Getting to the Root of Concurrent Binary Search Tree Performance

获取并发二进制搜索树性能的根源

国家光电硕1807班
M201872994
朱奥

- 介绍
- BSTs和实验装置
- 独立的BST性能
- 应用中的BST性能
- 建议

介绍

乐观的二进制搜索树(BSTs)

- 许多系统依赖乐观的并发搜索树来实现多核可扩展性
- 乐观树使用只读搜索，对共享内存的写入只发生在修改数据结构
- 并行运行
- 避免语义上不冲突的操作之间的同步争用

有一些问题

- 实现乐观树设计的全部性能优势远不简单
- 它们的性能受到难以识别和解决的系统软件和硬件子系统的微妙交互的影响

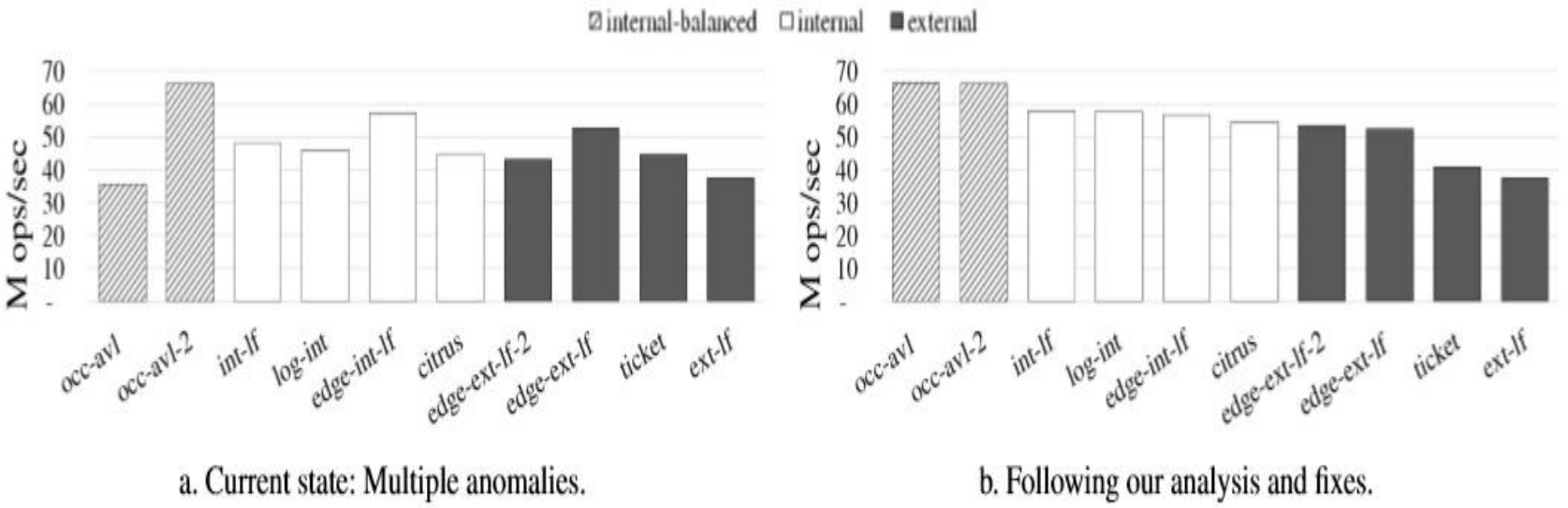
介绍

目标：解释和解决这种出乎意料的绩效结果

原因

- 我们发现性能异常是由BSTs与系统软件和硬件子系统的多种性能退化交互作用造成的，这些交互作用主要与缓存效应有关
- 将BST作为索引结构部署在内存数据库中，我们发现也存在类似的异常

介绍



a. Current state: Multiple anomalies.

b. Following our analysis and fixes.

图1：意外的BST性能结果

BSTs和实验装置

- 分析了8个BST算法的C实现，其中两个有独立的实现，总共有10个实现
- 实现了标准的键/值映射操作、查找、插入和删除
- 除了内部If以外，所有BST都具有只读遍历功能

BSTs和实验装置

name	synchronization technique	tree type	self-balance?	impl. source†
occ-avl [7]	fine-grained locks	part. ext	✓	[22]
occ-avl-2			✓	ASCYLIB
edge-int-lf [33]	lock-free	int		authors
log-int [14]	fine-grained locks	int		ASCYLIB
citrus [2]	fine-grained locks	int		authors
int-lf [23]	lock-free	int		ASCYLIB
edge-ext-lf [32]	lock-free	ext		authors
edge-ext-lf-2				ASCYLIB
ticket [12]	fine-grained locks	ext		authors
ext-lf [15]	lock-free	ext		ASCYLIB

表1：BST实施研究（按预期的搜索性能排序）

BSTs和实验装置

system name	*abu-dhabi*	*haswell*
processors	4× AMD Opteron 6376 (Abu Dhabi)	2× Intel Xeon E7-4830 v3 (Haswell)
# cores/proc	16 (2 dies w/ 4 modules, 2 cores per module)	12 (24 hyperthreads)
core freq.	2.3 GHz	2.1 GHz
L1d cache	16 KiB, 4-way	32 KiB, 8-way
L2d cache	2 MiB, 16-way (per mod.)	256 KiB, 8-way
last-level cache (LLC)	2 × 8 MiB, 64-way (shared, per die)	30 MiB, 20-way (shared)
interconnect	6.4 GT/s HyperTransport (HT) 3.0	6.4 GT/s QuickPath Interconnect (QPI)
memory	128 GiB Sync DDR3-1600 MHz	128 GiB Sync DDR3-1600 MHz

表2：硬件平台

独立的BST性能

BST实施问题

- 膨胀的节点：大多数实现不必要地膨胀了树节点，从而减少了可以容纳在缓存层次结构的每一级中的树的数量
- 分散的字段：通常由遍历读取的字段（键/左/右，以及与检测并发树修改相关的字段）应首先位于节点结构中，以最小化搜索在遍历节点时访问两个高速缓存线的机会
- 重遍历：edge - int - lf和ext - lf将所有操作都基于一种共享遍历方法，因此最终会给查找操作增加负担

独立的BST性能

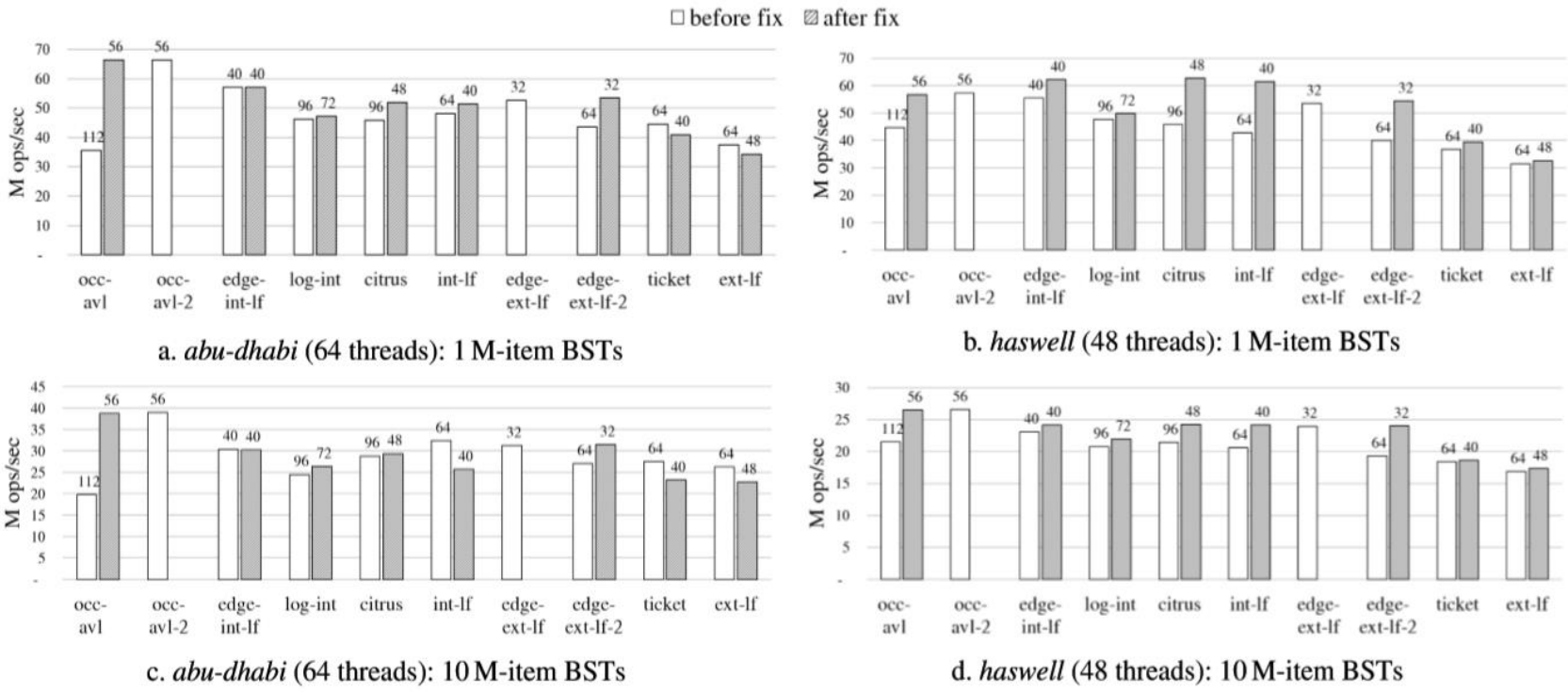


图2：修正BST实施问题的影响（条形顶部的数字显示BST节点大小）

独立的BST性能

意想不到的结果仍然存在

- 为什么减少节点大小会损害阿布扎比10m项目int - lf、ticket和ext - lf的吞吐量？
- 为什么int - lf受益于将64字节节点减少到48字节节点，比haswell上的ticket多得多？
- 为什么log - int的性能比其他未升级的内部BST差？

独立的BST性能

内存布局问题

- 独立自由列表分配
- 跨越高速缓存线
- 由于分配模式，缓存未得到充分利用
- 并置孩子节点

独立的BST性能

预取问题

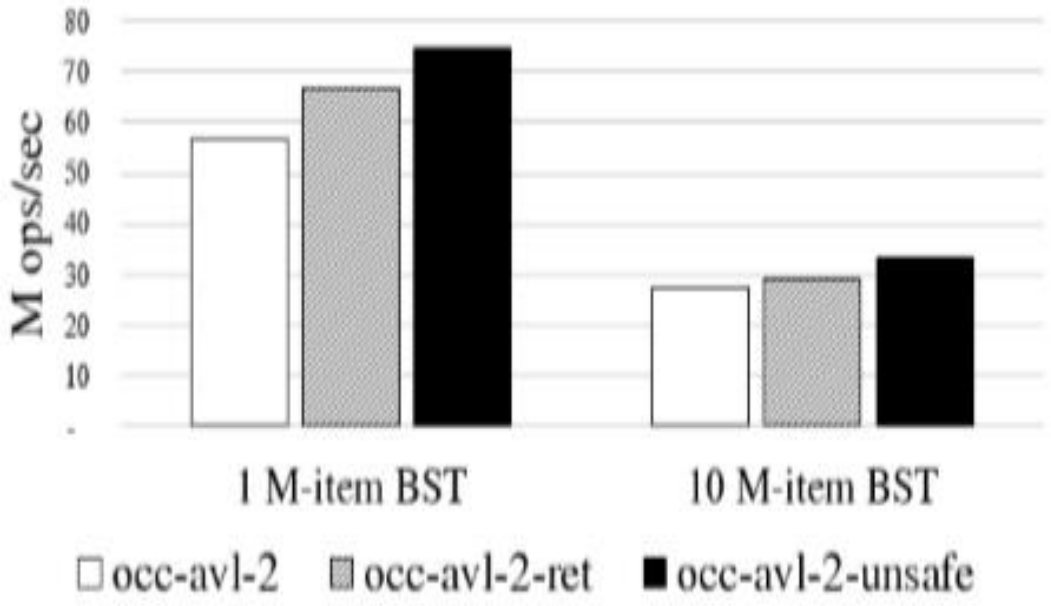


图3：occ-avl-2（haswell）中OCC变化的影响。

应用中的BST性能

- 这些性能问题是否仅仅是微基准测试的产物，或者类似的问题是否出现在更复杂的软件中。我们研究了一个名为DBx1000 的内存数据库管理系统，该系统用于多核数据库研究
- DBx实现了一个关系数据库，它包含一个或多个表，每个表都由一系列行组成。它提供了各种不同的并行控制机制，允许进程访问表和行

应用中的BST性能

为了分析DBx中各种BST实现的性能，我们使用了：

- Yahoo!云服务基准(YCSB)和跨行动处理性能委员会的TPC - C基准
- YCSB中的事务相对简单，主要是工作量大的单一指数
- TPC-C具有更复杂的事务，有许多索引和许多写入

应用中的BST性能

YCSB

- 运行YCSB核心的一个子集，其中包含一千万行的单个表
- 每个线程执行固定数量的事务，当第一个线程完成其事务时，执行终止
- 使用内存分配器的几个单独的实例来实现分离:一个用于YCSB，一个用于我们希望从其他对象类型中分离的每种类型的对象

应用中的BST性能

与微基准的比较

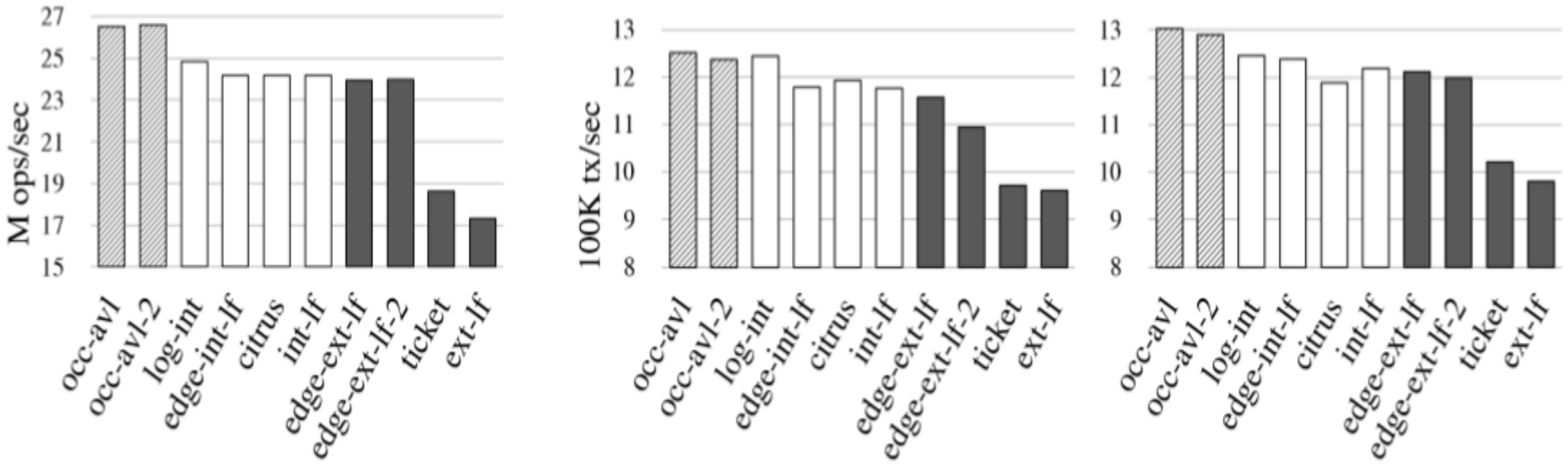


图4 :微基准与YCSB结果的比较: (左) 10MBST项目的微基准(中)没有隔离的YCSB，(右)有隔离的YCSB。

应用中的BST性能

内存布局问题

- 由于分配模式，缓存未得到充分利用
- 意外修复内存布局问题
- 不必要的页面散布

应用中的BST性能

TCP-C

- 为批发供应商的订单输入环境模拟了一个大规模在线交易处理应用程序
- 代表了“任何必须管理、销售或分销产品或服务的行业”的业务活动
- 假设业务运营是围绕固定数量的仓库组织的，每个仓库为多个地区服务

应用中的BST性能

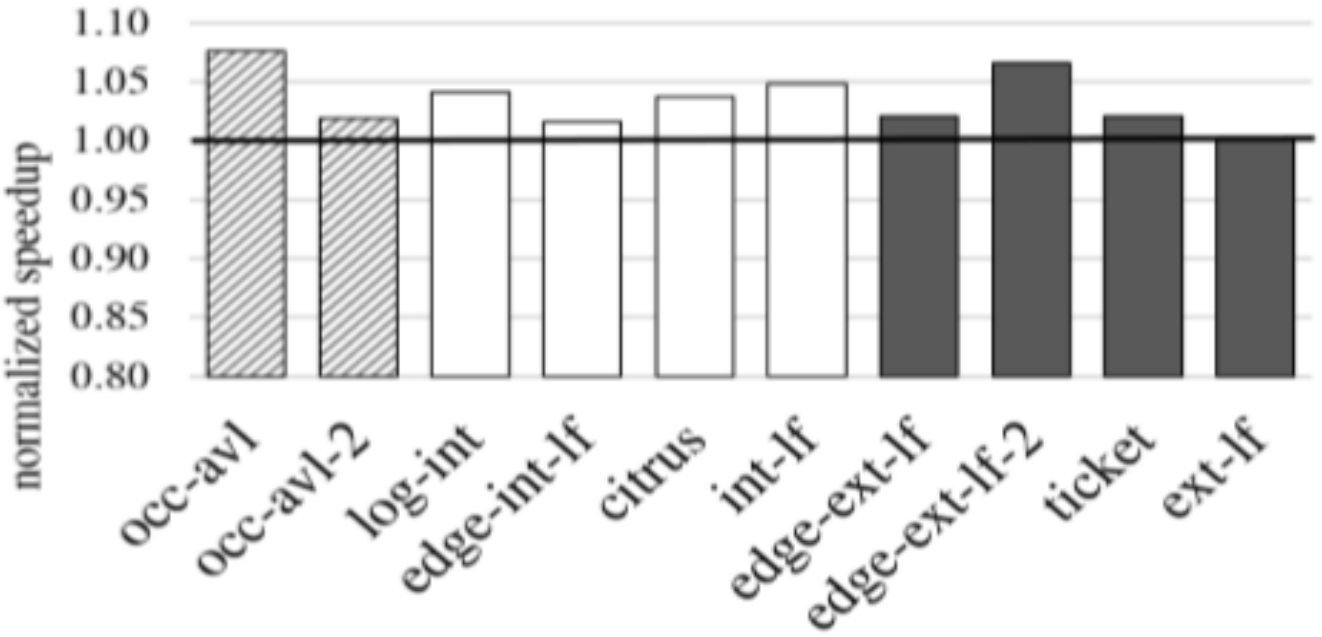


图5：TPC-C：基线与改进的实施方式

应用中的BST性能

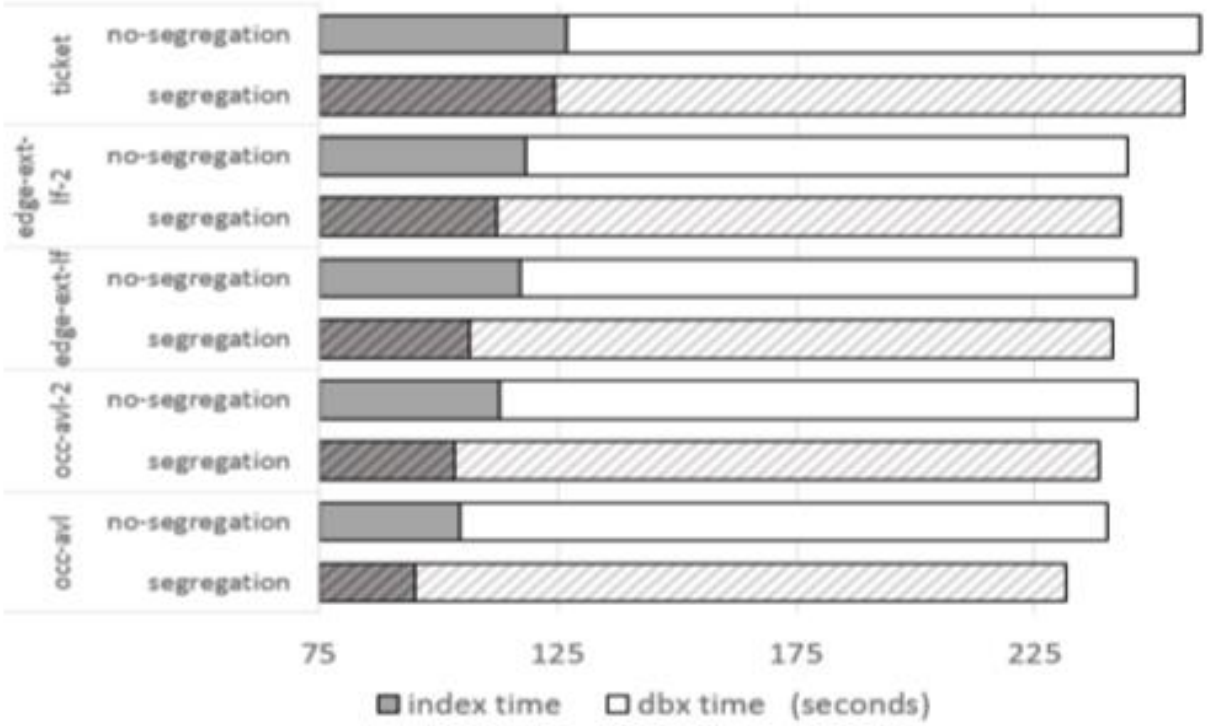


图6：隔离对TPC-C的影响

建议

对数据结构设计者和实现者：

- 如果与节点相邻的高速缓存线通常包含其他对象，则节点可能未充分利用高速缓存
- 使用填充来避免错误共享，特别是在频繁访问的节点和其他程序数据之间。且这种填充应该考虑预取
- 不分青红皂白地填充所有节点可能会降低性能，因为这会减少LLC中适合的节点数量

建议

对使用数据结构的程序员：

- 数据结构导入程序会合并两个内存空间，并可能会产生或消除错误共享或缓存未充分利用的问题。
- 应该(a)检查数据结构和程序的组合内存布局，并解决这些问题，或者(b)使用单独的分配器分离数据结构的内存

建议

对内存分配器的设计者和实现者：

- 为分配分离提供一个接口
- 为内存布局检查提供接口或工具，以允许确定(1)对象到大小类的映射；(2)哪些对象类型经常位于存储器中彼此靠近(3)将对象分配到LLC中的高速缓存集中

谢谢