Like the Clappers - The Story of an Indexer

Vaughan Kitchen

November 5, 2019

At the Beginning

“ No student has ever written an indexer faster than mine ”

— Andrew Trotman

Months of Programming Later

engine	**time (s)**
ATIRE	8.27
cocomel	6.46
JASSjr	19.30
JASSjr-Java	29.78
rangahautia	17.75

Test Details

- WSJ collection: a 500mb XML file
- Run 3 times. Take the lowest time
- Late 2013 Mac - Intel Core i5-4570 @ 3.20GHz - 8GB 1600MHz DDR3

Now

“ No student has ever written a UTF-8 indexer faster than mine ”

— Andrew Trotman

What This Talk Will Be

- A lot of exploration through my Git history
- A little bit of guidelines for writing performant code

How to Make Fast

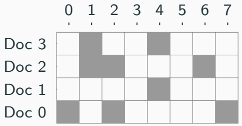
- Choose your language
- Choose your data structures
- Tune everything
- The last 20% is the hardest

Language Choice

- C
- C++
- Fortran
- Anything else at your peril
- (I'm aware of Rust. Prove me it's faster)

Data Structures - Building a Search Engine

- Bitstring Signature Files



- Inverted Index

of	$\rightarrow$	1
Otago	$\rightarrow$	1, 2, 3
University	$\rightarrow$	1, 2

First Attempt

- Commit: 5d671da
- Date: Aug 21, 2018
- Msg: Basic slow single word search
- Time: 353s
- Notes: Trees move slowly

```
std::map<std::string, std::vector<size_t> > postings;
do {
    token = tokenizer_next(tok);
    std::string word(token.value);
    std::vector<size_t> docs;
    postings[word] = docs;
    postings[word].push_back(token.doc_number);
} while (token.type != END);
```

Oh Man Bugs...

- Commit: 99f5cb6
- Date: Aug 31, 2018
- Msg: Switch to LIBAVL version of RBT to stop values getting lost
- Time: 20.18s
- Notes: Hash table with Red-Black Tree. Custom Malloc. Compression
- Revert features from here to show the effect on performance

Smaller is Faster?

- Commit: 88f99ca
- Date: Aug 30, 2018
- Msg: Compress postings
- Time (previous): 20.18s
- Time (removed): 26.18s
- Notes: 2642, 2646, 2656, 2657 $\rightarrow$ 2642, 4, 10, 1
- Notes: Variable byte compression means most values are 8bits

$n \log(k)$ **Beats** $O(n^2)$

- Commit: a069ca1
- Date: Aug 29, 2018
- Msg: log k merge
- Time (previous): 26.18s
- Time (removed): 243.83s - beats 353s with map

```
for (size_t gap = 1; gap < h->capacity; gap *= 2) {
    for (size_t i = 0; i < h->capacity; i += gap * 2) {
        if (h->store[i] == NULL) {
            h->store[i] = h->store[i+gap];
            continue;
        }
        rbt_kv_merge_left(h->store[i], h->store[i+gap]);
    }
}
```

Avoid Syscalls

- Commit: 5f6377c
- Date: Aug 22, 2018
- Msg: Custom malloc
- Time (previous): 26.18s - before compression, after log k merge
- Time (removed): 32.11s
- Notes: Take large blocks of memory from malloc and hand it out in chunks - this is a linear allocator

Know Your Data

- Commit: e2319b8
- Date: Sep 3, 2018
- Msg: Use plain BST to back hash table. Input guaranteed random and a BST takes less work than an RBT
- Time (previous): 20.18s
- Time: 13.72s
- Notes: Came from talking to Andrew. RBT messes up the branch prediction

Don't Believe Everything a Profiler Says

Profiling information (using gprof)		
Function	**% of total time**	**time (s)**
RBT_find()	47.13%	3.90
BST_find()	28.70%	1.50
RBT_insert()	0.48%	0.04
BST_insert()	0.19%	0.01

Good Answers Fast Win Over Right Answers

- Commit: a6ef328
- Date: Apr 27, 2019
- Msg: Reduce index size by capping term count
- Time (previous): 13.72s
- Time: 13.08s
- Notes: If a term occurs more than 255 times in a document it has negligible effect. Reducing memory usage however improves performance

```
struct posting {
    size_t id, diff;
    uint8_t count, *id_store;
    size_t id_capacity, id_length;
    dynamic_array<uint8_t> *counts;
};
```

Don’t Alloc What You Don’t Need

- Commit: f971c6c
- Date: May 13, 2019
- Msg: Reduce string allocation when parsing
- Time (previous): 13.08s
- Time: 11.96s
- Notes: Adding terms is rare. Looking up terms is common

```c
struct bst_kv_node *make_node(char *key, void *val) {
        struct bst_kv_node *n = memory_alloc(sizeof(*n));
        n->key = string_s_dup(key);
        n->val = val;
        n->link[0] = n->link[1] = NULL;
        return n;
}
```

Know Your Lifecycle

- Commit: 87d9ef7
- Date: May 14, 2019
- Msg: Use custom allocator in more places as it's generally faster
- Time (previous): 11.96s
- Time: 10.63s
- Notes: Overload the new method in C++ so that class instantiation goes through the linear allocator

Go Level When Available

- Commit: dee147d
- Date: May 15, 2019
- Msg: Slight performance gain in hash function. Increased robustness for NULLs
- Time (previous): 10.63s
- Time: 9.82s
- Notes: Modulo is slow - it requires repeat division

```
-    unsigned int hash = htable_word_to_int(key) % h->capacity;
+    unsigned int hash = htable_word_to_int(key) & 0x7FFF;
```

Rookie

- Commit: 30787ec
- Date: May 27, 2019
- Msg: Write the hash table as the index without conversion
- Time (previous): 9.82s
- Time: 9.02s
- Notes: Why convert the hash table to an ordered list when it can be searched directly? Rookie

Keep Related Data Together

- Commit: aae4014
- Date: Jun 8, 2019
- Msg: Refactor to be more truly C++
- Time (previous): 9.02s
- Time: 8.01s
- Notes: Low hanging fruit of value vs reference types gobbled up. Closer storage of related values

Inline Everything

- Commits: f33d200, b0b9fe
- Date: Jun 10, 2019
- Msg: Faster implementations of ctype functions
- Msg: Inline strcmp
- Time (previous): 8.01s
- Time: 6.43s
- Notes: Function calls are expensive
- Notes: Actually inlining everything would be slow due to throwing out the instruction cache

Things That Are Slow

- Function calls
- Branch misprediction
- Memory access
- System calls
- Bad algorithms

Questions?

Find me

- http://vaughan.kitchen
- https://github.com/vkitchen/cocomel

Questions?