Aiman Priester

P09 FlatFlash: Exploiting the Byte-Accessibility of SSDs within a Unified Memory-Storage Hierarchy

Summary

With this paper, the authors are introducing a way to essentially overcome scaling issues present in modern day storage called FlatFlash. The paper acknowledges the stark contrast in cost and limitations of DRAM and SSDs when scaling, citing the cost difference and physical limitations of the host, in this case, the number of slots. Authors aimed to rethink the design of a unified storage system to overcome execution delays, trashing and declining SSD performance.

FlatFlash is designed in a way that the CPU has "direct" access to the DRAM stored in the SSD while maintaining an Adaptive Page Promotion Mechanism to determine which flash pages are to be stored in DRAM for faster access. A promotion look-aside buffer is implemented to avoid stalls and redirect memory requests for a page to be promoted to physical addresses. In order to maintain persistence, FlatFlash uses a battery backed DRAM in order to keep necessary data.

Strengths

FlatFlash makes quite a few changes. Notably, the paper suggests that the SSD's page cache should use Re-Reference Interval Prediction (RRIP) in replacement to the conventional set-associative cache. Since the SSD will be used as memory, access patterns will be more random and locality is more often than not filtered by the upper levels of memory. By treating the SSD this way, memory access costs will reduce and in turn, speed up the total time taken to execute.

By essentially doing byte-mutation, an entire page does not have to be sent over to be changed. By the use of the battery-backed DRAM, small changes to a page can be made solely by changing a few bytes within the page, instead of taking the entire page, rewriting it, and placing the information back in memory. This idea clears up bandwidth and allows for more changes to happen at the same time.

Weaknesses

As with most things, there is always a tradeoff. Since the paper introduces the unification of an SSD with DRAM, when scaling this system, the dependencies of an additional hardware system comes at a risk. In the event of a power fault or a damaged system, it is likely that an entire system that is using this idea would fail. Giving up data security for speed is something implementers would have to rationalize. Granted, this system should be persistent due to it being battery backed, however, it does not solve the risk of a total system failure.

Unresolved Issues

The performance increase is tested by running an emulation. This effectively removes most real-world problems in implementation. The paper does suggest that this is taken into account, however as seen in previous papers, emulations do cause uncertainties before being ported into real-world tests. It would be best to see this run to expose some of the faults that were unconsidered during emulation.

Discussion

Since the paper introduces the idea of mutating a page by changing data in the byte level, what is safeguarding this data from being erroneously mutated while being stored in memory?