Bloom Filters for Estimating Set Differences

Carleton University

COMP 4905 – Honours Project

Student - William Dullemond

Supervisor - Patrick Morin

December 2018

Abstract

With the rise of data collection and decentralized systems the need to understand and synchronize data across multiple systems is more present then ever. The purpose of this paper is to show how Bloom Filters can be utilized for finding the difference between two remote sets with minimum communication. In systems with large sets of data or small constrained systems, set reconciliation can require a costly communication. Therefore an estimation of set differences can allow systems to make informed decisions about reconciliation. By using Bloom Filters communication would be lessened allowing for estimations at a reduced communication cost. This paper examines two approaches to estimating set differences. Method one is based on examining the bitwise subtraction of Bloom Filters to estimating the number of different elements. This Method is fast but is has large error bars. Method two is based on testing each element of one set against the Bloom Filter of another set. This method is slower and requires larger Bloom Filters but is much more reliable. Both methods provide a means of estimating set differences with different strengths and weaknesses.

Acknowledgments

Thanks to my Keurig Machine for always making my coffee

Thanks to my Coffee Cup for never failing to hold my coffee

Thanks to Austin Appleby for placing MurmurHash, his fast non-cryptographic hash function, into the public domain

Table of Contents

List of Figures

Motivation

With the large data set that are constantly being generated, data synchronizing is becoming even more important. The cost of communication can often be a bottleneck for many systems so minimizing and optimizing communication can increase the effectivity of many systems. Bloom Filters provide a method of reduced communication in exchange for small probabilities of inaccuracy. The use of Bloom Filters and variants in set reconciliation is well explored and studied in various papers. However these papers focus on the full or partial set reconciliation and not just estimating differences. Estimating set differences is a sub problem of Set Reconciliation however these papers use Bloom Filters for a more then just this sub problem. The goal of this paper is to focus on estimating set differences with Bloom Filters specifically, removed from the larger context of Set Reconciliation. This separation may yield interesting results as these Bloom Filters are not required to serve multiple purposes and therefore can be optimized.

Related Work

The estimating of set differences is an important action that warrants exploration. There already exist several methods for estimating set differences each with advantages and disadvantages. One method of estimating set difference is to simply check each element of a set against the elements of the other set. This is a terrible method as it requires high communication and computation but it does provide the highest level of accuracy. To combat having to communicate an entire set you can employ Random Sampling. By only comparing **K** randomly selected elements a decent approximation of the sets differences can be determined. The accuracy of Random Sampling suffers greatly for low **K** or for low set differences as the sampling has less chances to hit different elements. Another method is Min-wise sketch[1] which was created for finding set similarities but can be inverted to find differences. This method works by selecting ***K*** random hashes and running each element of a Set ***S*** throughout each of the ***K*** hashes. We then record the minimum value produced for each of the ***K*** hashes in a Min-wise sketch. By comparing the number of matching Elements(***M***) in each sketch the similarity can be estimated to be (***M/K***). For the purpose of set differences this can be re-expressed as ***D* = (K-M)/K.** This method, similar to random sampling, losses precision for low **K** and when the difference is small.

Use Cases

The uses for this paper is to examine the different ways in which Bloom Filters can be used to estimate set differences. Estimating set differences efficiently is useful first tool for complete set reconciliation. For large commercial databases such as Walmart and Amazon keeping exact items counts across multiple database is extremely difficult and often not necessary. By estimating these databases relative difference, companies can identify if any databases are outside the acceptable margins of error and take appropriate action. This quick identifying of outliers is also useful for warning sensor systems. If the data between individual sensors differs greatly then it is likely that an event has happened near that sensor. Finally, with the rise of both distributed and cluster systems, communication has become a large bottleneck. By estimating the difference between two sets these systems can make informed decision about when and how to route communication. This is especially useful in dynamic networks were more direct routes between system nodes can be created. There are plenty of important uses for estimating set differences.

[1] [Broder et al., 2000]

Bloom Filters Overview

Background and Objective

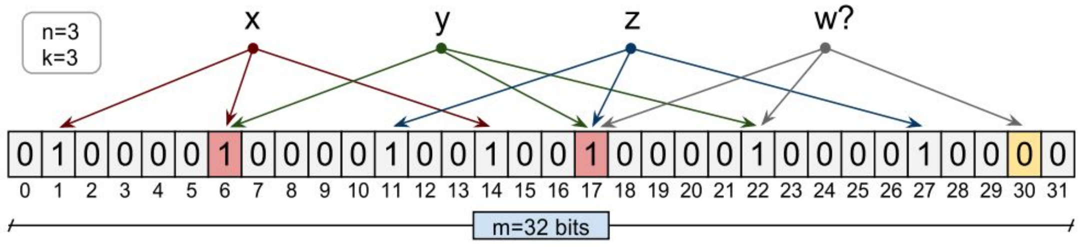
Bloom Filters were first designed by Bloom in 1970[2]. This data structure allows fast testing of set membership with small space and computational requirements. In exchange for this it there is a low probability of the filter returning a false positive for set membership but never a false negative. The probability of false positives are directly linked to the size of the filter and the number of elements in it. This allows the user to scale up or down depending on the systems requirements. A Counting Bloom Filter(*ref*) is the exact same design except that in exchange for more memory it counts the number of times an element has been inserted into the filter and can return the result for the user. This also has the side effect of allowing items to be removed from the filter which the standard filter does not support. The Bloom filter is the compromise between accuracy and storage space when testing set membership.

Algorithm's Description

The body of the Bloom Filter is bit string of length ***M*** where initially each bit is set to 0. To add an element to the BF you run the element through ***K*** preselected hashes. Each of these hashes are unique and return a number ***I*** that is between 0 and ***M***. Then for each hash ***K*** the bit ***I*** in the bitstring is set to 1. Testing set membership is done in the same way where membership is true if for each bit ***I*** is set to one and membership is false if any bit ***I*** is zero. (*Fig 1*) represents the method of addition of {x,y,z} for ***M*** = 32, ***K*** = 3 as well as a testing for membership of {w}.

2

(Fig 1) Bloom Filter example



Bloom Filters Math and Variables

M = Filter Length, N = Set Size, K = Number of Hashing Functions

Space Requirements = O(M)

Adding an Element = O(K)

Testing an Element = O(K)

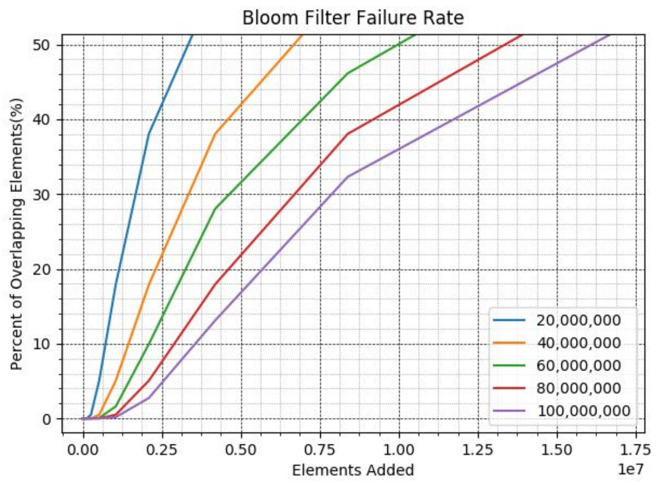
Probability of a False Positive = $\left(1-\left(1-\frac{1}{m}\right)^{kn}\right)^{k} \approx \left(1-e^{-kn/m}\right)^{k}$

Optimal K = $k_{opt} = \frac{m}{n}\ln 2$

(Fig.2)Bloom failure rate over elements

The probability of false positives increases as N increases(Fig 2.)

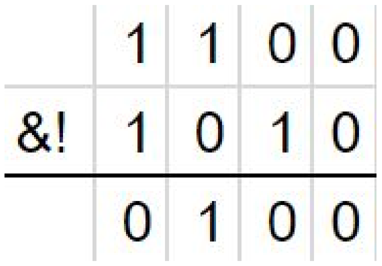
The probability of false positives decreases as M increases(Fig 2.)



Method One

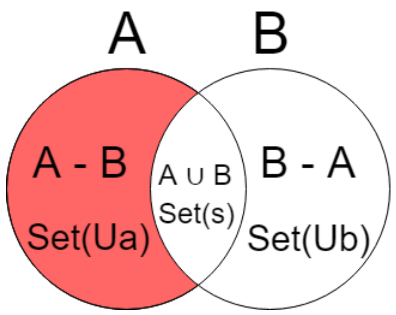
The objective of Method One is to find the size of the set difference between two sets **A** and **B** using only Bloom Filters. The first step for this method is to create a Bloom Filter for each set using the same values for **M** and **K**(see Bloom Filter). The values for **M** and **K** should be chosen with respect to the largest set. After the filters are constructed we have two bit strings(**bs)** of length **M** that represent the contents of the sets **A** and **B.** The second step is to create a **bs(C)** such that any bit **i** in **bs(C)** is only set if also set in **bs(A)** and not set in **bs(B)** in accordance with (fig.3). Although **bs(C)** is not the actual Bloom Filter of (**A-B**) the number of set bits is representative of the set(**A-B**) size.

(Fig.3) Bitwise Difference chart



	1	1	0	0
&!	1	0	1	0
	0	1	0	0

(Fig.3.5) Venn Diagram



To estimate the size of (**A-B**) we need to understand the probabilities involved with a bit being set in **bs(C).** For this we need to break up the sets **A** & **B** into smaller subset in as displayed in (fig.3.5)

$$Set(A) = Set(elements\ unique\ to\ A) + Set(elements\ shared\ with\ B)$$
$$Set(B) = Set(elements\ unique\ to\ B) + Set(elements\ shared\ with\ A)$$
$$Set(A) = Set(u_a) + Set(s) \qquad Set(B) = Set(u_b) + Set(s)$$
$$N_a = |Set(A)| \qquad N_b = |Set(B)|$$

Now that that the Sets **A** and **B** have been redefines to share a common subset **S** the difference between **A** and **B** is now captured in the set(**Ua**). Using this new expression we can examine the probability that any given bit exist in **bs(C).**

$$Pr(not\ set\ by\ S) * Pr(not\ set\ by\ U_b) * Pr(set\ by\ U_A) = Pr(only\ set\ byU_1)$$
$$(1-(1/m))^{k|s|} * (1-(1/m))^{k(n_b-|s|)} * (1-(1-(1/m)^{k|u_a|})) = x$$
$$((m-1)/m)^{k|s|} * ((m-1)/m)^{k(n_b-|s|)} * (1-((m-1)/m)^{k|u_a|}) = x$$
$$((m-1)/m)^{kn_b} * (1-((m-1)/m)^{k|u_a|}) = x$$
$$u = (log(((m-1)/m)^{(k*n)} - x) - k*n*log((m-1)/m))/(k*log((m-1)/m))$$

Since **M, Nb,** and **K** are defined the only undefined variable is **x** which we define as the probability that any given bit is set in **bs**(**C**)**.** This means that **x** is approximately equal to the number of set bits in **bs**(**C**) divided by the total number of bits **M**. This provides us with all the variables required to estimate the size of **U.**

Run Time

1. Generate **bs(C)** from **bc(A)** and **bc(B)** = O(**m**)
2. Count set bits in **bs(C)** = O(**m**)
3. Calculate **|Ua|** = O(1)

Runtime = O(**m**)

Storage = O(**m**)

Method Two

The objective of Method two is to find the difference between two sets with one Bloom filter. This is simply done by testing all elements in one set against the Bloom Filter other. Step one is to create a Bloom Filter from items in set(**B**). This filter can then be communicated to the system that contains the elements of set(**A**). Then each element can be tested against BF**(B)** and all the element that test negative(see Bloom Filter) belong to the set(**A-B**). Because of the nature of testing membership with Bloom Filters, the number of elements that fail the membership test function as a lower bound of the size of (**A-B**).

Runtime

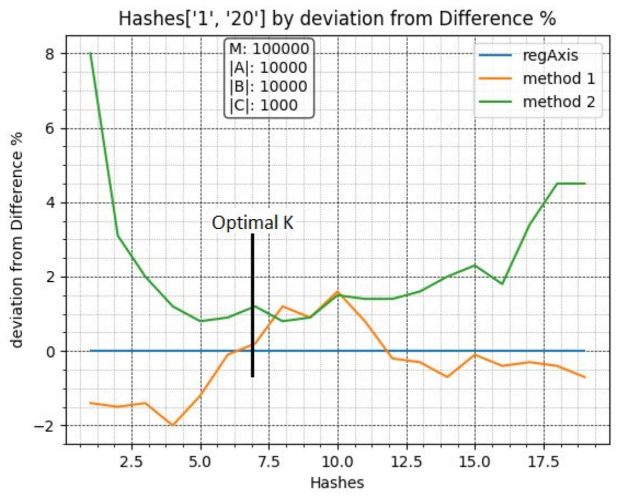
Communication = O(m)

Storage = O(m + n)

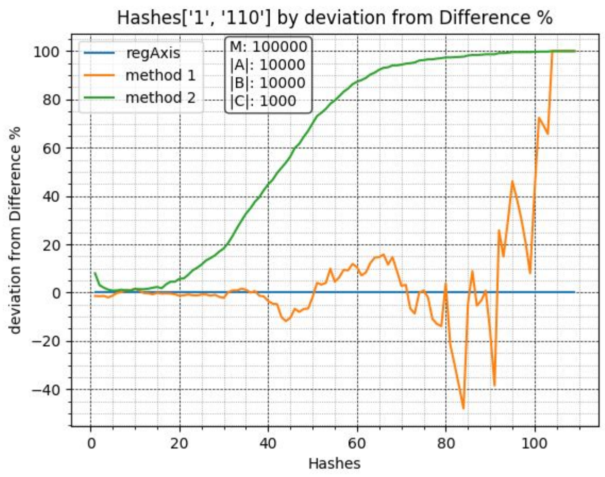
Runtime = O(N)

Results

(Fig.4) Hashes and Accuracy for small **K**



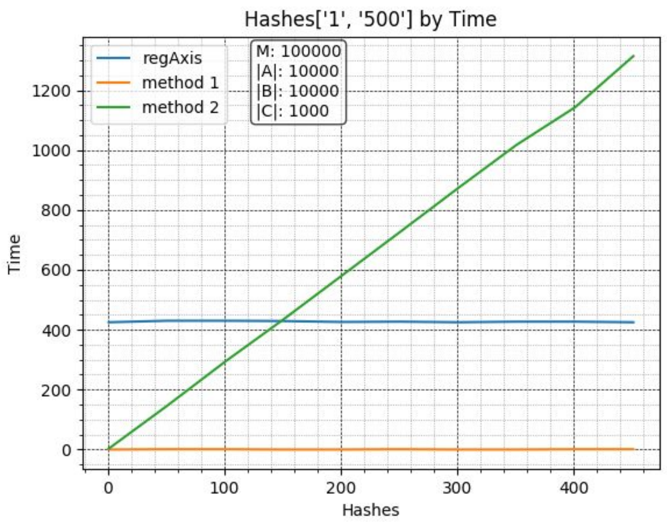
(Fig.5) Hashes and Accuracy for large **K**



Method One accuracy has clear ties to the size of **K**. As **K** increases the number of bits set increases which leads to a loss of accuracy as multiple items share bit in the Bloom Filter. This is also true for Method Two though this method follows a much more predictable pattern. Starting high and dipping down at for optimal **K** and steadily rising before failing at **K**=90.

Both Methods function best at optimal **K**(see Bloom Filter). Method one provides better accuracy for higher **K** but Method two has higher reliability.

(Fig.6) Hashes and Time for **K**

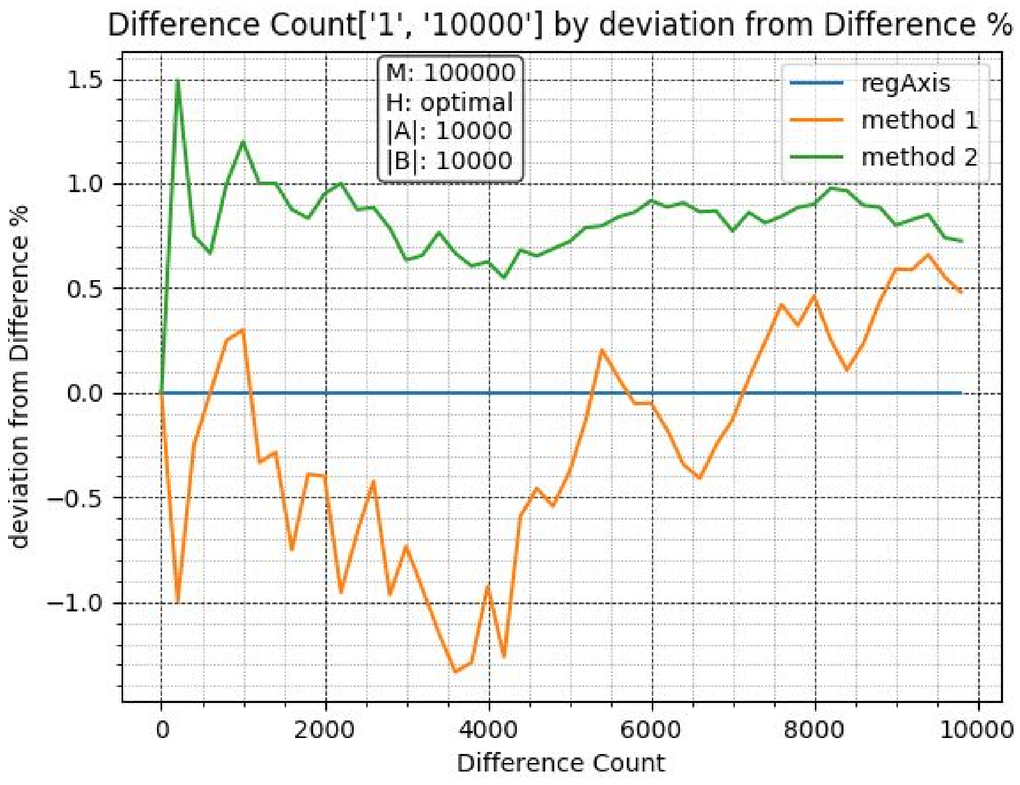


Method One’s time remains constant as none of it's computations rely of the number of Hashes.

Method Two’s time increases linearly with number of hashes. This is because when checking each element the work is O(**K**).

Method one is clearly faster

(Fig.7) Set difference and Accuracy

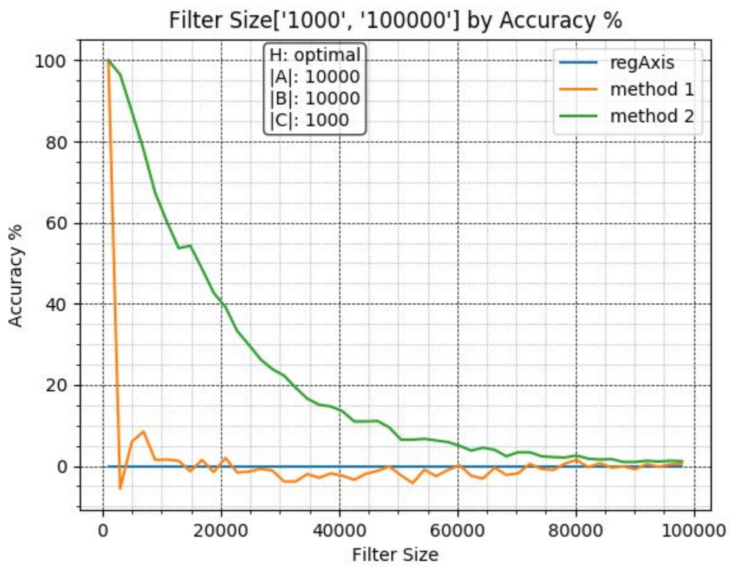


Method One the maximum level of error is 1.4% from the correct difference. There appears to be a slight increase in accuracy when the set differences increase.

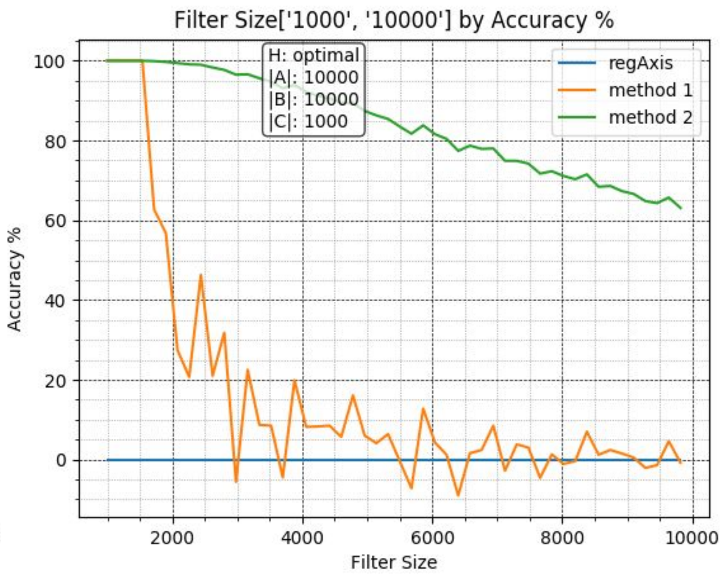
Method Two the maximum level of error is 1.5%. All estimations are below the actual differences allowing it to function as a lower bound. There appears to be no correlation between the percent of different elements and accuracy.

Both Methods do not substantially degrade based on how different the sets are.

(Fig.8) Filter Size and Accuracy large **M**



(Fig.9) Filter Size and Accuracy small **M**

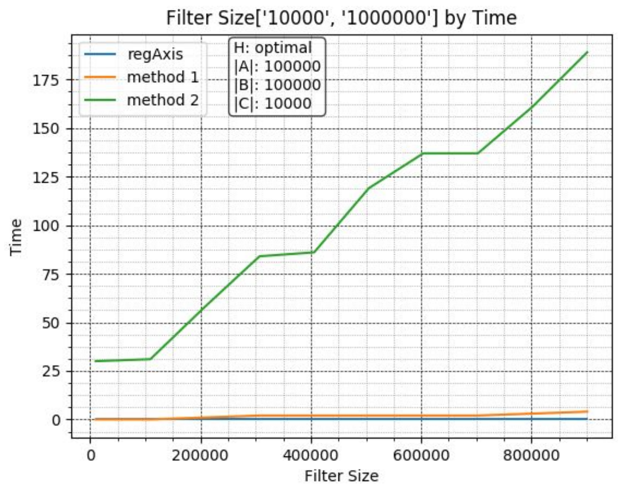


Method One's Accuracy is linked with the size of the filter(**M**). The Accuracy quickly increases hitting 20% with 4000 and 10% at 8000. After this is slowly increases as the filters size increases.

Method Two's Accuracy of the estimations are terrible for low filter size(**M**). The descent is fairly predictable only dropping below 10% after 50000. From here accuracy continues to slightly increase.

Method One has much higher accuracy for low values of **M** but Method two has higher reliability.

(Fig.10) Filter Size(**M**) and Time

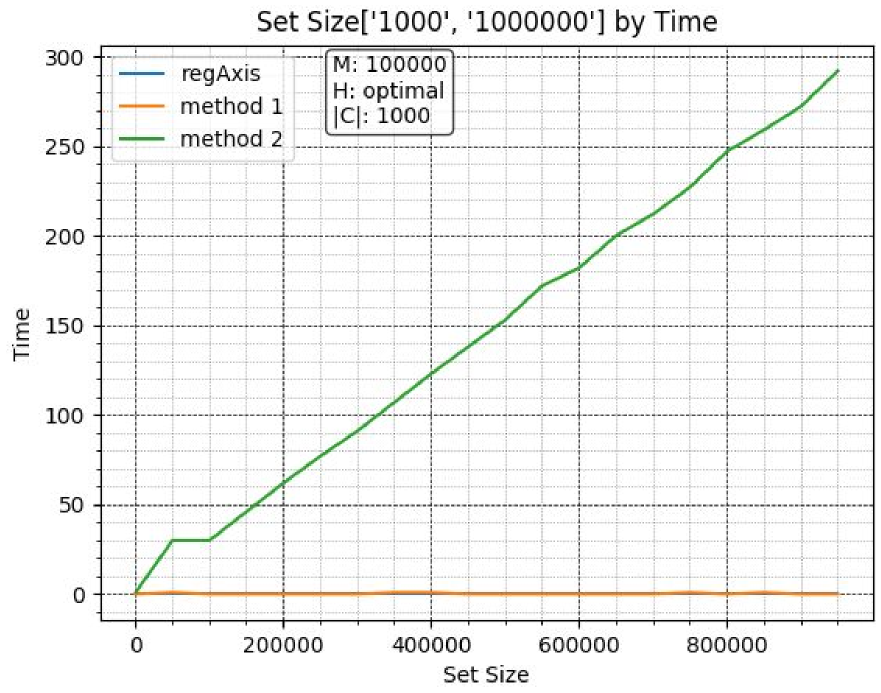


Method One's time slightly increases with the filter size but is still very low.

Method Two's time increases with filter size despite its runtime not being tied to **M.** This is likely because of more memory access requiring pieces of filter to be loaded in an out of the cache.

Method One is much faster

(Fig.11) Filter Size(**M**) and Time



Method One’s time is not linked to the size of the sets **A** and **B**

Method Two’s time is linked with the sizes of **A** and **B** in a linear nature. This is the result of having to check each element in **A** against the Bloom Filter of **B**

Method One is Clearly Faster

Conclusions

In conclusion, Method One is superior to Method Two in almost every way. Method one can provide higher levels of accuracy from smaller filter sizes(**M**). This means that Method One allows for smaller communication costs while maintaining a level of accuracy. Method One is also much faster requiring less computation which is only tied to the filter size(**M**) which can be smaller as per (Fig.9). The only advantage that Method Two held was its reliability. Where Method one was subject to large fluctuations of accuracy Method Two produced reliable if inaccurate results. This leads me to believe that the inaccuracy for small **M** could be offset by multiplying the results by some precalculated number similar to Method One. However this seems counterproductive as it would strip it of its reliability but remain more expensive computationally then Method one. Method one is faster and smaller than Method Two.

References

Agarwal, S., Trachtenberg, A. (2006). Approximating the number of differences between remote sets. Paper presented at the 217-221. doi:10.1109/ITW.2006.1633815

Bloom, B. (1970). Space/time trade-offs in hash coding with allowable errors. Commun. ACM 13, 7 (July 1970), 422-426.

Broder, A. Z., Charikar, M., Frieze, A. M., Mitzenmacher, M. (2000). Min-wise independent permutations. Journal of Computer and System Sciences, 60,pp. 630-659.

Chen, D., Konrad, C., Yi, K., Yu,W., Zhang, Qin. (2014) Robust Set Reconciliation, Hong Kong University of Science and Technology.

Eppstein, D., Goodrich, M. T., Uyeda, F., Varghese. G. (2011). What's the difference?: efficient set reconciliation without prior context. SIGCOMM Comput. Commun. Rev. 41, 4 (August 2011), pp. 218-229.

Gentili, Marco. (2015) Set Reconciliation and File Synchronization Using Invertible Bloom Lookup Tables. Bachelor's thesis, Harvard College.

Guo, D., Li, M. (2013). Set reconciliation via counting bloom filters. IEEE Transactions on Knowledge and Data Engineering, 25(10),pp. 2367-2380.

Tarkoma, S., Rothenberg, C. E., Lagerspetz, E. (2012). Theory and practice of bloom filters for distributed systems. IEEE Communications Surveys & Tutorials, pp. 131-155.