The paper presents Metall, a persistent memory allocator, and its evaluation against existing memory management systems. The focus of the paper is to address the need for persistently storing data beyond the single process lifecycle and provide a solution that can handle data analytics effectively.

The key ideas and contribution of Metall are its use of NVRAM, memory-mapped file mechanism, and support for persistence and snapshots. The primary result of the evaluation shows that Metall outperforms Boost. Interprocess and memkind (PMEM kind) in terms of performance.

One limitation of Metall is that it only supports single process use and is not designed for multi-process data sharing. Although multiple processes can open the same datastore in read-only mode, the data is not shared between processes. Another limitation is the restriction of references, virtual functions, and virtual base classes in persistent memory.

In summary, Metall is a promising solution for persistently storing data in NVRAM, providing efficient memory management and a snapshot feature. However, further research is needed to address the limitations and improve its compatibility with various data structures.

The authors then describe the different machine configurations used for their experiments. The first machine, named "EPYC," was optimized with a PCIe NVMe SSD and page cache behavior for improved performance. The second machine, named "Optane," was configured with an Intel Optane DC Persistent Memory device and used in both the App Direct Mode and ext4 filesystem DAX mode. The third machine, named "Corona," was one of many nodes in a cluster, connected to both Lustre and VAST parallel file systems, which demonstrated varying performance characteristics for different types of I/O. In their paper, the authors provide specifications for each machine.

In terms of persistent memory allocators, the authors explain the features of Boost.Interprocess, PMEM kind, and Ralloc, and specify the libraries and versions used. The author notes a modification to PMEM kind for better performance on the Optane machine, and why Ralloc was only used on that particular machine. They conclude by mentioning a custom STL-compatible allocator class was written using Ralloc.

The authors also detail their process for generating synthetic scale-free graphs for benchmarking purposes using an R-MAT generator. The generated graphs range in size from SCALE 24 to 30 and have vertex IDs scrambled to eliminate unexpected localities. The edges are treated as undirected and the number of edges inserted is calculated using the formula provided. The generation of edges is performed in chunks and the time taken to generate the edges is not included in the reports.

The authors then present the results of their benchmark comparison of three graph implementations: Metall, Boost.Interprocess (BIP), and PMEM kind, on both the EPYC and Optane machines. The results show that, on the EPYC machine, Metall significantly outperforms BIP and PMEM kind, with improvement factors ranging from 7.4 to 10.9x and 2.2 to 2.8x, respectively, at different graph sizes. However, due to DRAM capacity limitations, there is a drop in performance for all implementations at the largest graph size (SCALE 30). Despite this drop, Metall still outperforms BIP and PMEM kind by 11.7x and 48.3x, respectively.

On the Optane machine, the results indicate that Metall has slightly better performance than BIP, with improvement factors of 2.1 to 2.3x. The results also show that Ralloc did not finish processing the largest graph size due to a lack of persistent memory space. The performance of Metall was comparable to both Ralloc and the modified PMEM kind, with PMEM kind and Ralloc being up to 10% and 14% better than Metall, respectively.

The explanation of the performance difference between Boost.Interprocess and Metall is well-reasoned and clearly presented. The authors explain the performance limitations of Boost.Interprocess and how Metall was able to achieve better performance. The comparison of Metall's performance on the EPYC and Optane machines highlights its high portability, with reference to the design strategies from Supermalloc providing credibility to their explanation.

Finally, the authors describe their experiment evaluating the performance of constructing a persistent graph data structure using the Metall tool on Lustre and VAST network file systems. Using real temporal graph datasets from Wikipedia and Reddit and simulating a growing graph by sorting edges by timestamp, the benchmark involves constructing a banked adjacency list incrementally. The authors measure the time per iteration, broken down into ingestion and flush time, on a single node of the Corona cluster. The paper also describes two datasets used for the experiments: the Wikipedia page reference graph with 1.8 billion hyperlink insertions, and the Reddit author-author graph with 4.4 billion comment activities.

a. Topic of the paper: The paper presents Metall, a persistent memory allocator, and its evaluation against existing memory management systems.

b. Novelty or key ideas and contribution: The key ideas and contribution of Metall are its use of NVRAM, memory-mapped file mechanism, and support for persistence and snapshots.

c. Primary (evaluation) result: The primary result of the evaluation shows that Metall outperforms Boost. Interprocess and memkind (PMEM kind) in terms of performance.

d. Limitation and a very brief discussion: One limitation of Metall is that it only supports single process use and is not designed for multi-process data sharing. Another limitation is the restriction of references, virtual functions, and virtual base classes in persistent memory.