HARVARD
School of Engineering
and Applied Sciences

Data Systems Laboratory
School of Engineering and Applied Sciences
Harvard University
MD 139, 33 Oxford St
Cambridge, MA 02138
http://daslab.seas.harvard.edu/classes/cs265

CS265 Spring 2016 development project

Building and Parallelizing Log-Structure Merge Trees for Key-Value Stores

Keywords

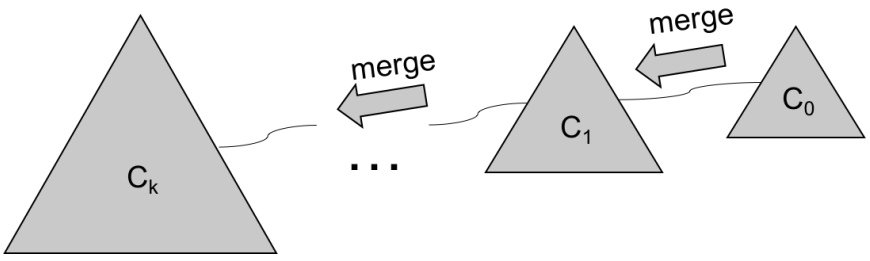
indexing, key-values stores, log-structure merge trees, parallel execution, multi-threading execution

Logistics

Systems projects for CS265 are tailored to provide background on state-of-the-art systems, data structures and algorithms. They include a design component and an implementation component in C or C++, dealing with low level systems issues such as memory management, hardware conscious processing, parallel processing, managing read/write tradeoffs and scalability. All Systems projects will be done individually, i.e., each student will have to work on the project on their own. This year's project is about designing a Log-Structure Merge (LSM) Tree. This is a focused project that while is not extremely heavy in terms of how much code you have to write, it will bring you against basic modern system design issues and tradeoffs.

Project

Indexing with Log-Structure Merge Trees [1, 5] is a core component of many modern key-value stores [2, 3, 4]. In this project you are expected to build a Log-Structured Merge Tree that can accommodate fast reads and writes.



A Log-Structured Merge-tree (LSM-tree) is a storage-aware data structure designed to provide low-cost indexing for a file experiencing a high rate of record inserts (and deletes) over an extended period. LSM-Trees defer and batches index updates cascading the changes from a small (typically memory-based) component to one or more larger (typically disk-based) components. During that process all values are continuously available apart from small locking periods and the updates cost is greatly reduced.

Today, LSM Trees are used in a variety of systems, typically key-value stores, as a means of fast indexing of heavily updated data. Google's BigTable [2], Amazon's Cassandra [3] and a number of research prototypes [4, 5, 6, 7] have implemented their own version of LSM Trees.

The project is divided in two parts. Each part will have a first milestone where you will discuss your design and development strategy with the teaching stuff. The first part of the project is about designing the basic structure of an LSM tree for reads and writes, while the second part is about designing and implementing the same functionality in a parallel way so we can support multiple concurrent reads and writes.

Expected deliverable

The final deliverable of this project is an LSM Tree implementation (both single threaded and parallel) with a number of tunable parameters: number of levels, size ratio between levels, variable storage layer used for each level (RAM, SSD, HDD), different merging strategies, and potentially additional tuning parameters the students design. The final LSM Tree is expected to support high update rate (in the order of 100K-1M updates per second for flash storage and 1K-10K of updates per second for HDD storage), while at the same time provide efficient reads (in the order of 1K-5K reads per second for flash storage and 20-100 reads per second for HDD storage). Key design decisions include, among others: how to protect the levels during merging, how to do the merging, and how to avoid reading unnecessary levels (if any). The parallel LSM Tree is expected to *scale with number of cores*, that is, we expect to see that as the number of cores used increases the performance of the tree is precipitously increasing.

Demo

Students are expected to create a demonstration scenario where they will show the behavior of their LSM Trees. In essence we expect to see in the demo different datasets and workloads that will lead to different behavior of the LSM Tree in order to get a clear idea about its behavior. For example, a workload consisting of skewed updates (inserting values from a small set of values again and again) and a workload consisting of evenly distributed updates, will lead to different behavior of the system. Combining this, with either skewed or evenly distributed reads we get four workloads that will behave very differently. We expect to see such cases in the demo in order to understand the trade-offs of the LSM Tree designed.

Timeline

1. Familiarization with LSM-Trees and variations (1 weeks)
2. Design of LSM-Tree and development plan – initial testing and development (2 weeks)
3. Development of an LSM Tree (3 weeks)
4. Design document describing LSM tree and the plan for parallelization (March 11th)
5. Feedback phase (1 week)
6. Refining LSM Tree and parallelization based on feedback (2 weeks)
7. Finalizing development of the parallel LSM Tree (3 weeks)
8. Presentation of the project (early May, **date TBD**)

References

[1] Patrick E. O'Neil, Edward Cheng, Dieter Gawlick, Elizabeth J. O'Neil: **The Log-Structured Merge-Tree (LSM-Tree)**. Acta Inf. 33(4):351-385 (1996) [http://paperhub.s3.amazonaws.com/18e91eb4db2114a06ea614f0384f2784.pdf]

[2] Fay Chang, Jeffrey Dean, Sanjay Ghemawat, Wilson C. Hsieh, Deborah A. Wallach, Michael Burrows, Tushar Chandra, Andrew Fikes, Robert Gruber: **Bigtable: A Distributed Storage System for Structured Data** (Awarded Best Paper!). OSDI 2006:205-218 [http://static.googleusercontent.com/media/research.google.com/en//archive/bigtable-osdi06.pdf]

[3] Avinash Lakshman, Prashant Malik: **Cassandra: a decentralized structured storage system**. Operating Systems Review (SIGOPS) 44(2):35-40 (2010) [https://www.cs.cornell.edu/projects/ladis2009/papers/lakshman-ladis2009.pdf]

[4] Xingbo Wu, Yuehai Xu, Zili Shao, Song Jiang: **LSM-trie: An LSM-tree-based Ultra-Large Key-Value Store for Small Data Items**. USENIX ATC 2015:71-82 [http://www.ece.eng.wayne.edu/~sjiang/pubs/papers/wu15-lsm-trie.pdf]

[5] Russell Sears, Raghu Ramakrishnan: **bLSM: a general purpose log structured merge tree**. SIGMOD 2012:217-228 [http://www.eecs.harvard.edu/~margo/cs165/papers/gp-lsm.pdf]

[6] Pradeep Shetty, Richard P. Spillane, Ravikant Malpani, Binesh Andrews, Justin Seyster, Erez Zadok: **Building workload-independent storage with VT-trees**. FAST 2013:17-30
[https://www.usenix.org/system/files/conference/fast13/fast13-final165_0.pdf]

[7] Peng Wang, Guangyu Sun, Song Jiang, Jian Ouyang, Shiding Lin, Chen Zhang, Jason Cong: **An efficient design and implementation of LSM-tree based key-value store on open-channel SSD**. EuroSys 2014:16:1-16:14
[http://www.ece.eng.wayne.edu/~sjiang/pubs/papers/wang14-LSM-SDF.pdf]