System-level Crash Safe Sorting on Persistent Memory

Omri Arad*, Yoav Ben Shimon*, Ron Zadicario* Daniel Waddington§, Moshik Hershcovitch*§, Adam Morrison*
Tel-Aviv University*, IBM Research,§
{omriarad, benshimon2, ronzadicario}@mail.tau.ac.il, daniel.waddington@ibm.com, moshikh@il.ibm.com, mad@cs.tau.ac.il

1 INTRODUCTION

Sorting is a fundamental operation in software systems. An example for that is a prepossessing phase before executing analytical operations.

Persistent memory (PM) is a nonvolatile device with low latency and byte-addressable access. Our experiments used Intel's first commercially available device- Intel Optane DC.

In this paper, we evaluate system-level Crash Safe Sorting trade-offs between performance and persistence (crash recovery), utilizing PM. In our scenario, cloud providers receive log records from machines across their data centers and store them in a high-performance data store with PM. A sorting operation is executed to prepare the data for analysis. We assume that data arrives in chunks, with each chunk containing many items of fixed size. The sorting goal is to sort and merge these chunks into one sorted chunk, which is then partitioned back to the original number of chunks.

We evaluate our sorting algorithm using MCAS (Memory Centric Active Storage) [2], which is a high-performance client-server key-value store explicitly designed for PM.

2 SORTING ALGORITHM

We use the merge-sort algorithm, which is widely used to organize and search for information [1]. This algorithm consists of two stages: **Sorting stage** - Sort the data in each chunk using QuickSort. **Merging stage**- Merge the sorted chunks. The Merge stage consists of multiple "Merge Phases" in which all chunks are merged in pairs, until a sorted chunk containing all data is generated. We integrate the merge-sort algorithm into MCAS using its ADO (Active Data Objects) plugin mechanism. We evaluate different approaches regarding the use of PM. A log maintained on PM is used in crash recovery scenarios:

(1) **DRAM-SORT** - The entire algorithm is executed in DRAM. After a crash, the system needs to restart the sorting algorithm from scratch.
(2) **DRAM-WITH-RECOVERY** - All stages are performed in DRAM, with results backed up to PM after the sorting stage and each merge phase. This enables recovery to the start of the last completed merge phase in case of a crash.
(3) **PM-SORT** - All stages are performed directly on the data stored in PM, enabling recovery to the last completed operation after a crash. The QuickSort is not in place (persisting after each swap is very time-consuming, causing x100 slower performance).

3 EVALUATION

In our benchmark, we generated 512 chunks. Each chunk of size 128MB has items of size 100B with a key of size 10B. The total data is 64 GB. The following table shows the time difference of the sorting approaches. The results are divided to the Sorting Stage and Merging Stage.

Task	Sorting Stage	Merging Stage	Total Time	relative
(1)	289.682s	307.233s	715.517s	1
(2)	352.466s	588.793	1023.222s	1.43
(3)	409.997s	685.558s	1130.683s	1.58

4 CONCLUSION

Sorting fully on PM is only 1.58 slower than sorting on DRAM while giving persistent ability. The results show that the time for persisting the data is a bottleneck in crash safe implementation, and demonstrate the potential for utilizing a trade-off between performance and persistence.

References

[1] Zbigniew Marszałek. 2018. Performance tests on merge sort and recursive merge sort for big data processing. *Technical Sciences/University of Warmia and Mazury in Olsztyn* (2018).
[2] Daniel Waddington, Clem Dickey, Moshik Hershcovitch, and Sangeetha Seshadri. 2021. An architecture for memory centric active storage (MCAS). *arXiv preprint arXiv:2103.00007* (2021).

SYSTOR '22, June 13–15, 2022, Haifa, Israel
© 2022 Copyright held by the owner/author(s).
ACM ISBN 978-1-4503-9380-5/22/06.
https://doi.org/10.1145/3534056.3535008