文献综述

计算机学院　硕 1708 班　M201773209 胡楠

摘要

在处理大量文件时，快速查询服务对于提高大型存储系统的整体性能非常重要。由于简单和易用性的突出特点，开放寻址 cuckoo 哈希计划已被广泛用于支持查询服务。不幸的是，传统的方案不足以解决项目插入期间具有无限循环的潜在问题，这降低了查询性能。为了解决这个问题，本文提出了一个名为 SmartCuckoo 的高效的 cuckoo 哈希方案。SmartCuckoo 背后的想法是将散列关系表示为有向伪森林，并使用它来跟踪项目的位置，以准确地预先确定无限循环的发生。SmartCuckoo 可以有效地预先确定失败插入，而无需花费高昂的成本进行逐步探测。

1. 背景概述

高效的查询服务对于各种规模的云存储系统至关重要，特别是在处理海量数据的时候。根据国际数据公司（IDC）2014 年的报告，到 2020 年，创建和复制的信息量将达到 44 ZB，近 50％的云服务将依靠存储系统中的数据。此外，最近在 26 个国家的 1,780 位数据中心管理人员的调查中，超过 36％的受访者面临两个关键挑战，这些挑战有效地支持了大量涌现的应用程序，并处理迅速增长的数据管理复杂性。这反映了我们正在创造和获取比以往任何时候都更多的数据的现实，并且这一趋势继续加速。数据量的爆炸式增长给存储系统带来了巨大的挑战，特别是对高效数据查询服务的支持。在从小型手持设备到大型数据中心的各种计算设施中，人们正在收集和分析越来越多的数据。用户通常会对存储在本地磁盘或云存储系统上的数百 GB 的数据进行查询。商业公司通常每天处理 TB 甚至数 PB 的数据。

对于云存储系统来说，快速提供查询变得越来越具有挑战性，这往往消耗大量资源来支持与查询有关的操作。云管理系统通常需要支持低延迟和高吞吐量查询。为了解决这些挑战，查询服务受到了更多的关注，如 top-k 查询处理，多用户环境下文件系统搜索的安全模型，文件系统上的元数据查询，在移动计算中使用多核的 Web 搜索，抽象细化的图形查询处理，数据中心的节能在线搜索，高效查询压缩网络负载，尾部查询的延迟，以及针对异步并发缩放搜索数据结构。

高效的哈希方案对提高查询服务的性能非常重要。哈希表需要将键映射到值，并以实时方式支持恒定时间访问。Cuckoo 哈希是一个快速和简单的散列结构与恒定时间最坏情况查找(O(ln1/ε))和消耗(1+ε)n 内存消耗，其中ε是一个小常量。由于开放寻址的理想属性以及对低查找延迟的支持，cuckoo 哈希已被广泛用于真实世界的云应用。然而，cuckoo 哈希由于无限循环的发生而受到显着的性能损失。目前，无限循环的存在仅在可能大量的逐步剔除操作之后才被检测到。在无尽的循环中搜索插入位置变得毫无结果。为了提高性能并提高查找效率，需要解决两个主要挑战：大量的资源消耗；非确定性性能。现有的方案没有有效地解决这两个挑战。例如，MemC3 使用一个较大的踢出阈值作为其默认的踢出上限，这可能导致内存访问过多和性能下降。CHS 通过使用辅助数据结构作为存储来解决无限循环的问题。

2. 理论基础

2.1 Cuckoo Hashing

Cuckoo 哈希是一个静态字典的动态化，散列以多散列方式解决散列冲突。作为一种高效的散列方案，Cuckoo 哈希利用开放寻址来提高大数据集的查找效率。 在这种情况下，项与桶之间的关系可以用 Cuckoo 图来描述，其中每个边表示一个哈希项，而其两个顶点表示哈希目录中哈希项的位置。Cuckoo 哈希不需要动态内存分配，可以有效地利用动态内存分配来提供实时查询服务。Cuckoo 哈希能够支持快速查询，最坏情况下的恒定查找时间，因为它的地址对应到一个项目的多个位置。

2.2 伪森林理论

一个伪森林在图论中是无向的图和每个最大连通的组成部分组成的，称为子图，几乎是一个循环。换句话说，它是一个无向图，其中每个子图都没有比顶点更多的边。在伪造中，由连续边组成的两个周期不共享顶点，并且不能通过连续边的路径相互连接。

最大有向伪森林：最大有向伪森林是一个有向图，其中每个顶点有一个正好是 1 的出度。我们命名一个子图的顶点数等于它的边数最大的子图，最大的子图包含一个循环，最大定向伪树中的任何子图都是最大的子图。

本文认为 cuckoo 图是一个有向的伪森林。 伪林的每个顶点对应于一个哈希表桶，每个边对应于该项目的两个候选位置之间的项，插入的项目因此产生边缘。根据这个性质，一个最大的子图没有空间承认一个新的边，当有向边被遍历时，最终导致无限循环，这样一个无休止的循环将不会包含循环的最大子图。

3. SmartCuckoo 设计与评估

作为 Cuckoo 散列的一种高效的变体，SmartCuckoo 可以保持较高的查找效率，并通过避免不必要的排出操作来提高插入性能。它将项目插入分为三种情况，并利用有向伪森林来表示散列关系，用于追踪项目的位置，从而准确地预测无限循环的发生。常规 cuckoo 哈希选择一个项目的位置候选职位之一，而不考虑是否会走到一个无尽的循环。本文设计通过跟踪子图的状态来预测插入步骤结果，从而提高插入效率。因此，SmartCuckoo 能够巧妙地选择要插入的项目的插入位置。

SmartCuckoo 是基于 CHS 实现的，本文实验中使用的服务器配置为 Intel 2.8GHz 16 核 CPU，12GB DDR3 RAM，峰值带宽为 32GB/sm，500GB 的硬盘。CPU 的 L1 和 L2 缓存分别为 32KB 和 256KB。本文使用三个 trace（Random Integer，MacOS 和 DocWords）以及 YCSB 基准来运行 Linux 内核 2.6.18 中的 SmartCuckoo 原型来评估其性能。

本文将 SmartCuckoo 与 CHS（cuckoohashing with a stash）作为基准，与 lib cuckoo 和 BCHT 方案进行比较。具体而言，对于 BCHT，我们实现了其主要组件，其中包括每个桶中的四个插槽，对于 lib cuckoo，使用其开放源代码的 C ++ 实现，该实现经过优化，可用于编写繁重的工作负载。

4. 相关工作

Cuckoo 哈希结构：SmartCuckoo 是 Cuckoo 哈希的一个变种，它支持快速和低成本的查找操作。Cuckoo 哈希是一个开放寻址哈希方案，为哈希表中的每个项目提供多个候选位置。Cuckoo 使用 Cuckoo 哈希表来增强计数布隆过滤器，以支持插入和删除操作，提高性能和空间效率。Horton 表是一个增强桶分化布谷鸟

哈希表，以减少在每个查询中访问的 CPU 高速缓存行的数量。相比之下，本文研究了具有踢出行为的有向 Cuckoo 图的特征，所提出的 SmartCuckoo 利用有向伪森林，即图论中的一个概念来跟踪哈希表中的项目分布，以预先确定无限循环的发生。

基于内容的搜索：NEST 使用 Cuckoo 散列来解决传统的局部敏感哈希（LSH）中的负载不平衡问题，并支持近似查询。 HCTrie 是文件系统中使用科学元数据进行文件搜索的多维结构，支持大量的维度。MinCounter 为每个桶分配一个计数器来跟踪桶中的踢出时间，这通过选择较少使用的踢出路由来减少数据插入期间无尽循环的发生。SmartCuckoo 旨在避免由于项目插入中的无限循环而导致不必要的踢出操作。

可搜索的文件系统：Spyglass 是一种文件元数据搜索系统，基于命名空间组织的分层划分，具有高性能和可扩展性。Smartstore 基于下一代文件系统的文件语义信息重新组织文件元数据，它提供了高效且可扩展的复杂查询，并增强了系统的可扩展性和功能。Glance 是一个基于时间抽样的系统，在没有预先知识的情况下为聚合和顶 k 查询提供精确的答案。Ceph 使用动态子树分区来支持基于文件名的查询以及避免元数据访问热点。SmartCuckoo 为云存储系统提供快速查询服务。

5. 总结

快速高效的查询服务对于云存储系统非常重要。由于开放寻址的显著特点，Cuckoo 哈希支持快速查询，然而，它却受到插入过程中产生无数循环的问题的困扰。本文提出了一个名为 SmartCuckoo 的高效哈希方案，用于跟踪哈希表中的项目分布，通过将哈希关系表示为有向伪森林，SmartCucko 可以准确地确定 cuckoo 操作和无限循环的状态。本文从三个方面出发，即 Random Integer，Mac OS 和 Doc Words，以及 YCSB 基准来评估 SmartCuckoo 的性能。广泛的实验结果展示了 SmartCuckooover 在杜鹃散列存储，libcuckoo 和 BCHT 的状态工作方案下的诸多优点。SmartCuckoo 目前使用两个哈希函数来解决 cuckoo 哈希表无限循环的问题。众所周知，使用两个以上的哈希函数会显着增加操作的复杂性，因此使用较少，一般的方法是使用诸如双散列的技术将散列函数的数量减少到两个。未来我们计划使用两个以上的哈希函数在散列表上应用 SmartCuckoo 方法。此外，我们还将研究 SmartCuckoo 在每个桶中有多个槽的 cuckoo 散列表中的使用。

参考文献

[1] Libcuckoo library. https://github.com/efficient/libcuckoo.

[2]Symantec. 2010 State of the Data Center Global Data. http://www.symantec.com/ content/en/ us/about/media/pdfs/Symantec DataCenter10 Report Global.pdf (Jan. 2010).

[3] Traces and Snapshots Public Archive. http://tracer.filesystems.org (July 2014).

[4] Bag-of-words data set. http://archive.ics.uci.edu/ml/datasets /Bag+of+Words (Mar. 2008).

[5] ` ALVAREZ, C., BLESA, M., AND SERNA, M. Universal Stability of Undirected Graphs in the Adversarial Queueing Model. In Proceedings of the fourteenth annual ACM symposium on Parallel algorithms and architectures (2002), ACM, pp. 183 – 197.

[6] ARMBRUST, M., FOX, A., GRIFFITH, R., JOSEPH, A. D., KATZ, R., KONWINSKI, A., LEE, G., PATTERSON, D., RABKIN, A., STOICA, I., ET AL. A View of Cloud Computing. Communications of the ACM 53, 4 (2010), 50 – 58.

[7] BELL, G., HEY, T., AND SZALAY, A. Beyond the Data Deluge. Science 323, 5919 (2009), 1297 – 1298.

[8] BRESLOW, A. D., ZHANG, D. P., GREATHOUSE, J. L., JAYASENA, N., TULLSEN, D. M., XU, L., CAVAZOS, J., ALVAREZ, M. A., MORALES, J. A., AGUILERA, P., ET AL. Horton Tables: Fast Hash Tables for In-Memory Data-Intensive Computing. USENIX Association, pp. 281 – 294.

[9] BUTTCHER, S., AND CLARKE, C. L. A Security Model for Full Text File System Search in Multi-User Environments. In Proc. FAST (2005).

[10] BYKOV, S., GELLER, A., KLIOT, G., LARUS, J. R., PANDYA, R., AND THELIN, J. Orleans: Cloud Computing for Everyone. In Proc. SOCC (2011), ACM, p. 16.

[11] COOPER, B. F., SILBERSTEIN, A., TAM, E., RAMAKRISHNAN, R., AND SEARS, R. Bench marking Cloud Serving Systems with YCSB. In Proc. SoCC (2010), ACM, pp. 143 – 154.

[12] DAVID, T., GUERRAOUI, R., AND TRIGONAKIS, V. Asynchronized Concurrency: The Secret to Scaling Concurrent Search Data Structures. ACM SIGARCH Computer Architecture News 43, 1 (2015), 631 – 644.

[13] DEBNATH, B. K., SENGUPTA, S., AND LI, J. ChunkStash: Speeding up Inline Storage Deduplication using Flash Memory. In Proc. USENIX ATC (2010).

[14] DEVROYE, L., AND MORIN, P. Cuckoo hashing: Further analysis. Information Processing Letters 86, 4 (2003), 215 – 219.

[15] ERLINGSSON, U., MANASSE, M., AND MCSHERRY, F. ACool and Practical Alternative to Traditional Hash Tables. In Proc. WDAS (2006).

[16] FAN, B., ANDERSEN, D. G., AND KAMINSKY, M. MemC3: Compact and Concurrent MemCache with Dumber Caching and Smarter Hashing. In Proc. NSDI (2013).

[17] FAN, B., ANDERSEN, D. G., KAMINSKY, M., AND MITZENMACHER, M. D. Cuckoo Filter: Practically Better Than Bloom. In Proc. CoNext (2014), ACM, pp. 75 – 88.

[18] FAN, L., CAO, P., ALMEIDA, J., AND BRODER, A.Z. Summary Cache: A Scalable Wide-Area Web Cache Sharing Protocol. IEEE/ACM Transaction son Networking (TON) 8,3 (2000),281 – 293.

[19] FRIEZE, A., MELSTED, P., AND MITZENMACHER, M. An Analysis of Random-Walk Cuckoo Hashing. In Approximation, Randomization, and Combinatorial Optimization. Algorithms and Techniques. Springer, 2009, pp. 490 – 503.

[20] GABOW, H. N., AND WESTERMANN, H. H. Forests, frames, and games: Algorithms for matroid sums and applications. Algorithmica 7, 1 (1992), 465 – 497.
[21] GANTZ, J., AND REINSEL, D. Thedigitaluniversein2020: Big data, bigger digital shadows, and biggest growth in the far east. IDC iView: IDC Analyze the Future 2007 (2012), 1 – 16.
[22] HSU, C.-H., ZHANG, Y., LAURENZANO, M.A., MEISNER, D., WENISCH, T., MARS, J., TANG, L., AND DRESLINSKI, R. G. Adrenaline: Pinpointing and Reining in Tail Queries with Quick Voltage Boosting. In Proc. HPCA (2015), IEEE, pp. 271 – 282.
[23] HUA, Y., JIANG, H., ZHU, Y., FENG, D., AND TIAN, L. Smartstore: A New Metadata Organization Paradigm with Semantic-Awareness for Next-Generation File Systems. In Proc. SC (2009), ACM.
[24] HUA, Y., XIAO, B., AND LIU, X. Nest: Locality-aware Approximate Query Service for Cloud Computing. In Proc. INFOCOM (2013), IEEE, pp. 1303 – 1311.
[25] HUANG, H. H., ZHANG, N., WANG, W., DAS, G., AND SZALAY, A. S. Just-in-Time Analytics on Large File Systems. IEEE Transactions on Computers 61, 11 (2012), 1651 – 1664.
[26] HUSTON, L., SUKTHANKAR, R., WICKREMESINGHE, R., SATYANARAYANAN, M., GANGER, G. R., RIEDEL, E., AND AILAMAKI, A. Diamond: A Storage Architecture for Early Discard in Interactive Search. In Proc. FAST (2004), pp. 73 – 86.
[27] JANAPA REDDI, V., LEE, B. C., CHILIMBI, T., AND VAID, K. Web Search Using Mobile Cores: Quantifying and Mitigating the Price of Efficiency. In Proc. ISCA (2010), pp. 314 – 325.
[28] KIRSCH, A., AND MITZENMACHER, M. The Power of Qne Move: Hashing Schemes for Hardware. IEEE/ACM Transactions on Networking 18, 6 (2010), 1752 – 1765. [29] KIRSCH, A., MITZENMACHER, M., AND WIEDER, U. More Robust Hashing: Cuckoo Hashing with a Stash. SIAM Journal on Computing 39, 4 (2009), 1543 – 1561.
[30] KNUTH, D. E. The Art of Computer Programming: Sorting and Searching, vol. 3. Pearson Education, 1998.
[31] KRUSKAL, C. P., RUDOLPH, L., AND SNIR, M. Efficient parallel algorithms for graph problems. Algorithmica5,1(1990), 43 – 64.
[32] KUTZELNIGG, R. Bipartite Random Graphs and Cuckoo Hashing. In Fourth Colloquium on Mathematics and Computer Science Algorithms, Trees, Combinatorics and Probabilities (2006),Discrete Ma the matics and Theoretical Computer Science, pp. 403 – 406.
[33] LAM, H., LIU, Z., MITZENMACHER, M., SUN, X., AND WANG, Y. Information Dissemination via Random Walks in Dimensional Space. In Proc. SODA (2012), SIAM, pp. 1612 – 1622.
[34] LEUNG, A. W., SHAO, M., BISSON, T., PASUPATHY, S., AND MILLER, E. L. Spyglass: Fast, Scalable Metadata Search for Large-Scale Storage Systems. In Proc. FAST (2009), pp. 153 – 166.
[35] LI, Q., HUA, Y., HE, W., FENG, D., NIE, Z., AND SUN, Y. Necklace: An Efficient Cuckoo Hashing Scheme for Cloud Storage Services. In Proc. IWQoS (2014), IEEE, pp. 153 – 158.
[36] LI, X., ANDERSEN, D. G., KAMINSKY, M., AND FREEDMAN, M. J. Algorithmic Improvements for Fast Concurrent Cuckoo Hashing. In Proc. EuroSys (2014), ACM.
[37] LILLIBRIDGE, M., ESHGHI, K., BHAGWAT, D., DEOLALIKAR, V., TREZIS, G., AND CAMBLE, P. Sparse Indexing: Large Scale, Inline Deduplication Using Sampling and Locality.In Proc.FAST (2009), pp. 111 – 123.

[38] LIU, L., XU, L., WU, Y., YANG, G., AND GANGER, G. R. SmartScan: Efficient Metadata Crawl for Storage Management Metadata Querying in Large File Systems. Parallel Data Laboratory (2010), 1 – 17.
[39] MALTZAHN, C., MOLINA-ESTOLANO, E., KHURANA, A., NELSON, A. J., BRANDT, S. A., AND WEIL, S. Ceph as a scalablealternativetotheHadoopDistributedFileSystem. login: The USENIX Magazine 35, 4 (2010), 38 – 49.
[40] MATSUMOTO, M., AND NISHIMURA, T. Mersenne Twister: A 623-Dimensionally Equidistributed Uniform Pseudo-Random Number Generator. ACM Transactions on Modeling and Computer Simulation (TOMACS) 8, 1 (1998), 3 – 30.
[41] OHARA, Y. HCTrie: A Structure for Indexing Hundreds of Dimensions for Use in File Systems Search. In Proc. MSST (2013), IEEE, pp. 1 – 5.
[42] PAGH, R., AND RODLER, F. Cuckoo hashing. In Proc. ESA (2001), Springer, pp. 121 – 133.
[43] PAGH, R., AND RODLER, F. F. Cuckoo hashing. Journal of Algorithms 51, 2 (2004), 122 – 144.
[44] POLYCHRONIOU, O., RAGHAVAN, A., AND ROSS, K. A. Rethinking SIMD Vectorization for In-Memory Databases. In Proc. SIGMOD (2015), ACM, pp. 1493 – 1508.
[45] ROSS, K. A. Efficient Hash Probes on Modern Processors. In Pro c. ICDE (2007), IEEE, pp. 1297 – 1301.
[46] SUN, Y., HUA, Y., FENG, D., YANG, L., ZUO, P., AND CAO, S. Min Counter: An Efficient Cuckoo Hashing Scheme for Cloud Storage Systems. In Pro c. MSST (2015), IEEE.
[47] TARASOV, V., MUDRANKIT, A., BUIK, W., SHILANE, P., KUENNING, G., AND ZADOK, E. Generating Realistic Data sets for Deduplication Analysis. In Pro c. USENIX ATC (2012), USENIX Association, pp. 261 – 272.
[48] TAYLOR, T., COULL, S. E., MONROSE, F., AND MCHUGH, J. Toward Efficient Querying of Compressed Network Payloads. In Pro c. USENIX ATC (2012), USENIX Association, pp. 113 – 124.
[49] TURNER, V., GANTZ, J. F., REINSEL, D., AND MINTON, S. The digital universe of opportunities: Rich data and the increasing value of the internet of things. Framing ham (MA): IDC (2014).
[50] VAMANAN, B., SOHAIL, H. B., HASAN, J., AND VIJAYKUMAR, T. Time Trader: Exploiting Latency Tail to Save Data center Energy for Online Search. In Pro c. MICRO (2015), ACM, pp. 585 – 597.
[51] WANG, C., REN, K., YU, S., AND URS, K. M. R. Achieving Usable and Privacy-assured Similarity Search over Outsourced Cloud Data. In Proc. INFOCOM (2012), IEEE, pp. 451 – 459.
[52] WANG, K., XU, G., SU, Z., AND LIU, Y. D. GraphQ: Graph Query Processing with Abstraction Refinement-Scalable andProgrammableAnalyticsoverVeryLargeGraphsonaSingle PC. In Pro c. USENIX ATC (2015), USENIX Association, pp. 387 – 401.
[53] WEIL, S. A., BRANDT, S. A., MILLER, E. L., LONG, D. D., AND MALTZAHN, C. Ceph: A Scalable, High-Performance Distributed File System. In Pro c. OSDI (2006), USENIX Association, pp. 307 – 320.
[54] WU, S., LI, F., MEHROTRA, S., AND OOI, B. C. Query Optimization for Massively Parallel Data Processing. In Pro c. SOCC (2011), ACM.

[55] ZHANG, K., WANG, K., YUAN, Y., GUO, L., LEE, R., AND ZHANG, X. Mega-KV: A Case for GPUs to Maximize the Throughput of In-Memory Key-Value Stores. Proceedings of the VLDB Endowment 8, 11 (2015), 1226 – 1237.
[56] ZUO, P., AND HUA, Y. A Write-friendly Hashing Scheme for Non-volatile Memory Systems. In Pro c. MSST (2017).