面向大数据的索引结构研究进展

严赵峰，张为华
复旦大学并行处理研究所，上海 201203

摘要
为了保证各类大数据系统的性能，以并发索引结构为核心的高效检索技术变得越来越重要。然而，数据存储体量的增加和用户对性能要求的提高使得并发索引结构面临诸多挑战，设计高效且易用的并发控制策略和提升索引结构操作性能成为系统领域关心的重要问题。结合现有的研究成果，综合分析了大数据时代并发索引结构的研究进展。首先讨论了关于设计更优化且易用的并发控制策略的研究现状，接着针对各类新型高性能硬件，探讨了基于新硬件结构特点的优化加速研究成果，并对可能的发展方向进行了展望。

关键词
大数据；并发索引结构；新型硬件

中图分类号：TP399 文献标识码：A doi: 10.11959/j.issn.2096-0271.2019028

A survey of index structure in big data era

YAN Zhaofeng, ZHANG Weihua
Fudan University Parallel Processing Institution, Shanghai 201203, China

Abstract
With the advent of the big data era, the volume of data storage has exploded. In order to ensure the performance of system, the concurrent index structure becomes more and more important. However, the increase of storage volume and the increase of user performance requirements have made the index structure face many challenges. Designing an efficient and easy-to-use index structure has become an important issue in the system field. Based on current research status, firstly, the research status of designing more optimized and easy-to-use concurrency control strategies were discussed, and then the researches based on the new hardware structure features and possible directions were discussed.

Key words
big data, concurrent index structure, new hardware

1 引言

随着互联网技术和相关应用的蓬勃发展，全球已进入大数据时代。国际数据公司（International Data Corporation，IDC）在2018年的调研报告中指出，全球数据正在以指数级的速率增长，如图1所示，2019年全球产生的数据约为40 ZB，到2025年全球将产生175 ZB的数据。在大数据时代，利用大数据进行处理与预测的应用变得越来越普遍，这些应用在人们生活中占有举足轻重的地位，比如人们可以基于大数据对城市交通拥堵情况进行有效的判断和防治。然而，大数据应用也正面临着数据爆炸带来的挑战，Netflix公司2018年的报告显示，该公司每天处理的数据量已达到12 PB，这对于其流处理系统而言是一个极大的挑战。

由于数据的爆发式增长会显著影响大数据应用的检索和处理效率，因此该问题得到了学术界和工业界的广泛关注。在大数据领域，基于高效的索引结构保证数据处理和检索的效率是一种通用的解决方案，如MySQL聚簇索引和辅助索引均使用B+tree索引结构保证数据库条目访问的效率，并基于该结构提供顺序遍历等高效的操作；Microsoft公司的SQLServer数据库将Hash表作为数据表的缩影，从而加快处理速度。除了数据量的增长外，大数据系统面临着大量用户同时访问系统的情况。例如，在2017年的阿里巴巴“天猫双十一”活动中，阿里云系统每秒需要处理数百万次的交易操作。为了更好地支持并发访问，索引结构往往需要一定的机制保护其数据结构的正确性，即支持高并发访问，该种结构被称为并发索引结构。并发索引结构在各类大数据系统中被广泛使用，已成为各种大数据系统的核心模块，因此系统中的并发索引结构性能的优劣将直接影响这些系统的效率。

随着数据量的激增和并发访问量的增多，并发索引结构正面临着诸多挑战。这些挑战可以被分为两方面：其一，用于保护索引结构的并发控制机制是决定索引数据结构正确性和处理效率的关键因素，经典的保护机制已经无法满足编程人员对易用性和用户对性能的要求，因此设计更为高效的并发控制机制变得越来越重要；其二，传统基于CPU的大数据系统由于计算能力的限制已经无法满足用户对高性能处理的需求，层出不穷的新硬件技术为并发索引结构的性能优化提供了潜在的可能，

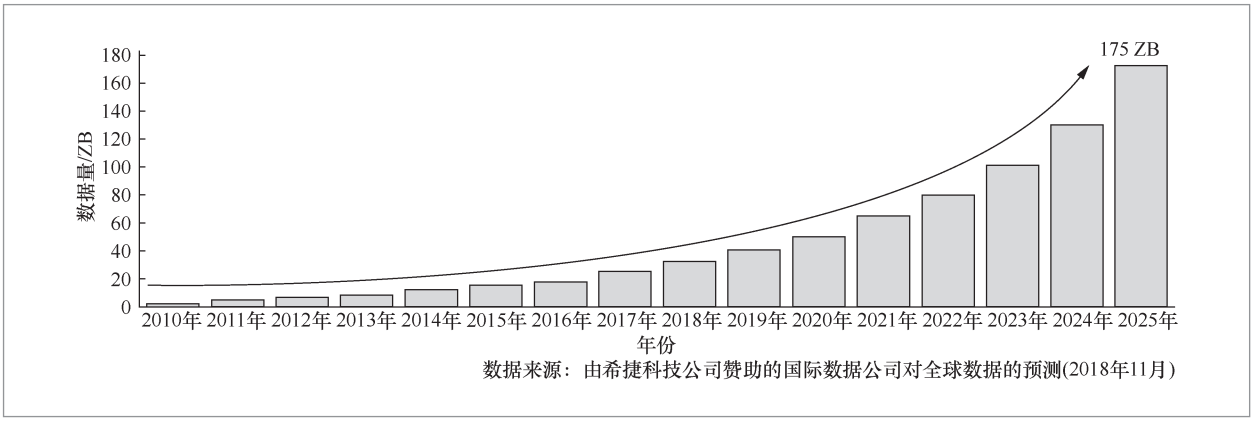


图1 每年全球数据量的增长

如何利用新型硬件的特性达到更好的执行效率以满足用户的需求，成为另一个重要的挑战。

本文结合并发索引数据结构面临的各项挑战，针对大数据时代对并发性能的要求，从并发控制技术和基于新型硬件的性能优化两个方面对现有的工作进行了总结与讨论。

2 背景

数据存储体量指数式的增长给大数据系统带来了挑战，如何高效地检索和处理数据以满足用户的需求，成为大数据系统领域的一个关键问题。并发索引数据结构作为一种能够有效组织底层数据以提供高效操作的方案，为解决该问题提供了可能。

2.1 正确性与易用性

数据的爆发同时也带来了用户数量的飞速增长。据Global Digital公司的数据统计，2018年已经有超过40亿的用户使用互联网，越来越多的人依赖于从大数据应用中得到个性化信息，方便自己的生活。随着在线用户的持续增加，大数据系统不仅要面对底层数据体量持续增大的挑战，同时也要面对用户的日益增长对并发访问效率造成的挑战。因此，日益增加的数据存储量和并发访问量给并发索引结构带来了严峻的挑战。

在高并发访问操作中保证结果的正确性和不同并发操作的可串行性是高并发大数据系统中至关重要的问题。传统的基于锁的并发保护策略在同步时可能存在死锁、活锁、优先级反转和并行度不足等潜在问题。另外，为实现一个高效且正确的锁控制策略，需要大量富有经验的程序员参与设计并实现，这对于日益增多的大数据应用而言存在较大的开发难度。因此，设计一种用户（编程者和用户）易用且高效的并发控制策略，以保证并发索引结构的正确性和高效性，成为一种挑战。

2.2 各类新型硬件

基于经典硬件的并发索引结构系统提供的处理效率和并发能力有限，这成为限制大数据系统提供高效服务的瓶颈。随着新型硬件技术的快速发展，各种新型硬件层出不穷，如硬件事务性内存（hardware transactional memory，HTM）、远程直接内存访问（remote direct memory access，RDMA）、图形处理器（graphics processing unit，GPU）和现场可编程门阵列（field-programmable gate array，FPGA）等。这些新型硬件为解决该问题提供了潜在可行的处理方案。

这些新型硬件具有各不相同的用途及特点。充分地理解和利用这些特点，从而对并发索引结构进行优化，才能取得理想的性能。如在GPU中，虽然存在大量的并发线程，但是由于其体系结构与CPU差异较大，因此在数据结构访问时需要避免乱序的访存或执行分歧等问题，这样才能获得理想的加速效果。再如HTM，它能够为程序提供堪比细粒度锁的高效硬件同步机制，减少不必要的锁开销，并简化用户的编程复杂度。然而，英特尔公司设计的RTM（一种HTM的具体实现）中存在许多限制条件，如事务内存（transactional memory，TM）中处理的工作集大小是有限的，TM中无法支持I/O操作等。这些问题对并发索引结构的设计提出了更高的要求。

基于传统的并发索引结构经过多年的发展，已经与传统硬件耦合严重，仅仅将

传统的并发索引结构移植到新型硬件平台上是难以获得理想的处理效果的。因此，如何基于新型硬件的特点设计和实现并发索引结构，以提供用户满意的检索处理效率，是当下另一个具有挑战的问题。

2.3 各类应用场景

针对并发索引结构对数据更新实时性的要求，可将其应用场景分为两大类：支持并发操作的场景和支持批更新处理的场景。并发索引数据结构应用场景的不同导致其设计的侧重点存在一定的差异。

在支持并发操作的应用场景中，大量用户并发地访问和更新底层数据，用户对数据的操作需要及时地更新在底层的数据结构上。为了支持该应用场景的需求，并发索引数据结构需要使用并发控制策略（如各种同步机制）来协调同时执行的数据读取操作和数据更新操作。这些同步机制给并发索引结构的执行带来了显著的开销，也影响着整体系统的性能和复杂度。因此，在支持并发操作的应用场景中，编程者将设计的重点聚焦于两方面：其一，如何设计一种完善的并发控制策略以保证并发访问的正确性，如常见的锁机制、无锁机制和事务内存机制等；其二，如何设计一些策略保证并发索引结构在复杂的高并发环境下的性能。

对于一些对数据更新实时性不是十分敏感的应用场景而言，如搜索引擎中的后端网页库，批处理更新是一种高效的处理方式。在该场景中用户对索引结构的更新操作可以被延迟至一定的阶段统一处理，并且可以将读取与更新操作相互分离，以简化两者之间的同步带来的额外开销，该应用场景主要应用于在线分析处理（on-line analytical processing，OLAP）、决策和数据挖掘等系统中[1-4]。在此类应用场景中，用户对于查询的性能更为敏感，因此该场景下设计的侧重点主要是如何提升并发索引结构的查询效率。

3 并发控制技术研究现状

并发索引结构可以对系统底层数据进行有效的组织，以提供高效的处理操作，它能够有效地应对数据量和用户数量的增长带来的挑战，并且在不同的索引结构应用场景和不同的硬件平台上均被广泛使用。然而，为了更好地服务用户（编程人员和系统使用者），提供简单且高效的并发控制技术存在一定的必要性。

为了应对大数据时代的挑战，设计一种高效且简单的并发控制策略来保证并发执行的正确性，成为一个普遍热门的问题。针对不同并发的应用场景，可将并发控制策略研究与实践的成果分为三大类：基于锁的并发策略、基于无锁的并发策略、基于事务内存的并发策略。另外，还有一些专门针对批更新处理场景的并发控制策略。这些不同的并发策略具有不同的特点，不同的应用场景和硬件平台需要结合其自身特点选择合适的策略。本节将依次介绍这4种更新策略的研究现状。

3.1 基于锁的并发策略

使用基于锁的并发控制策略保证系统执行的正确性是较常见且较直观的方式。早在1977年Bayer等人[5]就提出了针对B-tree索引结构的锁策略优化方案，以提高索引的并发度，其后又有大量的研究成果相继发表。经典的并发锁策略在查询时首先要锁定父节点，然后查找并获取子节点的锁，接着释放父节点，重复这个过程依次向下，直到到达叶节点。这种方式会导

致并发索引结构中过多的数据被锁定，算法整体的并行度较差。为了提高并行度，当前Blink-tree[6]和Masstree[7]是较经典且在当前各类大数据系统中应用较广泛的两种设计方案。**图2**为Blink-tree的示意图，其中，上半部分为树的组织结构示意，下半部分为具体节点的组织方式示意。Blink-tree对经典的B+tree索引结构进行了修改，在每个树节点中新增了一个最大键值（high-key）和指向同层右兄弟节点的指针（具体信息存在于节点的Link区域中）。该设计通过节点的读写锁和兄弟指针，降低并发操作过程中各个线程之间的相互影响的程度。当在Blink-tree上执行查询时，遍历树的过程首先比较当前节点的最大键值是否小于目标键值，如果小于目标键值，则目标键值是本节点的右兄弟节点，因此可通过右兄弟节点指针进行跳转，整个查询过程不会持有相应树节点的锁，这有效地提升了读并行度。当执行插入操作导致内部节点分裂时，在经典B+tree结构中往往整个树节点和新增的树节点将被锁住，无法执行查询等任务，但是基于Blink-tree的方式可以减小锁粒度控制的区域，这种方式显著地提高了整体的并发度和性能。

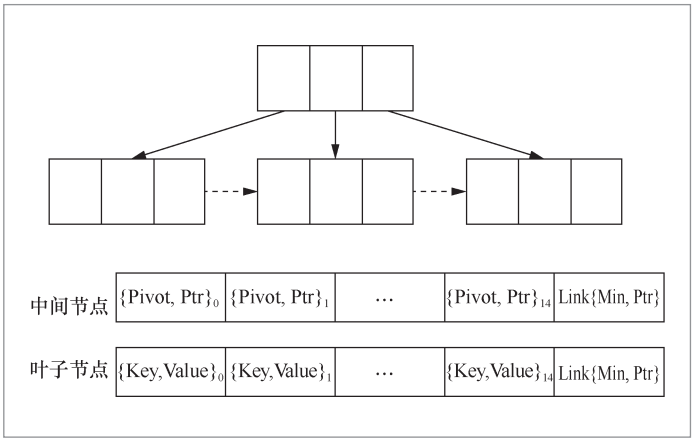


图 2 Blink-tree 结构 [6]

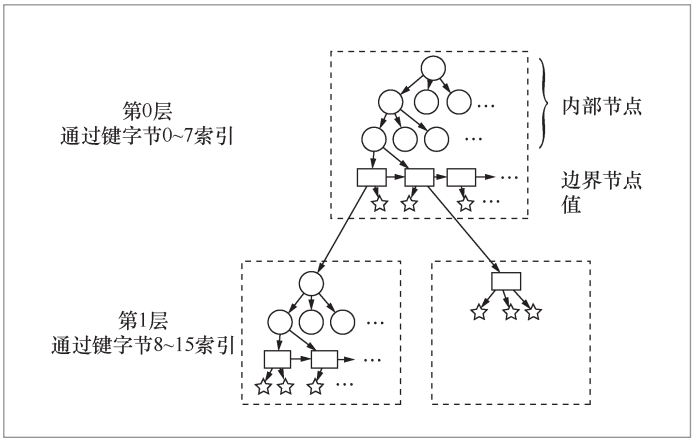


图 3 Masstree 结构 [7]

Masstree[7]于2012年被提出，其设计参考了以前的多种索引树结构，包括Blink-tree和OLFIT-tree[8]。它采用了B+tree索引结构和Trie树结构相互融合的方式，如**图3**所示，将树中的不同层次及同层次的不同节点按一定的规则划分成若干个不同区域，各区域之间按传统的B+tree的方式组织，而每个区域内部的节点按Trie树的方式组织。它使用版本验证（version verification）的方式代替读锁，并且配合细粒度锁实现更新与查询的并发执行。其中，并发控制策略的设计使用了读复制更新（read-copy-update，RCU）的概念。在更新数据时，系统需要先复制一份副本，在副本上完成修改，再一次性地替换旧数据。而读操作则可以不因任何更新操作而阻塞。因此该设计有效地提高了查询的执行效率。

基于锁机制的并发控制虽然容易设计，且经典结构较好地解决了访问并发度的问题，但是由于其需要频繁地访问锁，因此锁的开销随着并发度的提升也逐渐不可被忽略，成为锁机制的并发控制的性能问题之一。另外，使用锁的开发过程会引入很多潜在的漏洞，如死锁或活锁，基于锁的并发控制策略对编程人员的能力要求也较高。

3.2 基于无锁的并发策略

为了应对基于锁的并发策略中存在的问题（如死锁、锁同步开销过大等），基于无锁的并发策略逐渐得到了研究人员的重视。

Levandoski等人[9]设计的Bw-tree是当前主要的无锁树形结构。该数据结构通过对所有状态更改使用原子操作（CAS指令，即Compare and Swap指令）来避免不必要的开销，并提高并行效率，该设计已经被应用于微软公司的DocumentDB和AzureCosmosDB中。Bw-tree结构主要围绕映射表（mapping table）构建，通过该表有效地虚拟化索引页表的位置和页面大小，这为Bw-tree中基于主存的无锁策略提供了保证。在更新过程中，Bw-tree没有使用重写或者锁机制，使用CAS指令实现对树结构的更新操作。Sewall等人[10]设计了被称为PALM的B+tree，该树也可以通过无锁机制保证并发查询和更新操作的顺利执行。

基于无锁结构的并发控制策略有效地规避了锁同步的开销，从而能够有效地提升性能，但是由于其设计与实现难度较大，在实际应用中的广度仍不及基于锁的并发策略。

3.3 基于事务内存的并发策略

为了提升索引结构的并发度，基于事务内存的并发访问策略应运而生。事务内存将传统数据库中常用的事务与计算机的内存结合，保证不同的并发处理之间具有原子性、一致性、隔离性的特点。

基于事务的并发索引设计根据其实现方式的不同可分为两大类：软件事务内存（software transactional memory，STM）设计和HTM设计。这两类方案均可以在索引结构上实现简单易用的并发控制策略[11-13]。

Wang等人[14-15]使用HTM实现了多种并发数据结构，该系统（DBX系统）利用英特尔公司提供的硬件限制事物内存（restricted transactional memory，RTX）实现基于B+tree和Hash表的键值存储（key-value store）系统。这些并发数据结构在RTM的保护下保证了多线程访问的正确性和可串行性。另外，为了使新型硬件支持事务内存的并发处理方式，Chen等人[16]在GPU上实现了高效的HTM，以支持此并发策略的广泛应用。

Herliy等人[17]较早使用STM策略实现了高效的无锁并发数据结构。为了使不具备事务内存的硬件使用该特性，有许多研究使用软件的策略实现了事务内存策略。其中，Xu等人[18]在GPU上实现了STM策略，使得事务性内存策略在新型硬件上得以应用。

基于事务内存设计的并发控制策略能够较好地解决并发度的问题，并且可以降低编写并发程序的难度，缩短开发周期。基于此策略能够方便地对并发操作进行抽象和控制。

3.4 基于批处理更新的并发策略

为了实现批处理场景下索引数据结构的高效性，Pollari-Malmi等人[19]和Shneiderman等人[20]认为更新操作可以延迟，从而提高并发查询的执行效率。该设计模式将查询与并发更新按阶段依次执行，减少了查询与并发更新之间不必要的同步开销。Arge等人[21]和Graefe等人[22]提出了缓存写操作的更新策略，以避免不必要的I/O操作。批更新的并发处理策略在PALM[10]和HB+tree[2]中都得到了使用。其中，PALM是基于无锁索引结构和批处理策

略实现的索引结构，其使用了大量的预取操作和内核SIMD指令级并行来提供并发查询的吞吐量。HB+tree同样采用了批处理策略，在CPU端进行索引结构的批处理更新操作，在GPU端进行并发查询操作。为了更高效地使用CPU和GPU资源，HB+tree基于缓存行长度对B+tree结构进行了设计，有效地提高了整体的访存性能。最后，为了提高GPU与CPU的协作效率，HB+tree还提出了多种异构计算合作模式，比如CPU-GPU流水线的模式、双流模式（double buffering）等。批处理更新在Harmonia[23]和GPUB-tree[24]中也得到了使用。

基于批处理的并发控制策略针对更新非实时性特点，简化了并发处理策略的复杂度，提高了特定场景的检索效率。

4 针对新硬件特性的优化

随着半导体技术的飞速发展，各种新型硬件不断涌现，其中较为热门的是多核CPU、异构芯片、与分布式网络相关的硬件以及新型存储部件。它们针对传统硬件的某一方面进行了增强，以应对不同场景下不同计算的需求。高效使用新硬件特性优化并发索引结构，成为用户提高大数据系统质量的潜在途径之一。

4.1 基于多核CPU的优化

在并发索引结构优化领域，多核CPU的优化一直是学术界和工业界关心的问题。从基于多核CPU的体系结构可知，多核CPU间存在多个处理器核共享的缓存（L3缓存）和每个处理器核独享的缓存（L1、L2缓存）。因此，当前各类研究成果均期望充分利用缓存和多核CPU的各个计算核心，提高并发索引结构的处理效率。

Rao等人[25-26]提出的CSS-tree、CSB+tree均对多核CPU下的缓存行结构进行了针对性的设计。Chen等人[27-28]、Hankins等人[29]也在设计中得到了一致的结论，B-tree索引结构中树节点大小的设计会显著地影响并发索引结构在CPU上的性能。FAST[30]是这些研究成果的集大成者，它采用灵活的配置方式，基于CPU的缓存行大小、内存页大小和向量指令的宽度进行配置，并且尽最大能力减少存储器时延，充分利用线程级并行和数据级并行，提高索引结构在CPU端和GPU端的执行效率。

除了基于多核CPU存储器结构的设计之外，随着HTM新型硬件的出现和广泛使用，利用HTM优化索引结构也逐渐成为研究的热点。由于并发索引结构需要动态地更新数据结构，因此利用HTM能够实现易用的设计和高吞吐、低时延的事务处理。Wang等人[12-13]提出的Eunomia针对CPU端索引结构在高竞争场景中的并发索引结构面临的可拓展性问题，进行了详细的讨论与优化设计。作者分析了索引结构在高竞争场景中性能严重下降的问题（如数据冲突的可能性增大、重试的开销较大、假冲突的比例较多等），通过将原本粗粒度的HTM事务区间划分为两部分来减少冲突的概率和重试的开销，并提出了并发索引树在高竞争场景下的设计原则：分割事务区间减少事务重试开销，分段存储数据减少假冲突，检测消除真冲突和自适应的冲突控制策略。另外，作者还设计了基于版本号的一致性验证策略，以期解决HTM带来的整体一致性问题。基于以上原则重新设计并实现了并发B+tree，取得了在高竞争场景下高性能的处理效率和较好的可拓展性。

4.2 基于异构芯片的优化

GPU和FPGA是当下主流的异构计算硬

件，GPU因其更好的可编程性，被使用得更为广泛。GPU具有丰富的硬件资源，并广泛应用于各大数据中心。如何利用GPU实现高效的索引结构得到了越来越多的关注。目前已有许多关于使用GPU加速索引结构的工作相继被发表[30-33]，它们均期望在GPU上提升经典的B+tree结构的性能。

当前学术界较新的研究成果为Yan等人[23]提出的Harmonia和Awad等人[24]提出的与B-tree更新相关的优化方案。研究成果性能分布如**图4**所示，将HB+tree（CPU）作为性能比较基准，更新吞吐率和查询吞吐率的值越高表示性能越好。从**图4**可以看出，3种基于GPU的B+tree的设计（HB+tree（GPU）、GPU B-tree和Harmonia）在查询性能或更新性能方面优于HB+tree（CPU）的性能。GPU B-tree针对更新操作进行了优化，Harmonia针对查询操作进行了优化，这两种方案在各自的优化方向上取得了较为理想的优化效果。然而，这两种方案都只能在一个指标上取得较好的性能，不能兼顾两个指标。

Yan等人[23]提出的Harmonia将B+tree索引结构进行了重新设计，以期能够在GPU的硬件体系结构上达到理想的加速

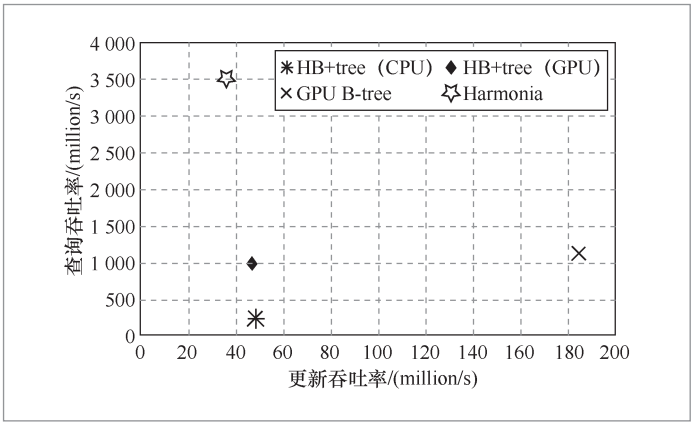


图 4 HB+tree、Harmonia 与 GPU B-tree 的性能分布

效果，如**图5**所示。他们将树结构划分为键区域和孩子节点信息区域，其中键值区域（键数组）存储原有树形结构节点中的键值信息，孩子节点信息区域存储每个节点的前缀和的信息，每个节点第一个孩子前的总节点数目被称为该节点的前缀和。通过前缀和可以很容易地计算出该节点第一个孩子节点的位置以及该节点包含的孩子节点的数目。相比于传统的树结构的存储方式，这种基于前缀的树形结构组织方式虽然引入了一定的额外计算，但却不需要存储所有孩子节点的指针信息，从而节省树形结构的存储开销。这种设计方式能够有效地减少B+tree结构的体积，并充分地利用GPU上不同类型的存储器缓存高时延的全局存储器访问。另外，Harmonia还基于B+tree数据结构设计了两种搜索优化访问：基于预排序的搜索优化和基于窄线程组的搜索优化策略，以期减少GPU上B+tree检索的内存访问分歧、执行分歧，并提高计算硬件的使用率。Awad等人[24]提出的GPU B-tree针对B-tree索引结构在GPU的更新及查询操作进行设计并实现。在该研究成果中，他们针对GPU的缓存行，对Blink-tree数据结构进行了特别的设计，以期获得更好的缓存效果。另外，他们采用了线程束配合工作的策略，将任务以线程束的线程为粒度分发，执行的过程则采用整个线程束合作完成的策略，以期在工作量和执行分歧之间达到一定的平衡。在GPU B-tree上，作者还将传统的B-tree索引结构的锁策略[34]应用到了GPU上，如预分类机制、重启机制等。

FPGA由于其电路芯片逻辑的可修改性和灵活性受到了各类优化方案的青睐。Yang等人[35]、Heinrich等人[36]、Qu等人[37]均使用该芯片对索引树结构进行了优化。其中Heinrich等人[36]基于FPGA提出了混合的索引结构，它支持CPU与FPGA端异

构协同工作。该混合索引以B+tree为基础，在CPU端存储叶子节点或者多层底层节点，在FPGA端存储内部节点或者多层的高层节点。通过FPGA的并行加速提升了混合索引上层的搜索速度，以此提升索引结构的整体处理效率。

4.3 基于分布式的优化

RDMA硬件目前已广泛应用于各种数据中心，可以显著提升网络的通信性能，为分布式并发索引结构的性能优化提供了可能。RDMA作为一种新型的网络硬件，可以减少传统远程过程调用带来的时延，有效地提高了分布式索引结构总事务的执行速度，使得原来受限于网络速率的分布式并发索引在性能上得到了有效的提升。

DrTM[38]的工作中将RDMA与HTM硬件的特性进行融合，使得分布式事务同步能够降低时延，并且使用这两种硬件技术可使跨机器的并发事务保持顺序性。DrTM以Hash表索引结构为基础，使用以HTM为单机的事务提供事务保护，在多机之间则采用类两段锁的方式配合HTM来保证事务的正确性，其中在远程访问内存时使用单边的RDMA提高访问的效率。DrTM+R[11]在DrTM的基础上，引入了乐观控制策略，进一步提升整体性能，并为系统高可用性提供更好的支持。

由Mitchell等人[39]提出的Cell（一种分布式B树存储）可基于RDMA的特性解决传统分布式存储中服务器端负载过高时客户端请求无法高效处理的问题。如**图6**所示，其中R为根节点，L为叶子节点。在该设计中，当客户端请求发送到服务器时，并发树结构的一个超级节点（64 MB）将会通过RDMA传输到客户端进行处理。这种方式避免了所有用户请求均由服务器处理的问题。另外，论文中还设计了

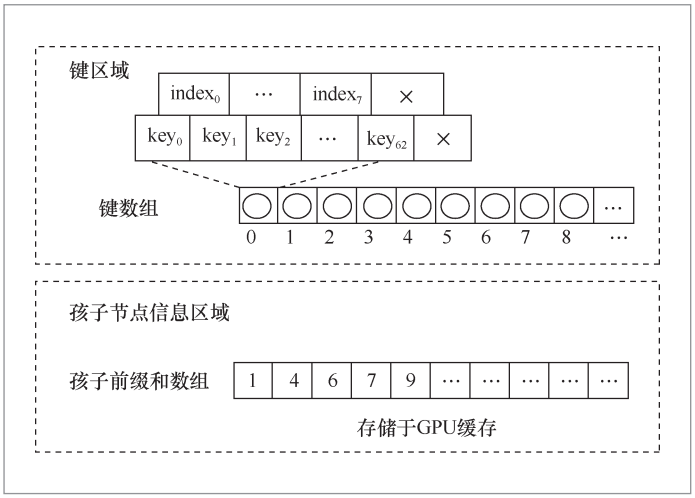


图 5 Harmonia 树结构 [23]

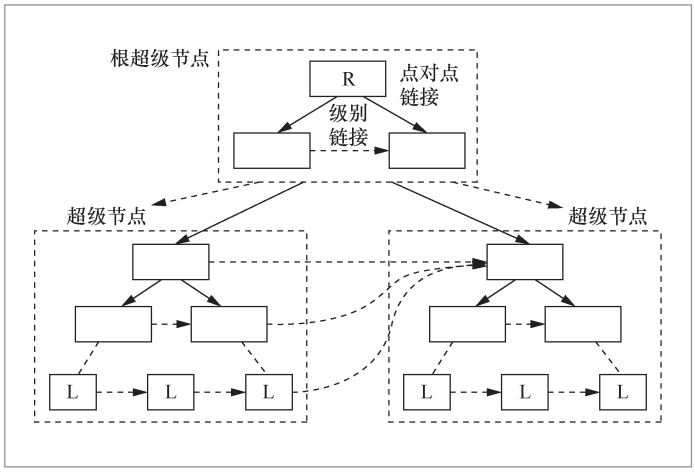


图 6 Cell 存储结构 [39]

一种动态调整执行模式的方案，该方案会根据不同工作集、CPU和网络等可用资源负载的动态变化情况进行动态调度和调整，以达到更好的性能。

4.4 基于NVM的优化

非易失性内存（non volatile memory，NVM）是一种新型内存硬件，它能够在系统关闭电源的情况下保存数据内容。而且，NVM的访问速度介于DRAM与SSD之间，能够很好地保证数据访问的效

率。这些特性能够为传统的数据存储领域带来新的性能提升，将数据库的数据与索引完全存储在非易失性存储器中，可实现高性能的访问，并快速地从错误中恢复系统。

当前已经有许多关于该方向的研究成果[40-43]，其中BzTree是由Arulraj等人[44]提出的一种充分利用NVM硬件特点的索引树结构。为了充分利用现代CPU的高并行性特点，部分现代存储器数据库使用无锁索引结构提供高效的并发处理能力，例如Bw-tree和跳跃表（skip list）等。为了在NVM硬件上实现高性能的索引结构，BzTree分析了Bw-tree的优缺点，并在设计上同样采用了无锁结构。为了实现这种无锁操作，BzTree利用一种被称为PMwCAS的比较交换指令实现高性能的操作。另外，该指令在NVM上使用时也提供持久性保证。BzTree相比于其他的基于NVM的设计，具有无锁设计、不需要基于NVM设计恢复机制、能够在NVM和传统存储器之间自由切换等特点。在NVM上使用雅虎公司推出的云数据库基准测试套件（Yahoo! cloud serving benchmark，YCSB）进行测试，结论是BzTree性能优于Bw-Tree，这证明使用新的硬件特性能够取得较好的性能。

5 结束语

伴随着底层数据量爆炸式的增长和用户数量的激增，如何高效地访问底层数据成为各个大数据系统面临的核心挑战。作为一种能有效组织数据并提供高性能访问的数据结构，并发索引结构已成为各种大数据系统的核心模块。然而在大数据时代，并发索引结构的易用性和性能仍然存在一定的不足，这些问题急需解决。本文探讨了当前学术界和工业界通过设计新的并发控制策略或者基于新型硬件特性来提高并发索引结构的易用性和高性能的各类方法，并对现有的研究成果进行了总结。

当前，信息技术日新月异，各种硬件结构不断推陈出新。这些新型硬件的出现为设计更好的并发索引结构提供了潜在的机会。然而，新型硬件的特性与传统硬件存在一定的差异，已有并发索引结构主要是面向传统硬件架构进行设计和优化的，无法充分发挥各种新型硬件的特性。因此，面向新型硬件的并发索引结构的设计和优化将得到越来越多的关注，并将得到越来越广泛的应用。

参考文献:

[1] GRAEFE G, KUNO H. Modern B-tree techniques[J]. Foundations & Trends in Databases, 2011, 3(4): 203-402.

[2] SHAHVARANI A, JACOBSEN H A. A hybrid B+-tree as solution for in-memory indexing on CPU-GPU heterogeneous computing platforms[C]//The 2016 International Conference on Management of Data, June 26-July 1, 2016, San Francisco, USA. New York: ACM Press, 2016: 1523-1538.

[3] VAISMAN A, ZIMÁNYI E. Data warehouses: next challenges[J]. Lecture Notes in Business Information Processing, 2011, 96: 1-26.

[4] VASSILIADIS P, SIMITSIS A. Near real time ETL[M]//New trends in data warehousing and data analysis. Boston: Springer, 2009: 1-31.

[5] BAYER R, SCHKOLNICK M. Concurrency of operations on B-trees[J]. Acta Informatica, 1977, 9(1): 1-21.

[6] LEHMAN P L. Efficient locking for concurrent operations on B-trees[J].

ACM Transactions on Database Systems (TODS), 1981, 6(4): 650-670.

[7] MAO Y, KOHLER E, MORRIS R T. Cache craftiness for fast multicore key-value storage[C]//The 7th ACM European Conference on Computer Systems, April 10-13, 2012, Bern, Switzerland. New York: ACM Press, 2012: 183-196.

[8] CHA S K, HWANG S, KIM K, et al. Cache-conscious concurrency control of main-memory indexes on shared-memory multiprocessor systems[C]//The 27th International Conference on Very Large Data Bases, September 11-14, 2001, Roma, Italy. San Francisco: Morgan Kaufmann Publishers Inc., 2001: 181-190.

[9] LEVANDOSKI J J, LOMET D B, SENGUPTA S. The Bw-Tree: a B-tree for new hardware platforms[C]// 2013 IEEE 29th International Conference on Data Engineering (ICDE), April 8-12, 2013, Brisbane, Australia. Piscataway: IEEE Press, 2013: 302-313.

[10] SEWALL J, CHHUGANI J, KIM C, et al. PALM: parallel architecture-friendly latch-free modifications to B+ trees on many-core processors[J]. Proceedings of the VLDB Endowment, 2011, 4(11): 795-806.

[11] CHEN Y, WEI X, SHI J, et al. Fast and general distributed transactions using RDMA and HTM[C]//The 11th European Conference on Computer Systems, April 18-21, 2016, London, UK. New York: ACM Press, 2016.

[12] WANG X, ZHANG W, WANG Z, et al. Eunomia: scaling concurrent search trees under contention using HTM[C]//The 22nd ACM SIGPLAN Symposium on Principles and Practice of Parallel Programming, February 4-8, 2017, Austin, USA. New York: ACM Press, 2017: 385-399.

[13] ZHANG W, WANG X, JI S, et al. Scaling concurrent index structures under contention using HTM[J]. IEEE Transactions on Parallel and Distributed Systems, 2018, 29(8): 1837-1850.

[14] WANG Z, QIAN H, CHEN H, et al. Opportunities and pitfalls of multi-core scaling using hardware transaction memory[C]//The 4th Asia-Pacific Workshop on Systems, July 29-30, 2013, Singapore. New York: ACM Press, 2013.

[15] WANG Z, QIAN H, LI J, et al. Using restricted transactional memory to build a scalable in-memory database[C]//The 9th European Conference on Computer Systems, April 14-16, 2014, Amsterdam, the Netherlands. New York: ACM Press, 2014.

[16] CHEN S, PENG L. Efficient GPU hardware transactional memory through early conflict resolution[C]//2016 IEEE International Symposium on High Performance Computer Architecture(HPCA), March 12-16, 2016, Barcelona, Spain. Piscataway: IEEE Press, 2016: 274-284.

[17] HERLIHY M, MOSS J E B. Transactional memory: architectural support for lock-free data structures[M]. New York: ACM Press, 1993.

[18] XU Y, WANG R, GOSWAMI N, et al. Software transactional memory for GPU architectures[C]//Annual IEEE/ACM International Symposium on Code Generation and Optimization, February 15-19, 2014, Orlando, USA. New York: ACM Press, 2014.

[19] POLLARI-MALMI K, SOISALON-SOININEN E, YLONEN T. Concurrency control in B-trees with batch updates[J]. IEEE Transactions on Knowledge and Data Engineering, 1996, 8(6): 975-984.

[20] SHNEIDERMAN B. Batched searching of sequential and tree structured files[J]. ACM Transactions on Database Systems (TODS), 1976, 1(3): 268-275.

[21] ARGE L, HINRICHS K H, VAHRENHOLD J, et al. Efficient bulk operations on dynamic R-trees[J]. Algorithmica, 2002, 33(1): 104-128.

[22] GRAEFE G. B-tree indexes for high update rates[J]. ACM Sigmod Record,

2006, 35(1): 39-44.

[23] YAN Z, LIN Y, PENG L, et al. Harmonia: a high throughput B+ tree for GPUs[C]// The 24th ACM SIGPLAN Symposium on Principles and Practice of Parallel Programming(PPoPP), February 16-20, 2019, Washington, DC, USA. New York: ACM Press, 2019: 133-144.

[24] AWAD M A, ASHKIANI S, JOHNSON R, et al. Engineering a high-performance GPU B-Tree[C]// The 24th ACM SIGPLAN Symposium on Principles and Practice of Parallel Programming(PPoPP), February 16-20, 2019, Washington, DC, USA. New York: ACM Press, 2019.

[25] RAO J, ROSS K A. Cache conscious indexing for decision-support in main memory[C]//The 25th International Conference on Very Large Data Bases, September 7-10, 1999, Edinburgh, Scotland. San Francisco: Morgan Kaufmann Publishers Inc., 1999: 78-89.

[26] RAO J, ROSS K A. Making B+-trees cache conscious in main memory[C]// The 2000 ACM SIGMOD International Conference on Management of Data, May 15-18, 2000, Dallas, USA. New York: ACM Press, 2000: 475-486.

[27] CHEN S, GIBBONS P B, MOWRY T C. Improving index performance through prefetching[M]. New York: ACM Press, 2001.

[28] CHEN S, GIBBONS P B, MOWRY T C, et al. Fractal prefetching B+-trees: optimizing both cache and disk performance[C]//The 2002 ACM SIGMOD International Conference on Management of Data, June 3-6, 2002, Madison, USA. New York: ACM Press, 2002: 157-168.

[29] HANKINS R A, PATEL J M. Effect of node size on the performance of cache-conscious B+-trees[C]//The 2003 ACM SIGMETRICS International Conference on Measurement and Modeling of Computer Systems, June 11-14, 2003, San Diego, USA. New York: ACM Press, 2003: 283-294.

[30] KIM C, CHHUGANI J, SATISH N, et al. FAST: fast architecture sensitive tree search on modern CPUs and GPUs[C]// The 2010 ACM SIGMOD International Conference on Management of Data, June 6-10, 2010, Indianapolis, USA. New York: ACM Press, 2010: 339-350.

[31] KACZMARSKI K. Experimental B+-tree for GPU[J]. Advances in Databases and Information Systems, 2011(2): 11.

[32] KACZMARSKI K. B+-tree optimized for GPGPU[C]//OTM Confederated International Conferences "On the Move to Meaningful Internet Systems", September 10-14, 2012, Rome, Italy. Heidelberg: Springer, 2012: 843-854.

[33] DAGA M, NUTTER M. Exploiting coarse-grained parallelism in b+ tree searches on an apu[C]//2012 SC Companion: High Performance Computing, Networking, Storage and Analysis(SCC), November 10-16, 2012, Salt Lake City, USA. Piscataway: IEEE Press, 2012: 240-247.

[34] GRAEFE G. A survey of B-tree locking techniques[J]. ACM Transactions on Database Systems (TODS), 2010, 35(3): 16.

[35] YANG Y H E, PRASANNA V K. High throughput and large capacity pipelined dynamic search tree on FPGA[C]//The 18th Annual ACM/SIGDA International Symposium on Field Programmable Gate Arrays, February 21-23, 2010, Monterey, USA. New York: ACM Press, 2010: 83-92.

[36] HEINRICH D, WERNER S, STELZNER M, et al. Hybrid FPGA approach for a B+ tree in a semantic web database system[C]//2015 10th International Symposium on Reconfigurable Communication-centric Systems-on-Chip (ReCoSoC), June 29-July 1, 2015, Bremen, Germany. Piscataway: IEEE Press, 2015: 1-8.

[37] QU Y, PRASANNA V. Scalable high-throughput architecture for large balanced

tree structures on FPGA[C]//The ACM/SIGDA International Symposium on Field Programmable Gate Arrays, February 11–13, 2013, Monterey, USA. New York: ACM Press, 2013: 278.

[38] WEI X, SHI J, CHEN Y, et al. Fast in-memory transaction processing using RDMA and HTM[C]//The 25th Symposium on Operating Systems Principles, October 4–7, 2015, Monterey, USA. New York: ACM Press, 2015: 87–104.

[39] MITCHELL C, MONTGOMERY K, NELSON L, et al. Balancing CPU and network in the cell distributed B-tree store[C]//2016 USENIX Annual Technical Conference, June 22–24, 2016, Denver, USA. Berkeley: USENIX Association, 2016: 451–464.

[40] CHEN S M, QIN J. Persistent b+-trees in non-volatile main memory[J]. Proceedings of the VLDB Endowment, 2015, 8(7): 786–797.

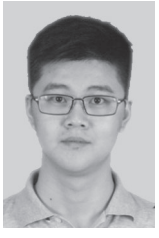
[41] CHEN S M, GIBBONS P B, NATH S. Rethinking database algorithms for phase change memory[C]//The 5th Biennial Conference on Innovative Data Systems Research, January 9–12, 2011, Asilomar, USA. [S.l.:s.n.], 2011.

[42] YANG J, WEI Q S, CHEN C, et al. NV-Tree: reducing consistency cost for NVM-based single level systems[C]//The 13th USENIX Conference on File and Storage Technologies(FAST15), February 16–19, 2015, Santa Clara, USA. Berkeley: USENIX Association, 2015: 167–181.

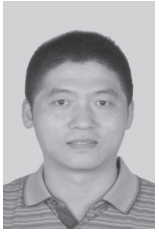
[43] OUKID I, LASPERAS J, NICA A, et al. FPTree: a hybrid SCM-DRAM persistent and concurrent B-tree for storage class memory[C]//The 2016 International Conference on Management of Data, June 26–July 1, 2016, San Francisco, USA. New York: ACM Press, 2016.

[44] ARULRAJ J. BzTree: a high-performance latch-free range index for non-volatile memory[C]//The 44th International Conference on Very Large Data Bases, August 27–31, 2018, Rio De Janeiro, Brazil. New York: ACM Press, 2018: 553–565.

作者简介



严赵峰（1993– ），男，复旦大学并行处理研究所硕士生，主要研究方向为异构计算。



张为华（1974– ），男，博士，复旦大学并行处理研究所教授，主要研究方向为编译器、系统软件、并行处理等。

收稿日期： 2019-05-06