Gaurav Singh | UFID: 36398850

COP 6726: Database Systems Implementation

Spring 2018
Weekly Assignment 5

Notes:

- Hashing Algorithm
- Let’s start with the assumption that we have enough memory, if we don’t we will deal with that later.
- Not having enough memory would mean we try external hashes because internal memory cannot hold all the information.
- How to we create these has functions?
- First consider how they are used, each of the tuple computes h(T) and then it is added to the corresponding bucket.
- Hash based schemes are little slower to build but are amazing are supporting lookups across the table.
- One of the things that became bad by switching to 64 bit architecture is that division started taking 80 cycles to complete compared to 40 on 32bit architecture.
- $h'(t) = h(t)\ \%\ B$
- Here B is the address size. Now modulus is what kills us with 80 cycles per instruction.
- So to avoid this massive waste, you try to find B which is in order of 2 and do bit shifting.
- Thus B=2^k and shift by k to divide it.
- Mercene primes 2^k -1 are thus a really important factor and helps us choose B which is very good for our specific situations.
- Two different strategies for making multi-threaded systems where discussed in quite some detail.

Practical:

I saw a very cool JS program on emulating the functionality of HashCode() from Java on Stack Overflow and analyzing it was pretty fun and gave me a neat little function to use later on.

Code from Stack overflow:

```
String.prototype.hashCode = function(){

       var hash = 0;

       if (this.length == 0) return hash;

       for (i = 0; i < this.length; i++) {

               char = this.charCodeAt(i);

               hash = ((hash<<5)-hash)+char;
```

```
                hash = hash & hash; // Convert to 32bit integer
        }
        return hash;
}
```

Firstly I learnt the pattern behind the whole thing .

s[0]*31^(n-1) + s[1]*31^(n-2) + ... + s[n-1]

One cool thing in this code was that they had already bit shifted the code to avoid a multiply.

```
hash = ((hash<<5)-hash)+char;
```

instead of hash += char*Math.pow(31,(lengthofString -i));

That was really neat another improvement, not included in this original code , and which should help is to put the value of string length in an actual variable. This may seem like a childish way to code but I believe by storing that value, you would attain two things.

a) Remove the need to compute the length again.
b) Make it predictable for the processor to serialize the whole thing more efficiently.

I also found this column store based JS DB called Datavore on github
https://github.com/StanfordHCI/datavore

This is supposed to provide fast(under 100ms) access to over million data points while running in browser. This seemed quite underwhelming at first but then I tried to run it in browser to see how fast it was, and it was really good.
The project seems mostly abandoned with only a few commits, but it seems a great little project. The fact that they managed to do it without any other dependencies is pretty cool.

I still don’t have any use for a column store DB but I learnt yet another tool in case I need it and I am going through the code of the whole thing and see if I can pick any cool tricks they used elsewhere.