验证数据结构的正确性,并通过实验来验证相应结构的性能.

5.2 Java中现有并发数据结构实现情况

由于并发数据结构存在设计和实现上的难度,因此真正已发布的、稳定的并发库其实并不太多.Java 中的 java.util.concurrent 是极少数得到广泛应用的软件包,现已在并发编程中成为很常用的工具类.在 Java 5 之前,Java 中如果需要处理并发,那么通常需要使用 wait()、notify()和 synchronized 手工实现并发处理的代码,加上需要考虑性能、死锁和资源管理等因素,开发的负担会很重.从 Java 5 开始,Java 推出了 java.util.concurrent 包,以简化并发的完成.这个包有几个小的、已标准化的可扩展框架和一些提供有用功能的类,没有这些类,这些功能会很难实现或实现起来很繁杂.java.util.concurrent 主要包括如下几个部分:Executors、Queues、Timing、Synchronizers、Concurrent Collections,其中,Concurrent Collections 包括了一些常用的并发数据结构.

截止 2015 年发布的 Java 8,java.util.concurrent 提供的数据结构有 ConcurrentLinkedQueue,ConcurrentLinkedDeque,ConcurrentHashMap,ConcurrentSkipListMap,ConcurrentSkipListSet,CopyOnWriteArrayList 和 CopyOnWriteArraySet.由此可见,目前 Java 8 版本中只提供一些最基本的并发数据结构,对于一些复杂的结构,则需要编程人员自己实现.

6 多核 CPU 数据结构正确性理论分析方法

设计出多核 CPU 数据结构后,就需要评估设计的正确性,通常有两种手段可选:实验的方法和理论分析的方法.实验的方法通常比较直观,理论分析的方法则逻辑性要求更强,表达起来也更困难.

通过对前人在多核 CPU 数据结构正确性分析方法进行对比研究,总结如下.

结构不变性

结构不变性主要用来说明并发操作并不会改变原始数据结构的属性.每种数据结构都有其自身特点的数据组织形式(如:链表和树的结构就不同),这些性质在数据结构被创建时就已成立,并且任何操作都不能改变这些性质.因此,当多线程并发地对多核 CPU 数据结构进行操作时,无论经过多少次插入、删除、查找,最终得到的还是同种性质的数据结构(如:初期是二叉树,经过 N 次并发操作后还应该是二叉树.不允许出现初期是二叉树,经过 N 次并发操作后变成 m 叉树).在具体分析说明时涉及到两个关键步骤:(1) 列举出这种结构的不变性质;(2) 对每一种操作都分析其不会改变原来数据结构的性质.举个例子:如果现在来分析并发跳表的结构不变性,则需要首先列举出跳表的结构的不变性质:(1) 哨兵节点不能改变.(2) 每一层的节点都按照关键字排序,并且不会出现重复的关键字.(3) 下层节点的节点数一定不少于上层节点.然后,再分别对查找、插入、删除、置标志位等操作进行分析,说明这些操作并不会改变其跳表的属性.

安全性(safety)

安全性意味着对并发数据结构的操作不会出错[5].对安全性的一个重要评价是可线性化,其基本思想是每一个并发的经历(history)都等价于一个顺序的经历.通常来说,用来说明并发数据结构的可线性化性质的方法就是指出该算法的可线性化点(linearizability point),然后指明在该线性化点的并发操作是如何映射为不同线程的等价顺序操作的.

活性(liveness)

当考虑并发数据结构的活性时,我们期待对这个数据结构的操作最终都能够完成[8].活性意味着并发数据结构算法最终在怎样的情况下完成.通常可以用演进条件来表示,如无死锁、无饥渴、无锁、无等待等.如果保证算法调用不会因为互相等待而无法获取资源,那么称其为无死锁.如果保证算法调用最终都能取得请求的资源,那么称其为无饥渴.如果保证算法某次调用能在有限的步骤内完成,那么这种方法是无锁的.如果算法每次调用总是在有限步骤内完成,那么这种方法是无等待的.

7 结束语及研究展望

随着商用多核(multicore)和众核(many-core)系统越来越普及,软件中数据结构对多核支持的需求也就越来

越迫切.因此,如何在多核时代提升数据结构的性能成为研究的热点.结合已有相关研究成果及其发展趋势,我们认为,以下有关研究将是研究者近年来所关注的热点问题.

基于硬件事务内存的并发数据结构研究

近年来,CPU 对硬件事务内存的支持不断提高.这为在此之上的并发算法研究提供了便利的条件.Intel 公司在 x86 架构中设计了 Transactional Synchronization Extensions(TSX)扩展指令集,加入 Hardware Transactional Memory(HTM)的支持.2013 年 6 月,Intel 推出了基于 Haswell 微架构的处理器.这成为主流设计中首次引入 Transactional Memory 的处理器.IBM 在 2013 年 8 月的 Hot Chips 会议上推出了 Power 8.它是基于 Power 架构的超标量体系结构对称多处理器家族,Power 8 加入了 Hardware Transactional Memory 的支持.

支持无等待的高效可实用的数据结构研究

按照多核数据结构演进条件的定义,演进条件越高,意味着活性越高,也在一定程度上表明其性能越好.在目前已经实现的数据结构中,只有采用线程监控数组技术实现了高效、实用的无等待的队列,虽然采用通用构建也可以得到无等待的数据结构,但是这些实现效率都不高[76,77].因此,对于很多并不具有无等待性质的数据结构来说,总是有部分比例的线程在多核执行时存在等待的时间.在这个问题上,任何高效、实用的数据结构(如:哈希、链表、树等)研究的突破都将产生较大的实际影响.

基于标志位和原子操作的无锁带调整复杂树型结构研究

标志位和原子操作是近几年来并发树型结构研究的重点,已在平衡二叉树上得到实现.然而,对于一些更加复杂的树形结构,如红黑树、空间树、前缀树、后缀树等,由于其结构组织本身具有特殊性或者存在特殊的调整策略,目前还少有文献涉及,因此这方面具有比较大的研究空间.

分布式系统的多核数据结构研究

现在大型系统都是分布式架构的,文献[78]对分布式系统中不同机器的内存建立了一个统一的映射,并在上面建立 B+树.这样,对于用户来说,看到的是一棵统一的 B+树,然而,B+树上不同节点是映射到不同机器的内存中的.文献[78]中 B+树的多核实现是基于锁的传统实现.那么,如何发挥多核优势,研究并设计出适应分布式环境的无锁数据结构也将是研究者关注的热点.

总之,多核的出现给我们带来机遇的同时也带来了巨大的挑战.基于共享内存的多核数据结构已成为研究者关注的热点问题.为了有效发挥硬件多核的优势,多核数据结构及其相关研究还存在许多挑战性问题,需要不断地去探讨和研究.

References:

[1] Gao L, Wang R, Qian DP. Deterministic replay for parallel programs in multi-core processors. Ruan Jian Xue Bao/Journal of Software, 2013,24(6):1390−1402 (in Chinese with English abstract). http://www.jos.org.cn/1000-9825/4392.htm [doi: 10.3724/SP.J.1001.2013.04392]

[2] Yuan QB, Zhao JB, Chen MY, Sun NH. Performance bottleneck analysis and solution of shared memory operating system on a multi-core platform. Journal of Computer Research and Development, 2011,48(12):2268−2276 (in Chinese with English abstract).

[3] Anderson JH, Kim YJ. An improved lower bound for the time complexity of mutual exclusion. Distributed Computing, 2002,15(4):221−253. [doi: 10.1007/s00446-002-0084-2]

[4] Arora NS, Blumofe RD, Plaxton CG. Thread scheduling for multiprogrammed multiprocessors. Theory of Computing Systems, 2001,34(2):115−144. [doi: 10.1007/s00224-001-0004-z]

[5] Herlihy M, Shavit N. The Art of Multiprocessor Programming (Revised Reprint). Elsevier, 2012.

[6] Fich FE, Hendler D, Shavit N. Linear lower bounds on real-world implementations of concurrent objects. In: Proc. of the 46th Annual IEEE Symposium on Foundations of Computer Science, 2005 FOCS. IEEE, 2005. 165−173. [doi: 10.1109/SFCS.2005.47]

[7] Moir M, Shavit N. Concurrent data structures. Lecture Notes in Computer Science, 2001,605(12):703−724.

[8] Shavit N. Data structures in the multicore age. Communications of the ACM, 2011,54(3):76−84. [doi: 10.1145/1897852.1897873]

[9] Amdahl GM. Validity of the single processor approach to achieving large scale computing capabilities. In: Proc. of the Spring Joint Computer Conf. ACM, 1967. 483−485. [doi: 10.1145/1465482.1465560]

[10] Michael MM, Scott ML. Nonblocking algorithms and preemption-safe locking on multiprogrammed shared memory multiprocessors. Journal of Parallel and Distributed Computing, 1998,51(1):1−26. [doi: 10.1006/jpdc.1998.1446]

[11] Treiber RK. Systems Programming: Coping with parallelism. Int'l Business Machines Incorporated, Thomas J. Watson Research Center, 1986.

[12] Shavit N, Touitou D. Elimination trees and the construction of pools and stacks. Theory of Computing Systems, 1997,30(6): 645−670. [doi: 10.1007/s002240000072]

[13] Hendler D, Shavit N, Yerushalmi L. A scalable lock-free stack algorithm. In: Proc. of the 16th Annual ACM Symp. on Parallelism in Algorithms and Architectures. ACM, 2004. 206−215. [doi: 10.1145/1007912.1007944]

[14] Lamport L. Specifying concurrent program modules. ACM Trans. on Programming Languages and Systems (TOPLAS), 1983,5(2): 190−222. [doi: 10.1145/69624.357207]

[15] Hendler D, Shavit N. Work dealing. In: Proc. of the 14th Annual ACM Symp. on Parallel Algorithms and Architectures, ACM, 2002. 164−172. [doi: 10.1145/564870.564900]

[16] Herlihy MP, Wing JM. Linearizability: A correctness condition for concurrent objects. ACM Trans. on Programming Languages and Systems (TOPLAS), 1990,12(3):463−492. [doi: 10.1145/78969.78972]

[17] Martin P, Moir M, Steele G. DCAS-Based concurrent deques supporting bulk allocation. Theory of Computing Systems, 2002, 35(3):59−73.

[18] Herlihy M, Luchangco V, Moir M. Obstruction-Free synchronization: Double-Ended queues as an example. In: Proc. of the 23rd Int'l Conf. on Distributed Computing Systems. IEEE, 2003. 522−529. [doi: 10.1109/ICDCS.2003.1203503]

[19] Bayer R, Schkolnick M. Concurrency of operations on B-trees. Acta informatica, 1977,9(1):1−21. [doi: 10.1007/BF00263762]

[20] Lea D. Concurrent Programming in Java: Design Principles and Patterns. Addison-Wesley Professional, 2000.

[21] Valois JD. Lock-Free linked lists using compare-and-swap. In: Proc. of the 14th Annual ACM Symp. on Principles of Distributed Computing. ACM, 1995. 214−222. [doi: 10.1145/224964.224988]

[22] Harris TL. A pragmatic implementation of non-blocking linked-lists. In: Distributed Computing. Springer-Verlag, 2001. 300−314. [doi: 10.1007/3-540-45414-4_21]

[23] Michael MM. High performance dynamic lock-free hash tables and list-based sets. In: Proc. of the 14th Annual ACM Symp. on Parallel Algorithms and Architectures. ACM, 2002. 73−82. [doi: 10.1145/564870.564881]

[24] Ellis CS. Concurrency in linear hashing. ACM Trans. on Database Systems (TODS), 1987,12(2):195−217. [doi: 10.1145/22952.22954]

[25] Kung H, Lehman PL. Concurrent manipulation of binary search trees. ACM Trans. on Database Systems (TODS), 1980,5(3): 354−382. [doi: 10.1145/320613.320619]

[26] Hsu M, Yang WP. Concurrent operations in extendible hashing. In: Proc. of the VLDB. Springer-Verlag, 1986. 25−28.

[27] Lea D. Concurrency JSR-166 Interest Site. 2014. http://gee.cs.oswego.edu/dl/concurrency-interest/index.html

[28] Shalev O, Shavit N. Split-Ordered lists: Lock-Free extensible hash tables. Journal of the ACM (JACM), 2006,53(3):379−405. [doi: 10.1145/1147954.1147958]

[29] Pagh R, Rodler FF. Cuckoo hashing. Journal of Algorithms, 2004,51(2):122−144.

[30] Fan B, Andersen DG, Kaminsky M. MemC3: Compact and concurrent memcache with dumber caching and smarter hashing. NSDI, 2003,13:385−398.

[31] Herlihy M, Shavit N, Tzafrir M. Hopscotch hashing. In: Distributed Computing. Berlin, Heidelberg: Springer-Verlag, 2008. 350−364. [doi: 10.1007/978-3-540-87779-0_24]

[32] Afek Y, Kaplan H, Korenfeld B, Morrison A, Tarjan RE. CBTree: A practical concurrent self-adjusting search tree. In: Distributed Computing. Springer-Verlag, 2012. 1−15. [doi: 10.1007/978-3-642-33651-5_1]

[33] Brown T, Ellen F, Ruppert E. A general technique for non-blocking trees. In: Proc. of the 19th ACM SIGPLAN Symp. on Principles and Practice of Parallel Programming. ACM, 2014. 329−342. [doi: 10.1145/2555243.2555267]

[34] Prokopec A, Bronson NG, Bagwell P, Odersky M. Concurrent tries with efficient non-blocking snapshots. In: Proc. of the ACM Sigplan Notices. ACM, 2012. 151−160. [doi: 10.1145/2145816.2145836]

[35] Prokopec A, Bagwell P, Odersky M. Lock-Free resizeable concurrent tries. In: Languages and Compilers for Parallel Computing. Springer-Verlag, 2013. 156−170. [doi: 10.1007/978-3-642-36036-7_11]

[36] Fraser K. Practical lock-freedom [Ph.D. Thesis]. Cambridge: University of Cambridge, 2004.

[37] Crain T, Gramoli V, Raynal M. A speculation-friendly binary search tree. ACM Sigplan Notices, 2012,47(8):161−170. [doi: 10.1145/2370036.2145837]

[38] Bronson NG, Casper J, Chafi H, Olukotun K. A practical concurrent binary search tree. In: Proc. of the ACM Sigplan Notices. ACM, 2010. 257−268. [doi: 10.1145/1693453.1693488]

[39] Ellen F, Fatourou P, Ruppert E, Van Breugel F. Non-Blocking binary search trees. In: Proc. of the 29th ACM SIGACT-SIGOPS Symp. on Principles of Distributed Computing. ACM, 2010. 131−140. [doi: 10.1145/1835698.1835736]

[40] Howley SV, Jones J. A non-blocking internal binary search tree. In: Proc. of the 24th ACM Symp. on Parallelism in Algorithms and Architectures. ACM, 2012. 161−171. [doi: 10.1145/2312005.2312036]

[41] Natarajan A, Mittal N. Fast concurrent lock-free binary search trees. In: Proc. of the 19th ACM SIGPLAN Symp. on Principles and Practice of Parallel Programming. ACM, 2014. 317−328. [doi: 10.1145/2555243.2555256]

[42] Drachsler D, Vechev M, Yahav E. Practical concurrent binary search trees via logical ordering. In: Proc. of the 19th ACM SIGPLAN Symp. on Principles and Practice of Parallel Programming, ACM, 2014. 343−356. [doi: 10.1145/2555243.2555269]

[43] Chatterjee B, Nguyen N, Tsigas P. Efficient lock-free binary search trees. arXiv preprint arXiv:14043272, 2014.

[44] Brown T, Helga J. Non-Blocking *k*-ary search trees. In: Principles of Distributed Systems. Springer-Verlag, 2011. 207−221. [doi: 10.1007/978-3-642-25873-2_15]

[45] Brown T, Avni H. Range queries in non-blocking *k*-ary search trees. In: Principles of Distributed Systems. Springer-Verlag, 2012. 31−45. [doi: 10.1007/978-3-642-35476-2_3]

[46] Crain T, Gramoli V, Raynal M. A contention-friendly binary search tree. In: Euro-Par 2013 Parallel Processing. Springer-Verlag, 2013. 229−240. [doi: 10.1007/978-3-642-40047-6_25]

[47] Leo JG. A dichromatic framework for balanced trees. In: Proc. of the Annual IEEE Symp. on Foundations of Computer Science. IEEE Computer Society, 1978. 8−21. [doi: 10.1109/SFCS.1978.3]

[48] Herlihy M. Wait-Free synchronization. ACM Trans. on Programming Languages and Systems (TOPLAS), 1991,13(1):124−149. [doi: 10.1145/114005.102808]

[49] Herlihy M. A methodology for implementing highly concurrent data objects. ACM Trans. on Programming Languages and Systems (TOPLAS), 1993,15(5):745−770. [doi: 10.1145/161468.161469]

[50] Herlihy M. A methodology for implementing highly concurrent data structures. In: Proc. of the ACM SIGPLAN Notices. ACM, 1990. 197−206. [doi: 10.1145/99163.99185]

[51] Nurmi O, Soisalon-Soininen E. Chromatic binary search trees. Acta Informatica, 1996,33(6):547−557. [doi: 10.1007/s002360050057]

[52] Boyar J, Fagerberg R, Larsen KS. Amortization results for chromatic search trees, with an application to priority queues. Journal of Computer and System Sciences, 1997,55(3):504−521. [doi: 10.1006/jcss.1997.1511]

[53] Besa J, Eterovic Y. A concurrent red-black tree. Journal of Parallel and Distributed Computing, 2013,73(4):434−449. [doi: 10.1016/j.jpdc.2012.12.010]

[54] McKenney PE, Slingwine JD. Read-Copy update: Using execution history to solve concurrency problems. In: Parallel and Distributed Computing and Systems. 1998. 509−518.

[55] Matveev A, Shavit N, Felber P, Marlier P. Read-Log-Update: A lightweight synchronization mechanism for concurrent programming. In: Proc. of the 25th Symp. on Operating Systems Principles. New York: ACM, 2015. 168−183. [doi: 10.1145/2815400.2815406]

[56] Kung HT, Robinson JT. On optimistic methods for concurrency control. ACM Trans. on Database Systems (TODS), 1981,6(2):213−226. [doi: 10.1145/319566.319567]

[57] Herlihy M, Luchangco V, Moir M, Scherer Iii WN. Software transactional memory for dynamic-sized data structures. In: Proc. of the 22nd Annual Symp. on Principles of Distributed Computing. ACM, 2003. 92−101. [doi: 10.1145/872035.872048]

[58] Ballard L. Conflict avoidance: Data structures in transactional memory [Ph.D. Thesis]. Brown University Undergraduate, 2006.

[59] Herlihy M, Lev Y, Luchangco V, Shavit N. A provably correct scalable concurrent skip list. In: Proc. of the Conf. on Principles of Distributed Systems (OPODIS), 2006. http://citeseerx.ist.psu.edu/viewdoc/download?doi=10.1.1.170.719&rep=rep1&type=pdf

[60] Herlihy M, Moss JEB. Transactional memory: Architectural support for lock-free data structures. Journal of the ACM, 1993,21(2):289−300. http://hpl.americas.hp.net/techreports/Compaq-DEC/CRL-92-7.pdf

[61] Wikipedia. Transactional synchronization extensions. 2014. http://en.wikipedia.org/w/index.php?title=Transactional_Synchronization_Extensions&oldid=606067145

[62] Shavit N, Touitou D. Software transactional memory. Distributed Computing, 1997,10(2):99−116. [doi: 10.1007/s004460050028]

[63] Adl-Tabatabai AR, Kozyrakis C, Saha B. Unlocking concurrency. Queue, 2006,4(10):24−33. [doi: 10.1145/1189276.1189288]

[64] Adl-Tabatabai AR, Lewis BT, Menon V, Murphy BR, Saha B, Shpeisman T. Compiler and runtime support for efficient software transactional memory. In: Proc. of the ACM SIGPLAN Notices. ACM, 2006. 26−37. [doi: 10.1145/1133981.1133985]

[65] Ansari M, Kotselidis C, Watson I, Kirkham C, Luján M, Jarvis K. Lee-TM: A non-trivial benchmark suite for transactional memory. In: Algorithms and Architectures for Parallel Processing. Springer-Verlag, 2008. 196−207. [doi: 10.1007/978-3-540-69501-1_21]

[66] Minh CC, Chung J, Kozyrakis C, Olukotun K. STAMP: Stanford transactional applications for multi-processing. In: Proc. of the 2008 IEEE Int'l Symp. on Workload Characterization. IEEE, 2008. 35−46. [doi: 10.1109/IISWC.2008.4636089]

[67] Dalessandro L, Marathe VJ, Spear MF, Scott ML. Capabilities and limitations of library-based software transactional memory in C++. In: Proc. of the 2nd ACM SIGPLAN Workshop on Transactional Computing. 2007. 49. https://anon.cs.rochester.edu/u/scott/papers/2007_TRANSACT_RSTM2.pdf

[68] Dice D, Shalev O, Shavit N. Transactional locking II. In: Distributed Computing. Springer-Verlag, 2006. 194–208. [doi: 10.1007/11864219_14]

[69] Dice D, Shavit N. What really makes transactions faster. In: Proc. of the TRANSACT Workshop. 2006. http://www.cs.tau.ac.il/~shanir/nir-pubs-web/Papers/TRANSACT06.pdf

[70] Eddon G, Herlihy M. Language support and compiler optimizations for STM and transactional boosting. In: Distributed Computing and Internet Technology. Springer-Verlag, 2007. 209–224. [doi: 10.1007/978-3-540-77115-9_22]

[71] Wikipedia. Software transactional memory. 2014. http://en.wikipedia.org/w/index.php?title=Software_transactional_memory&oldid=613666501

[72] Cascaval C, Blundell C, Michael M, Cain HW, Wu P, Chiras S, Chatterjee S. Software transactional memory: Why is it only a research toy. Queue, 2008,6(5):40–46. [doi: 10.1145/1454456.1454466]

[73] Kogan A, Petrank E. Wait-Free queues with multiple enqueuers and dequeuers. ACM SIGPLAN Notices, 2011,46(8):223–234. [doi: 10.1145/2038037.1941585]

[74] Kogan A, Petrank E. A methodology for creating fast wait-free data structures. In: Proc. of the ACM SIGPLAN Notices. ACM, 2012. 141–150. [doi: 10.1145/2145816.2145835]

[75] Timnat S, Braginsky A, Kogan A, Petrank E. Wait-Free linked-lists. In: Principles of Distributed Systems. Springer-Verlag, 2012. 330–344. [doi: 10.1007/978-3-642-35476-2_23]

[76] Fatourou P, Kallimanis ND. A highly-efficient wait-free universal construction. In: Proc. of the 23rd Annual ACM Symp. on Parallelism in Algorithms and Architectures. ACM, 2011. 325–334. [doi: 10.1145/1989493.1989549]

[77] Chuong P, Ellen F, Ramachandran V. A universal construction for wait-free transaction friendly data structures. In: Proc. of the 22nd ACM Symp. on Parallelism in Algorithms and Architectures. ACM, 2010. 335–344. [doi: 10.1145/1810479.1810538]

[78] Aguilera MK, Golab W, Shah MA. A practical scalable distributed b-tree. Proc. of the VLDB Endowment, 2008,1(1):598–609. [doi: 10.14778/1453856.1453922]

附中文参考文献:

[1] 高岚,王锐,钱德沛.多核处理器并行程序的确定性重放研究.软件学报,2013,24(6):1390−1402. http://www.jos.org.cn/1000-9825/4392.htm [doi: 10.3724/SP.J.1001.2013.04392]

[2] 袁清波,赵健博,陈明宇,孙凝晖.多核平台共享内存操作系统性能瓶颈分析及解决.计算机研究与发展,2011,48(12):2268−2276.

周维(1974－),男,湖南常德人,博士,副教授,主要研究领域为分布式计算,云计算.

姚绍文(1966－),男,博士,教授,博士生导师,主要研究领域为分布式计算,工作流.

周可人(1992－),男,学士,主要研究领域为并发编程,并行算法优化.

钱德沛(1952－),男,教授,博士生导师,CCF会士,主要研究领域为计算机系统结构,分布式计算,高性能计算,并行计算.

栾钟治(1971－),男,博士,副教授,CCF 高级会员,主要研究领域为分布式计算,高性能计算,并行计算,计算机系统结构.