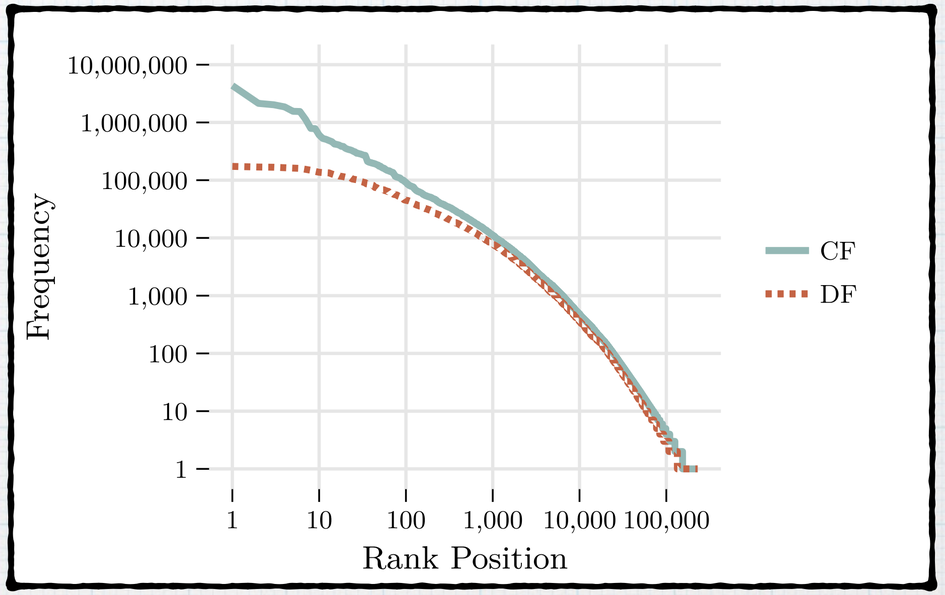
Collision Resolution in Hash Tables for Vocabulary Accumulation During Parallel Indexing

Matt Crane & Andrew Trotman

Introduction

* Vocabulary accumulation
  * Use some data structure
* Term frequency skewed
  * Both collection and document
* Parallel indexing



Some Data Structure?

* Lots to choose from
  * Hash tables generally agreed to be fastest
* Collision resolution in hash tables
  * Chaining uses a secondary structure to do so
  * Some data structure for chaining

Parallel Indexing

* Indexing documents in embarrassingly parallel
  * So index them separately
* Combine per document indexes together
  * Single threaded merging (assign docids etc.)
* Analogous to Map/Reduce

Collections

	Documents	Unique Terms	Total Terms
Wall Street Journal	173K	230K	83M
.GOV	1.2M	54M	1.1B
.GOV2	25M	37M	20B
ClueWeb09 Cat. B	50M	96M	55B

Parallel Indexing Speedup in ATIRE

* When document indexer discovers new term (for that doc):
  * Search the global vocabulary, storing reference if found
* When merging:
  * If already have a reference, update term details
  * Otherwise, upsert the term

Some Collision Resolution Structure?

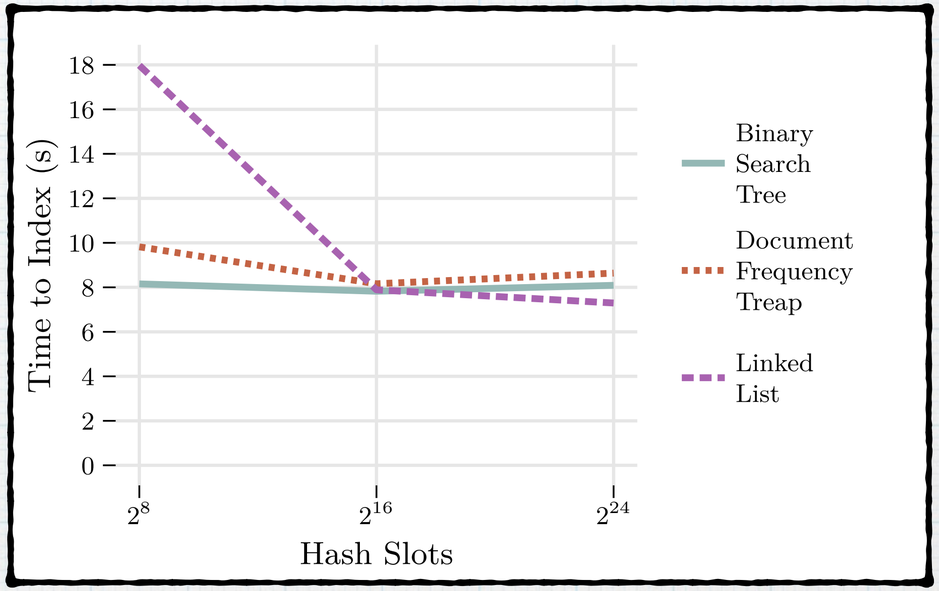
* Multi reader, single writer
* For the chained structures tested, achievable with atomic compare-and-swap operations
* Global vocabulary is consulted for each document occurrence rather than total occurrence

Structures Tested

* Linked Lists (insert-at-back heuristic)
* Binary Search Trees
* Periodic Self-Balancing BSTs
* Document-Frequency Treaps

Structure Performance a Function of Density

- Increase density by reducing number of hash slots available
- Results shown for Wall Street Journal:
  - Expected degradation for lists
  - Treaps always slower
  - PSBBSTs not shown



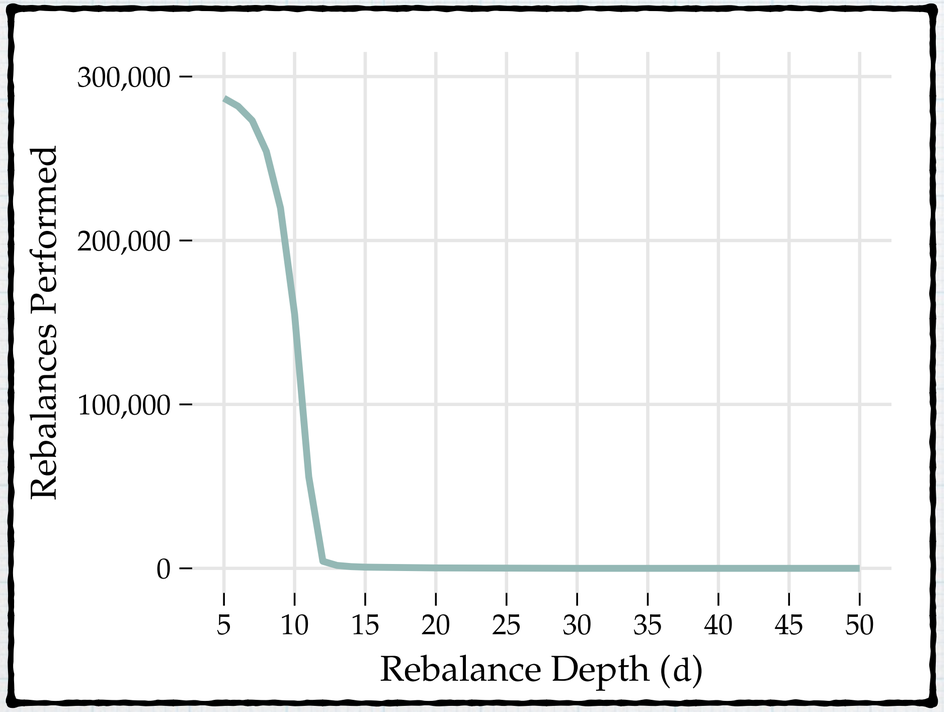
Periodic Self Balancing BSTs

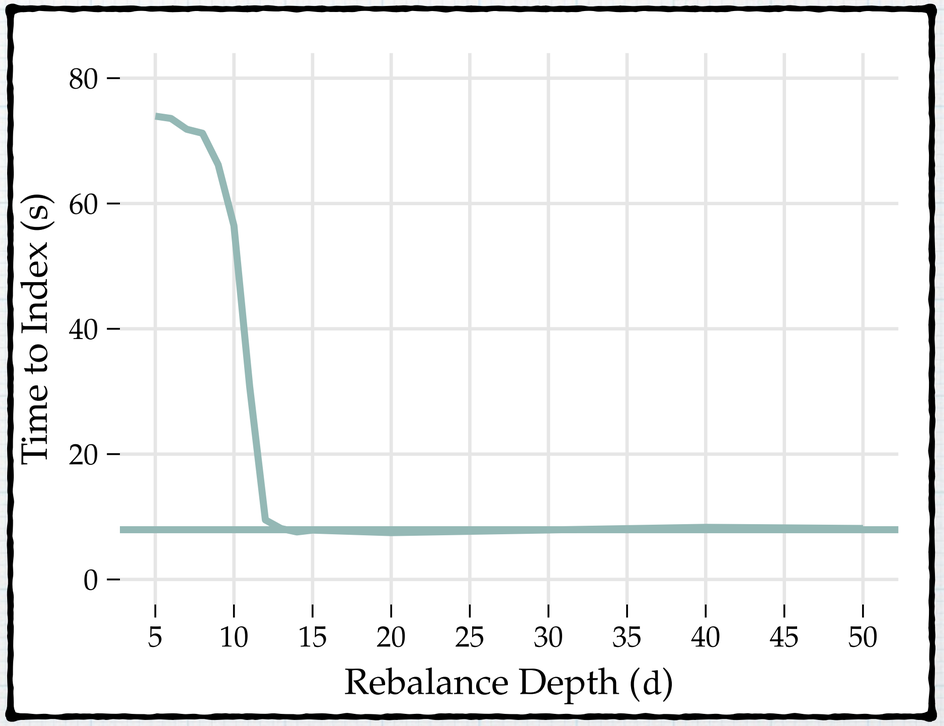
* BSTs are dependant on order of data inserted
  * Degrade to lists (at least two docs in CW1 2)
* So balancing the trees
  * Previous work shows splaying periodically to be better than always doing so
  * Balance when new term inserted at depth d

Periodic Self Balancing BSTs

* Day-Stout-Warren algorithm
* Degrade the tree to a list by right rotation
* Perform left rotations to restore complete BST
* Happens in place and in linear time

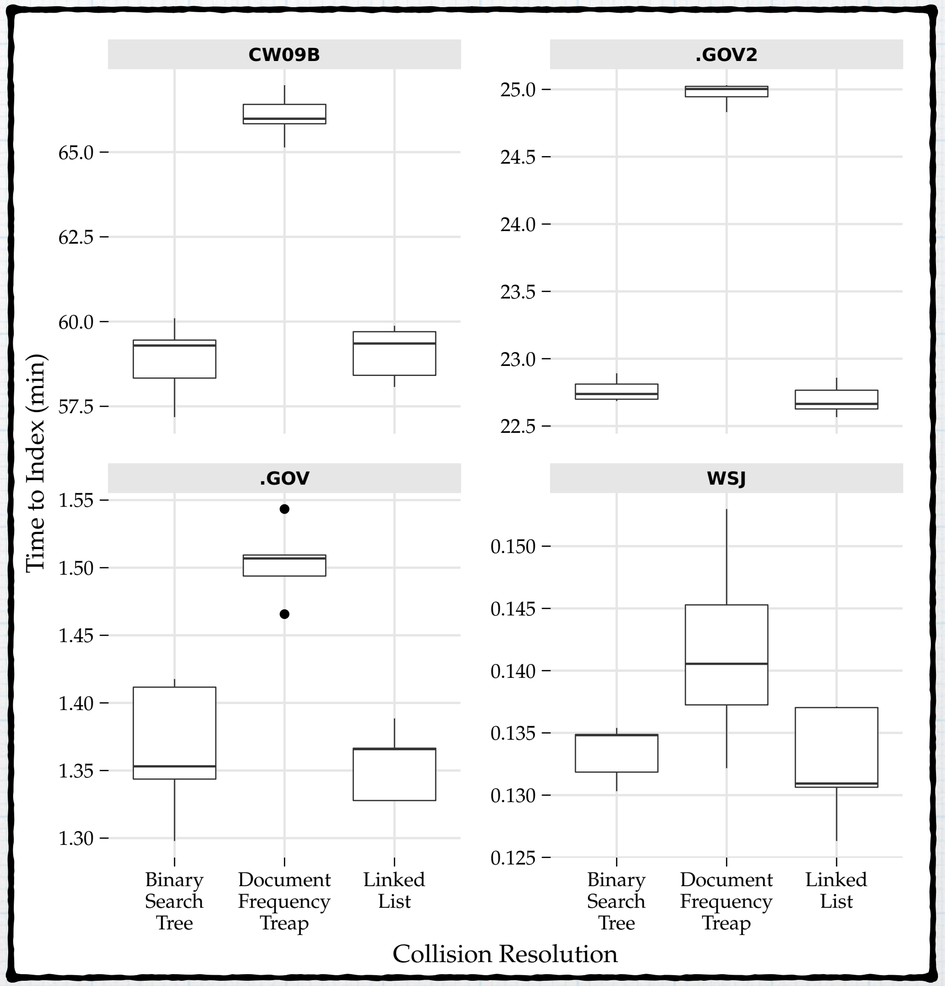
Periodic Self Balancing BSTs





Results

- PSBBSTs too parameter sensitive
- BST & Linked List equally good
  - Low density (Hash slots: 2^{24})
- Treap consistently worse



Document Frequency Treaps

* Treaps have, and maintain, two properties
  * Sorted ordering — same as BSTs
  * Heap property — in this case, document frequency
* Larger document frequencies are closer to root

Document Frequency Treaps

* If more frequent terms are closer to root, why always slower?
* One test for maintenance is cheap, but a lot are done
    * At least one comparison per document occurrence per term
* Can cause missed lookups while rotations are performed

Conclusions

* Single writer, multi reader structures
* Lookup feature saves substantial time
* Some structures are very sensitive to parameters
* Nicer theoretical structures can have higher computation costs

Future Work

* Smarter self-balancing trigger
* Periodic treapificiation

Questions?
// Comments