries and stores the graph with a distributed network of servers.

- Each server owns a particular subrange of the edges; we keep a PCSR storing edges in that subrange (and put a maximum on the size of this PCSR).
- On **query** or **insert**, we route a request to the correct server, based on the requesting server's view of subranges.
- If any PCSR gets too full, we have it redistribute with its neighbors before handling more inserts.
- Inserts are received in a queue; to process an insert one checks if it needs to be re-routed (perhaps the subrange has changed), otherwise, it adds it to a local PCSR.



Fixed vs. Active Redistribution

On a redistribute, there are two strategies we can use:

- Give up and hope we never see this case in practice. Thus, the subranges will stay *fixed*.
- Alternatively, we can allow *active* subranges that can change as the data structure evolves. To do a redistribute from server S:
  - S figures out how many neighbors it needs to redistribute with and notifies them.
  - Dynamically change S and S's neighbors' subranges according to the element redistribution.
  - Propagate these changes to the other servers asynchronously.
  - Use a global lock; only one global redistribute allowed at a time.
  - After a redistribute, a request may need to be re-forwarded to its final destination.

In the active case, we were able to show the following guarantee:

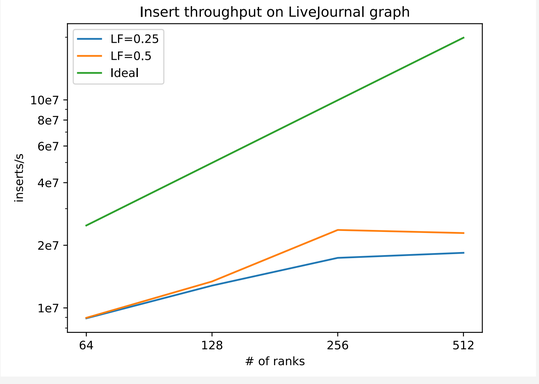
The data structure does not drop, duplicate, or infinitely route any edges.

UPC++ Implementation

- Queries and inserts are both implemented using the Remote Procedure Call (RPC) mechanism in UPC++.
- The query RPC is straightforward, as it simply performs a local query on a remote machine then returns the result.
- Inserts are less straightforward, since any insert could trigger a call to redistribute, which requires the use of synchronization operations not available within an RPC function. Thus, we implemented inserts as an RPC which would place a request within the remote machine's insert queue, which it would then flush within its own progress loop.
- To check consistency, we can add UPC++ barriers to clearly separate insert and query phases.

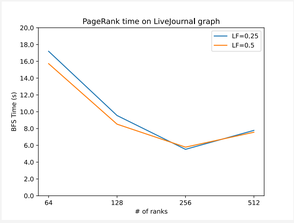
Testing Insert Throughput

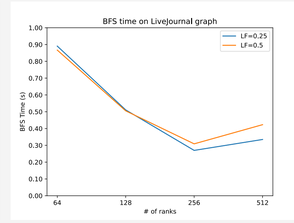
We compared several settings of our data structure (LF is the load factor, which is the maximum fraction of the global data structure that is allowed to be full) against an idealized model on the LiveJournal social network graph, with roughly 8.5 million edges (identical to [2]).



Running Graph Algorithms

- Breadth-first search (**BFS**) and PageRank (**PR**) are two graph algorithms that are commonly run in a distributed setting and used as benchmarks for graph processing systems.
- Generally, both algorithms have low computational intensity, and thus the latency and bandwidth of accessing the data structure cannot be hidden.
- The distributed PCSR fares even worse, as many of its accesses incur an additional cost of network serialization / deserialization, which can be an order of magnitude slower than accessing main memory.





Conclusion and Future Directions

Some ideas we still need to explore to improve performance:

- *Fine-grained locking* will ensure the single global redistribute lock is not a serial bottleneck. On more servers, this would quickly become an issue that prevents further scaling. May need a system similar to that used in [2] to avoid deadlocks between different ranks trying to redistribute simultaneously.
- *2-dimensional distribution* wherein each server is in charge of a rectangular block of the adjacency matrix, since in combination with graph partitioning, it can dramatically reduce the cost of communication for sparse matrix-vector operations.

We demonstrate the usefulness of Packed-Memory Array Data Structures in the distributed memory model.

Acknowledgements

We submitted this as our CS267 Project. Thank you to the course staff for acquainting us with UPC++ and other frameworks. In addition, thank you to Helen Xu and Brian Wheatman, the authors of previous packed-memory data structures, for sharing some serial PMA code and realistic graph workloads for us to use.

References

1. M. A. Bender and H. Hu. *An adaptive packed-memory array*, ACM TODS, 32 (2007), p.26.
2. B. Wheatman and H. Xu, *Packed compressed sparse row: A dynamic graph representation*, in 2018 IEEE HPEC, 2018.
3. B. Wheatman and H. Xu. *A parallel packed memory array to store dynamic graphs*. In 2021 ALENEX, pages 31–45.