Dandelion Hashtable: Cracking the Billion Requests Barrier—Effortlessly

Antonios Katsarakis*, Vasilis Gavrielatos*, Nikos Ntarmos
Huawei Research, *equal contribution

Abstract

This paper presents DLHT, a concurrent in-memory hashtable. Despite efforts to optimize hashtables, that go as far as sacrificing core functionality, state-of-the-art designs still incur multiple memory accesses per request and block request processing in three cases. First, most hashtables block while waiting for data to be retrieved from memory. Second, open-addressing designs, which represent the current state-of-the-art, either cannot free index slots on deletes or must block all requests to do so. Third, index resizes block every request until all objects are copied to the new index. Defying folklore wisdom, DLHT forgoes open-addressing and adopts a fully-featured and memory-aware closed-addressing design based on bounded cache-line-chaining. This design ❶ offers lock-free operations and deletes that free slots instantly, ❷ completes most requests with a single memory access, ❸ utilizes software prefetching to hide memory latencies, and ❹ employs a novel non-blocking and parallel resizing. In a commodity server and a memory-resident workload, DLHT surpasses 1.6B requests per second and provides 3.5× (12×) the throughput of the state-of-the-art closed-addressing (open-addressing) resizable hashtable on Gets (Deletes).

1 Introduction

Concurrent in-memory hashtables are essential and versatile data structures in the modern cloud. They are responsible for storing and accessing large amounts of data in main memory via thread-safe *Get*, *Put*, *Insert*, and *Delete* requests. To ensure requests complete rapidly as the dataset expands, hashtables must be also able to efficiently *Resize* their index. In-memory hashtables serve a wide spectrum of applications, including in-memory storage, online services, caching, key-value stores, and transactional databases [1, 3, 4, 7, 10].

To meet the ever-growing performance demands [1, 5], state-of-the-art hashtables from industry and academia offer designs that attain close to a billion requests per second on a single server [2, 8, 9, 11, 12, 14, 15]. Problematically, their evaluation hints that such high throughput is reachable only under *cache-resident* workloads where accesses are served by hardware caches and seldom reach main memory – i.e., due to small datasets [2], data partitioning [11, 14, 15], or highly skewed accesses [8, 9, 12]. So we pose the following question: *Can state-of-the-art in-memory hashtables attain a billion requests per second under a memory-resident workload?*

To answer this question, we evaluate eight state-of-the-art designs over a *memory-resident* workload of 100 million objects accessed uniformly on a commodity server. As shown

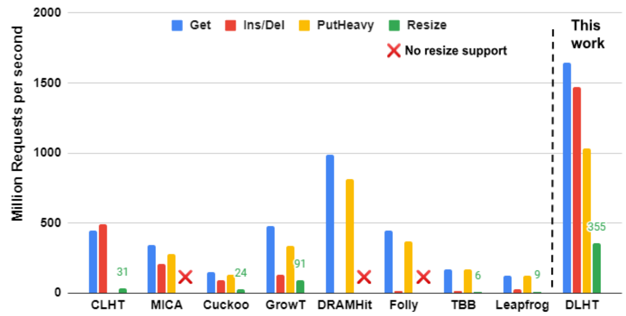

Figure 1. *Throughput of state-of-the-art hashtables and DLHT with 64 threads in a memory-resident workload (100M objects).*

in Figure 1, almost all hashtables are more than 2× slower than a billion requests per second. The most recent work, DRAMHiT [13], is the only one close to the target (in Gets), but its open-addressing design hinders Deletes and Resizes. Hence, achieving a billion requests per second without forfeiting core functionality on a commodity server remains a challenge for memory-resident workloads.

In a deeper inspection (detailed in our poster), state-of-the-art designs offer lock-free accesses but sacrifice core functionality, incur multiple memory accesses per request, and block processing in three cases. First, most hashtables stall processing on every request when accessing memory. Second, open-addressing designs offer impaired Deletes that either cannot reclaim index slots or must cease processing and rebuild the entire index to do so. Third, those that support index Resizes block every request until all objects are copied to the new index. These stalling factors impede the throughput of state-of-the-art hashtables, rendering them *practically blocking* under memory-resident workloads.

In this work, we introduce DLHT, a concurrent hashtable that is *memory access aware* and *practically non-blocking* (i.e., alleviates stalling) to transcend a billion requests per second in memory-resident workloads. Defying folklore wisdom [12, 13, 16], DLHT forgoes open-addressing and adopts a closed-addressing approach. Its design is based on bounded cache-line-chaining and has the following features. First, it enables lock-free index operations, including deletes with immediate index slot reclamation. Second, it minimizes memory traffic and completes most requests with a single memory access. Third, it exploits software prefetching to overlap the memory latency of a request with productive work on other requests. Finally, it incorporates a novel, non-blocking (but not lock-free) Resize where requests complete with strong consistency while a multi-threaded index migration occurs in parallel.

Unlike state-of-the-art designs that trade core functionality for throughput [2, 6, 9, 12], DLHT provides a complete set of implemented features to accommodate its clients' needs. Beyond core functionality, this includes namespaces, variable-sized key-value pairs, efficient single-thread and hashset variants, as well as pointer APIs that minimize copies.

We extensively evaluate DLHT on a commodity server using micro-benchmarks, sensitivity studies, application examples, and standard single- and multi-key OLTP benchmarks (YCSB, TATP, and Smallbank). We compare DLHT with eight state-of-the-art concurrent in-memory designs. DLHT surpasses 1.6B Get (1.4B Inserts/Deletes, 1B Gets/Puts) requests per second. This is more than 3.5× (3×, 2.7×) the performance of the fastest closed-addressing design and an order of magnitude faster Deletes than open-addressing designs. Finally, the parallel and non-blocking resize of DLHT allows for a population that is 3.9× faster than the state-of-the-art.

References

[1] Nathan Bronson, Zach Amsden, George Cabrera, Prasad Chakka, Peter Dimov, Hui Ding, Jack Ferris, Anthony Giardullo, Sachin Kulkarni, Harry Li, Mark Marchukov, Dmitri Petrov, Lovro Puzar, Yee Jiun Song, and Venkat Venkataramani. 2013. TAO: Facebook's Distributed Data Store for the Social Graph. In *Proceedings of the 2013 Conference on Annual Technical Conference (ATC'13)*. USENIX, Berkeley, 49–60.

[2] Tudor David, Rachid Guerraoui, and Vasileios Trigonakis. 2015. Asynchronized Concurrency: The Secret to Scaling Concurrent Search Data Structures. In *Proceedings of the Twentieth International Conference on Architectural Support for Programming Languages and Operating Systems (ASPLOS '15)*. Association for Computing Machinery, New York, NY, USA, 631–644.

[3] Giuseppe DeCandia, Deniz Hastorun, Madan Jampani, Gunavardhan Kakulapati, Avinash Lakshman, Alex Pilchin, Swaminathan Sivasubramanian, Peter Vosshall, and Werner Vogels. 2007. Dynamo: Amazon's Highly Available Key-value Store. *SIGOPS Oper. Sys.* 41, 6 (2007), 5–20.

[4] Aleksandar Dragojević, Dushyanth Narayanan, Miguel Castro, and Orion Hodson. 2014. FaRM: Fast Remote Memory. In *11th USENIX Symposium on Networked Systems Design and Implementation (NSDI 14)*. USENIX Association, Seattle, WA, 401–414.

[5] Mostafa Elhemali, Niall Gallagher, Nick Gordon, Joseph Idziorek, Richard Krog, Colin Lazier, Erben Mo, Akhilesh Mritunjai, Somasundaram Perianayagam, Tim Rath, Swami Sivasubramanian, James Christopher Sorenson III, Sroaj Sosothikul, Doug Terry, and Akshat Vig. 2022. Amazon DynamoDB: A Scalable, Predictably Performant, and Fully Managed NoSQL Database Service. In *2022 USENIX Annual Technical Conference (USENIX ATC 22)*. USENIX Association, Carlsbad, CA, 1037–1048. https://www.usenix.org/conference/atc22/presentation/elhemali

[6] Facebook. 2023. Folly: An open-source C++ library developed and used at Facebook. https://github.com/facebook/folly.

[7] Bin Fan, David Andersen, and Michael Kaminsky. 2013. MemC3: Compact and Concurrent MemCache with Dumber Caching and Smarter Hashing. In *Proceedings of the 10th USENIX Conference on Networked Systems Design and Implementation (nsdi'13)*. USENIX Association, Berkeley, CA, USA, 371–384. http://dl.acm.org/citation.cfm?id=2482626.2482662

[8] Bojie Li, Zhenyuan Ruan, Wencong Xiao, Yuanwei Lu, Yongqiang Xiong, Andrew Putnam, Enhong Chen, and Lintao Zhang. 2017. KV-Direct: High-Performance In-Memory Key-Value Store with Programmable NIC. In *Proceedings of the 26th Symposium on Operating Systems Principles (SOSP '17)*. ACM, New York, NY, USA, 137–152.

[9] Sheng Li, Hyeontaek Lim, Victor W. Lee, Jung Ho Ahn, Anuj Kalia, Michael Kaminsky, David Andersen, Seongil O, Sukhan Lee, and Pradeep Dubey. 2016. Full-Stack Architecting to Achieve a Billion-Requests-Per-Second Throughput on a Single Key-Value Store Server Platform. *ACM Trans. Comput. Syst.* 34, 2, Article 5 (April 2016), 30 pages.

[10] Hyeontaek Lim, Dongsu Han, David Andersen, and Michael Kaminsky. 2014. MICA: A Holistic Approach to Fast In-memory Key-value Storage. In *Proceedings of the 11th Networked Systems Design and Implementation (NSDI'14)*. USENIX Association, USA, 429–444.

[11] Clemens Lutz, Sebastian Breß, Steffen Zeuch, Tilmann Rabl, and Volker Markl. 2022. Triton Join: Efficiently Scaling to a Large Join State on GPUs with Fast Interconnects. In *Proceedings of the 2022 International Conference on Management of Data (SIGMOD '22)*. Association for Computing Machinery, New York, NY, USA, 1017–1032.

[12] Tobias Maier, Peter Sanders, and Roman Dementiev. 2019. Concurrent Hash Tables: Fast and General(?)! *ACM Trans. Parallel Comput.* 5, 4, Article 16 (feb 2019), 32 pages.

[13] Vikram Narayanan, David Detweiler, Tianjiao Huang, and Anton Burtsev. 2023. DRAMHiT: A Hash Table Architected for the Speed of DRAM. In *Proceedings of the Eighteenth European Conference on Computer Systems (EuroSys '23)*. Association for Computing Machinery, New York, NY, USA, 817–834.

[14] Stefan Schuh, Xiao Chen, and Jens Dittrich. 2016. An Experimental Comparison of Thirteen Relational Equi-Joins in Main Memory. In *Proceedings of the 2016 International Conference on Management of Data (SIGMOD '16)*. Association for Computing Machinery, New York, NY, USA, 1961–1976.

[15] Panagiotis Sioulas, Periklis Chrysogelos, Manos Karpathiotakis, Raja Appuswamy, and Anastasia Ailamaki. 2019. Hardware-Conscious Hash-Joins on GPUs. In *2019 IEEE 35th International Conference on Data Engineering (ICDE)*.

[16] Alex Stivala, Peter J. Stuckey, Maria Garcia de la Banda, Manuel Hermenegildo, and Anthony Wirth. 2010. Lock-Free Parallel Dynamic Programming. *J. Parallel Distrib. Comput.* 70, 8 (aug 2010), 839–848.