Hashing and Bloom filters key lookup comparison

names

March 24, 2020

Abstract

1 Introduction

Hashing methods and probabilistic data structures such as Bloom filters have and increasing use nowadays with the necessity of fast and efficient data structures for dictionary abstraction. Dictionaries provide use with a way to organize data and enable a fast search and manipulation. In this report we wish to compare different different hash table implementations and bloom filters in the task of key lookup in the dictionary. In this research we compare three types of dictionaries implementations Open Addressing Tables, Separate Chaining Tables and Bloom Filters.

2 Experiment pipeline and methodology

The experimentation process was a very tedious task since it required to take in account a lot of parameters. An experiment is divided into two parts the table build, where every n keys of a dictionary are added, and the table lookup where $2 * n + n * keyPercenatge$ keys are searched in each type of hash table and the bloom filters. In the lookup part $2 * n$ keys of the search are not in the table and should return an unsuccessful search, the rest of the keys should return a successful search. to the table and the table lookup, where given a series of keys .....

Parameters List	
Parameter	Definition
n	This is the number of keys that where inserted into the table
m	This is the size of the table
α	The table load factor $(\frac{n}{m})$
k	Number of hash functions to use for the bloom filters
$keyPercentage$	The percentage of keys that appear in the text files
$seed$	The initial number for the random integer generation
$maxLoop$	Max recursive probes for the Cuckoo hashing
Q	the number of repetitions per experiments

3 Data Generation

The data used in each of the experiments consisted of a sequence of unsigned integers, the data was written in two different files. One file contained the keys to insert into the dictionary and the other the text to find in the dictionary.

Linear Congruential Method

The randomness of the keys was based on the Linear congruential method (LCM) [5]. We needed to have experiments as reliable as possible thus we needed a way to control the data generation, that is why we used this method. The LCM is one of the oldest and best-known random number generators, it is also very easy to implement. This method generates a cyclic sequence of numbers based on an initial seed. Given a seed X_0 it is possible to generate the next number in the sequence

$$X_i = (X_{i-1} * a + c) \mod m \tag{1}$$

where a is called the multiplier, c the incrementer and m the modulus. The size of the sequence depends on the parameters used, independently on the initial seed. We used a combination of parameters that yield the maximum size sequence, that is parameters that yield m numbers. This ensured us that that there where never repeated numbers in either of the two files since the size of m was sufficiently big.

4 Experiment parameters

Size of n

The keys file contained n numbers and the text contained $2 * n + p * n$ where p is the percentage of keys that are already inserted in the dictionary. We had a lot of discussion in which n used and we tried different combinations but they yielded very similar results so we finally chosed an $n = 10e^7$. That happens to

be 10 million keys that where inserted into the dictionary, although the table size was much bigger. We tried to used a bigger size of n but we encountered the size of the files we created where extraordinary big, we tried to divide the files but that wasn't of much use since we stilled needed to save all the numbers in the hash tables. When we tried to to this the linux oom-killer killed to process because of the out of memory that was caused. Nevertheless other researches [3] used similar sizes of n so this size wasn't really an issue.

Number of repetitions Q

Each experiment was defined by a series of parameters to get the best results and avoid overfitting of the data to a single sample we repeated each experiment with $Q = 100$ that is we performed the same experiment 100 times and averaged the results.

Load factor α

The load factor is one of the variables that really helps into comparing each type of dictionary implementation, we used in total 9 different load factors ranging from 0.10 to 0.9 with an offset of 0.10. We could have subdivided the interval with a lower offset but after experimenting we found that there wasn't really a substantial difference.

Key percentage p

The key percentage represents the percentage of keys that are in the keys (inserted in the dictionary) that are in the search file (the second file where the lookups are done). This variable did not have a big impact in the results since we average the lokuup time of each key individually and even if we chosed a really low key percentage we had a big enough Q that it didn't matter. Thus we chosed $p = 0.5$;

Number of hash functions k

This was a difficult parameter to choose since it had to be chosen optimally because if had a very big k then the number of false positives might diminish, but only if the size of the table was big enough, on the other hand if we chosed a small value of k the table would fill more slowly but it was more probable to find to keys with the same bits mask. To decide into what to choose we simply tested different values of k with different results.

Hash functions

The hash functions we chosed where not the purpose of this experiment altough we tried to choose.

Random seed

The seed number is really a determining factor into how the dictionaries will perform since different seeds yield to different keys and conscuently differnet hash outputs. That is why for each experiment we chosed the same initial seed, altough the next seed was chosen at random based on the initial seed, this enabled reprodudability. In total a number of Q different seeds were selected, because the results of the Q repetitions where average this enabled us to remove any outliers that could happend choosing a specific seed value.

5 Results

6 Discussion

References

[1] Burton Howard Bloom. "Space/Time Tradeoffs in Hash Coding with Allowable Errors." In: *Communications of the ACM* (1970).

[2] Andrei Broder and Michael Mitzenmacher. "Network Applications of Bloom Filters: A Survey". In: *Internet Mathematics* (2004).

[3] Bernhard Grill. "A Survey on Efficient Hashing Techniques in Software Configuration Management". In: (2014).

[4] Adam Kirsch and Michael Mitzenmacher. "Less Hashing, Same Performance: Building a Better Bloom Filter". In: *Wiley InterScience* (2007).

[5] Donald Knuth. *The art of computer programming, Volume 2*. Addison Wesley, 2011.

[6] Donald Knuth. *The art of computer programming, Volume 3*. Addison Wesley, 2011.

[7] Professor Clark F. Olson and Carol Zander. "Hashing and Hash Tables." In: *University of Washington Bothell's web site.* (2017).

[8] Rasmus Pagh and Flemming Friche Rodler. "Cuckoo Hashing." In: *Brics: Basic Research in Computer Science.* (2001).

[9] Abubakar Sulaiman Gezawa Abdurra'uf Garba Saifullahi Aminu Bello Ahmed Mukhtar Liman and Abubakar Ado. "Comparative Analysis of Linear Probing, Quadratic Probing and Double Hashing Techniques for Resolving Collusion in a Hash Table." In: *International Journal of Scientific Engineering Research.* (2014).