CS561 Spring 2022 - Research Project

Title: Implementation of a variable-size bufferpool

Background: The bufferpool of a DBMS always maintains *in memory* a set of pages, and allows incoming page requests (writes or reads) to be served without accessing the disk. The purpose of the bufferpool is to improve database system performance. Since data can be accessed much faster from memory than from disk, the fewer times the database manager needs to access disk, the better the performance. Traditional page replacement policies exchange one disk write (write-back during eviction) for one disk read (fetching in the bufferpool the page that caused the miss). This design makes an inherent assumption that the underlying storage has symmetric read and write performance. However, most modern storage devices like solid-state disk (SSDs) possess an inherent read/write asymmetry (wrties are slower) because of their physical organization. At the same time, these devices have immense parallelism opportunities. In light of these new characteristics (asymmetric read/write cost), it is not fair to exchange one write with one read. However, it is not possible to exchange multiple reads for one write, because this would grow the bufferpool perpetually. Hence, a variable size bufferpool might be the solution. The primary idea is to have a bufferpool of a specified size and a possible extended size. During evicting dirty page, the bufferpool may need to grow. However, there must be a way to revert back to its original size periodically to ensure that it remains at its normal size most time.

Objective: The objective of the project is to develop a variable size bufferpool in presence of asymmetric read/write cost.

(a) Get familiarized with modern storage device's properties like read/write asymmetry, concurrency.
(b) Design a variable size bufferpool which is optimized for such modern storage devices. The main idea is that when evicting a dirty page, the bufferpool can get an extended size, however, it should operate in its regular size most time.

[1] Guy E Blelloch, Jeremy T Fineman, Phillip B Gibbons, Yan Gu, and Julian Shun. 2016. Efficient Algorithms with Asymmetric Read and Write Costs. In Proceedings of the Annual European Symposium on Algorithms (ESA). 14:1—-14:18.
[2] Stratis D. Viglas. 2012. Adapting the B +-tree for asymmetric I/O. In Lecture Notes
in Computer Science (including subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics), Vol. 7503 LNCS. 399–412.
[3] Feng Chen, Rubao Lee, and Xiaodong Zhang. 2011. Essential roles of exploiting internal parallelism of flash memory based solid state drives in high-speed data

processing. In Proceedings of the IEEE International Symposium on High Performance Computer Architecture (HPCA). 266–277.