COMP4420 - Advanced Design and Analysis of Algorithms
Project Proposal

Robin Swanson
umswans5@myumanitoba.ca
7615743

Collaborators: N/A

February 2014

After recent leaks of information from various online companies (e.g., the recent release of Target credit cards, Adobe emails and password hashes, etc.) tools based on Bloom Filters appeared on various websites to determine whether your email, account, or password may have been compromised. Upon investigation, and subsequent realization this type of filter had nothing to do with computer graphics, I decided this type of tool warranted further research on my part.

The bloom filter is a space efficient probabilistic data structure introduced by Burton Howard Bloom in 1970[1], able to test whether a value is in a given set or not. However, due to its probabilistic nature, false positives are possible while false negatives are not. This makes it an ideal data structure for something like checking if your email was compromised – better to think that it was when it wasn't and change your password than to erroneously think you are safe. However, it is commonly used as a preliminary hash function for data sets too large for a traditional hash table. If the Bloom Filter thinks it is in the set, we can do a more expensive test to verify. According to Bonomi et al.[2] only 10 bits are required per element to achieve a 1% false positive rate as opposed to traditional hashes which require $O(n + a)$ space .

For my project I propose the research and implementation of a bloom filter, as well as the more recent variation known as a Cascading Bloom Filter[3] (or Quotient Filter) which is aimed at reduce thrashing. After implementation comparisons could then be drawn with the use of a large data set (larger than available memory), looking at run times as well as the frequency of false positives. In addition, we can monitor and compare the amount of cache misses/thrashing through the use of profiling software such as Valgrind[1] and, more specifically, its cache profiler Cachegrind[2].

I aim to have both data structures implemented by March 1^{st}, and all of the testing completed by March 15^{th}. This will leave me approximately 15 days to assemble, prepare, and practice my presentation and another week or more to finish my final report. Preliminary research has already been completed and a representative data set of adequate size ($\geq$ 6GB) has already been acquired – and many more could easily be generated.

If time permits I may also explore other variations on the bloom filter such as the Bloomier Filter[4]. This will also provide some wiggle room if either of the previous algorithms prove too difficult or easy to implement and could provide further comparison. I may also explore different numbers and types of hashing functions however, from preliminary research, more computationally simple (non-cryptographic) functions are more than adequate when used to generate a sufficiently large number of hashes[5].

References

[1] "Space/time trade-offs in hash coding with allowable errors", B.H. Bloom; Magazine Communications of the ACM 13-7, 422-426 (1970)

[1] Valgrind - http://valgrind.org/
[2] Valgrind Tools - http://valgrind.org/info/tools.html#cachegrind

[2] "An Improved Construction for Counting Bloom Filters", F. Bonomi, et al,;
http://theory.stanford.edu/∼rinap/papers/esa2006b.pdf, accessed February 5^{th} 2014

[3] "Dont Thrash: How to Cache your Hash on Flash", Michael A. Bender, et al.;
http://static.usenix.org/events/hotstorage11/tech/final_files/Bender.pdf, accessed February 5^{th} 2014

[4] "The Bloomier Filter: An Efficient Data Structure for Static Support Lookup Tables", B. Chazelle, et al,;
https://www.cs.princeton.edu/∼chazelle/pubs/soda-rev04.pdf, accessed February 6^{th} 2014

[5] "Less hashing, same performance: Building a better bloom filter"; A. Kirsch, et al.; In Proc. the 14th Annual European Symposium on Algorithms (ESA 2006)