© February 2016 | IJIRT | Volume 2 Issue 9 | ISSN: 2349-6002

A Different approach of Bloom Filters for Error Detection and Correction

B. Ramesh [1], G. Kishore Kumar [2]

[1,2]*Assistant Professor©, Dept.of ECE, University College of Engineering (A), Osmania University, Telangana,*

Abstract—**A Bloom Filter (BF) is a data structure compatible for performing set membership queries very effectively. A standard Bloom Filter representing a set of n elements is generated by using an array of m bits and uses k unbiased hash functions. Bloom Filters have some attractive properties together with low storage requirement, fast membership checking and no false negatives. False positives are viable however their probability is also managed and significantly lowered depending upon the application standards. Our main contributions are exploring the design space and the evaluation of a series of extensions (1) to increase the practicality and performance of iBFs, (2) to enable false-negative-free element deletion, and (3) to provide security enhancements.. The proposed scheme may also be of interest in useful designs to without difficulty mitigate mistakes with a lowered overhead in terms of circuit area and power.**

Index Terms—**Bloom filters (BFs), error correction, soft errors.**

I. INTRODUCTION

Recent advances in next-generation sequencing (NGS) technologies have made it possible to rapidly generate high-throughput data at a much lower cost than traditional Sanger sequencing technology. NGS technologies enable cost-efficient genomic applications, including de novo assembly of many non-model organisms, identifying functional elements in genomes, and finding variations within a population. A Bloom Filter is an area effective probabilistic data constitution which is used to symbolize a collection and participate in membership queries [1] i.e. To query whether or not an element is a member of the set or now not. The Bloom Filter data structure used to be offered through Burton H. Bloom [2] in 1970. A Bloom Filter occupies negligible space in comparison with the entire set. Space saving comes on the cost of false positives however this difficulty does not affect the processing of data if the chance of an error is made sufficiently low. Bloom Filters normally find applications in instances that involve making a choice on membership of an aspect for a sufficiently huge set in small period of time. Today, Bloom Filters are utilized in vast variety of applications together with spell checking, network traffic routing and monitoring, database search, differential file updating, allotted community caches, and textual analysis. In this paper we will be able to describe bloom filter, its editions and its functions in unique areas of computer science.

II. LITERATURE REVIEW

In this work, we focus on the subset of distributed networking applications that use packet-header-size Bloom filters to share some state (i.e. information set S) among network nodes. The specific state carried in the Bloom filter varies from application to application, ranging from secure credentials to IP prefixes and link identifiers, with the shared requirement of a fixed-size packet header data structure to efficiently verify set memberships. The commonality of recent inter-networking proposals is relying on Bloom filters to move application state to the packets themselves in order to alleviate system bottlenecks (e.g. IP multicast, source routing overhead), enable new in-network applications (e.g. security) or stateless protocol designs.

We refer to the BF used in this type of applications as an in-packet Bloom filter (iBF). In a way, an iBF follows a reverse approach compared to a traditional standalone BF implementation: iBFs can be issued, queried, and modified by multiple network entities at packet processing time. These specific needs benefit from additional capabilities like element removals or security enhancements. Moreover, careful design considerations are required to deal with the potential effects of false positives, as every packet header bit counts and the actual performance of the distributed system is a key goal In this article, we present a new Bloom filter-based error correction algorithm, known as BLESS. BLESS belongs to the k-mer spectrum-based method, but it is designed to cast off the aforementioned limitations that previous k-mer spectrum-situated options had. Our new procedure has three main new points:

(1) BLESS is designed to target high memory efficiency for error correction to be run on a commodity laptop. The k-mers that exist more than a specific number of occasions in reads are sorted out and programmed into a Bloom filter.

(2) BLESS can handle repeats in genomes higher than earlier k-mer spectrum-based methods, which results in better accuracy. This is since BLESS is in a position to make use of longer k-mers in comparison with prior methods. Longer k-mers untangle repeats better.

(3) BLESS can extend reads to proper mistakes at the finish of reads as thoroughly as other constituents of the reads. Usually an inaccurate k-mer is also recognized as an error-free one due to the fact that of an irregularly tremendous multiplicity of the k-mer.

False positives from the Bloom filter may additionally rationale the same difficulty. BLESS extends the reads to search out multiple k-mers that cover the inaccurate bases on the end of the reads to give a boost to error correction on the finish of the reads.

In this section we explore and describe variants of Bloom Filter [5] built on the Standard Bloom Filter data structure.

The Standard Bloom Filter works fine when the members of the set do not change over time. Addition of elements only requires hashing the additional item and setting the corresponding bit locations in the array. However, deletion is not possible in the Standard Bloom Filter since it will require setting 0's in the array to already set 1's that was result of hashing another item which is still a member of the set.

The Variable Increment Counting Bloom Filter (VI – Bloom) [7] is a generalization of the Counting Bloom Filter that uses variable increments to update each entry. In this structure, a set of possible variable increments are defined. For each counter update by an element we hash the element into the variable increment set and use it to increment the counter. Similarly, to delete an element we decrement by its hashed value in the variable increment set.

A Scalable Bloom Filters consist of two or more Standard Bloom Filters, allowing arbitrary growth of the set being represented. When one Bloom Filter gets filled due to the limit on the fill ratio, a new filter is added. Querying an element involves testing the presence in each filter.

III. METHODOLOGY

The proposed scheme is based on the statement that a CBF, additionally to a structure that permits rapid membership check to an element set, can also be in a technique a redundant illustration of the element set. Therefore, this redundancy might in all probability be used for error detection and correction. To discover this concept, common implementations of CBFs where the elements of the set are saved in a sluggish memory and the CBF is saved in turbo memories are regarded. In specified, it's assumed that the elements of the set are saved in DRAM whilst the CBF is saved in a cache [10]. The reasoning behind that is that the CBF is accessed typically and wants a quick access time to maximize efficiency, at the same time the elements of the set are only accessed when factors are learn, added or eliminated and therefore the entry time is no longer an obstacle. It should also be noted that after the whole aspect set is stored in a slow memory, no unsuitable deletions can arise as they could be detected when taking away the element from the slow memory.

A. Simple Procedure for the Correction of Errors in the Element Set:

To propose the simple correction system, allow us to count on that a single bit error impacts detail x and that it's detected using the parity bit. Hence, x_e is read from the reminiscence. The right worth x has to be x_e if the error affected the parity bit. If the error affected the i^{th} data bit, the correct value can be $x_{em}(i)$ the place $x_{em}(i)$ is the value read (x_e) with the i^{th} bit inverted. To assess which of these is actually the right value x, the candidates [x_e and all the $x_{em}(i)$] can be verified for membership to the CBF. If most effective one of the candidates is located within the CBF, then no false positives have come about and the value observed is the correct one. Rather, if more than one candidate is observed, the approach is unable to seek out the right worth due to the occurrence of false positives. This straightforward and quick method requires handiest l + 1 queries to the CBF, where l is the quantity of bits in each aspect of the set. However, the correction expense that can be carried out is dependent upon the false positive price of the CBF. In detailed, the chance that an error can be corrected using this procedure can be approximated as

$$P_{correction} \cong (1 - p_{fp})^l$$

which is the probability that none of the l candidates that are not x return a false positive on a query.

B. Advanced Procedure for the Correction of Errors in the Element Set

When the simple system described above correct error, a supplementary developed method can be utilized. The correction approach starts by means of making a replica of the CBF in DRAM memory. Then, all of the factors within the set besides for the erroneous one are removed from the CBF. This will go away a CBF with simplest the values that correspond to the original worth of the element x. Once that's executed, the candidates [x_e and all the x_{em} (i)] may also be queried over the CBF that has best x as an entry. As in the prior approach, if most effective probably the most candidates suit the CBF, that is the correct value. If more than one candidate suits the CBF then the error can't be corrected. The likelihood that a given price x and yet another price y produce precisely the same values of the hash capabilities h_1, h_2... h_k can also be approximated as

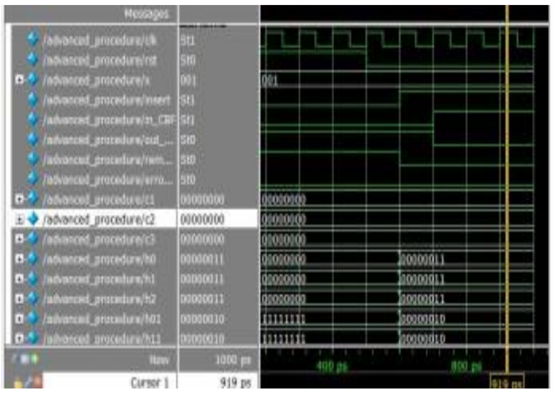
$$P_{CBF(x)} = CBF(y) \cong \frac{k!}{m^k}$$

Therefore, the correction probability for this advanced procedure can be approximated as

$$P_{correction} \cong (1 - \frac{k!}{m^k})$$

So that they can be very just about one hundred% in lots of sensible eventualities as m is in general huge. The accelerated correction rate comes on the cost of an additional complex correction method that wants the replication of the CBF, the removal of the entire entries besides the erroneous one (n−1), and sooner or later the query for the l + 1 candidates. However, as soft error are infrequent events, and the process is only wanted when the straightforward approach provided earlier than are not able to correct an error, the scheme may also be valuable in real purposes.

IV. RESULTS AND DISCUSSIONS



```
Messages
/advanced_procedure/clk     St1
/advanced_procedure/rst     St0
/advanced_procedure/x       001
/advanced_procedure/insert  St1
/advanced_procedure/in_CBF  St1
/advanced_procedure/out_... St0
/advanced_procedure/rem...  St0
/advanced_procedure/erro... St0
/advanced_procedure/c1      00000000
/advanced_procedure/c2      00000000
/advanced_procedure/c3      00000000
/advanced_procedure/h0      00000011
/advanced_procedure/h1      00000011
/advanced_procedure/h2      00000011
/advanced_procedure/h01     00000010
/advanced_procedure/h11     00000010
Now                         1000 ps
Cursor 1                    919 ps
```

Fig.1 Simulation output

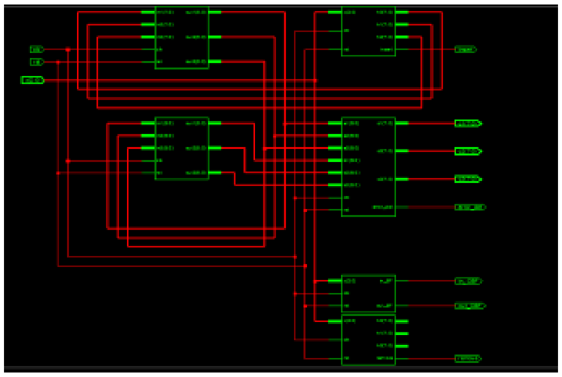


Fig.2 RTL Schematic.

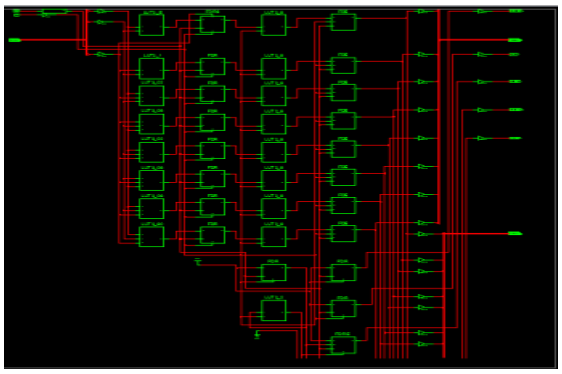


Fig.3 Technology Schematic.

V. CONCLUSION

This paper explores an exciting front in the Bloom filter research space, namely the special category of small Bloom filters carried in packet headers. The configuration regarded on this temporary is that of a memory included with a per phrase parity bit for which it is confirmed that the CBF can be utilized to gain single bit error correction. This suggests how present CBFs can be used to gain error correction in addition to perform their normal membership checking function.

REFERENCES

[1] Peter Brass, Advanced Data Structures, Cambridge University Press, 2008, pp. 402-405.

[2] Burton H. Bloom, Space/time trade-offs in Hash Coding with Allowable Errors, Communications of the ACM, Volume 13, Issue 7, 1970.

[3] A. Moshovos, G. Memik, B. Falsafi, and A. Choudhary, "Jetty: Filtering snoops for reduced energy consumption in SMP servers," in Proc. Annu. Int. Conf. High-Perform. Comput. Archit., Feb. 2001, pp. 85–96.

[4] C. Fay et al., "Bigtable: A distributed storage system for structured data," ACM TOCS, vol. 26, no. 2, pp. 1–4, 2008.

[5] F. Bonomi, M. Mitzenmacher, R. Panigrahy, S. Singh, and G. Varghese, "An improved construction for counting bloom filters," in Proc. 14th Annu. ESA, 2006, pp. 1–12.

[6] M. Mitzenmacher, "Compressed bloom filters," in Proc. 12th Annu. ACM Symp. PODC, 2001, pp. 144–150.

[7] M. Mitzenmacher and G. Varghese, "Biff (Bloom Filter) codes: Fast error correction for large data sets," in Proc. IEEE ISIT, Jun. 2012, pp. 1–32.

[8] S. Elham, A. Moshovos, and A. Veneris, "L-CBF: A low-power, fast counting Bloom filter architecture," IEEE Trans. Very Large Scale Integr. (VLSI) Syst., vol. 16, no. 6, pp. 628–638, Jun. 2008.

[9] T. Kocak and I. Kaya, "Low-power bloom filter architecture for deep packet inspection," IEEE Commun. Lett., vol. 10, no. 3, pp. 210–212, Mar. 2006.

[10] S. Dharmapurikar, H. Song, J. Turner, and J. W. Lockwood, "Fast hash table lookup using extended bloom filter: An aid to network processing," in Proc. ACM/SIGCOMM, 2005, pp. 181–192.

[11] Almeida, Paulo; Baquero, Carlos; Preguica, Nuno; Hutchison, David (2007), Scalable Bloom Filters, Information Processing Letters, pp.255–261.

[12] Rafael Laufer, Pedro B. Velloso, and Otto Carlos M. B. Duarte, A Generalized Bloom Filter to Secure Distributed Network Applications, Computer Networks, vol. 55, no. 8, pp. 1804-1819, June 2011.

[13] Chazelle, Bernard; Kilian, Joe; Rubinfeld, Ronitt; Tal, Ayellet, The Bloomier Filter: an Efficient Data Structure for Static Support Lookup Tables, Proceedings of the Fifteenth Annual ACM-SIAM Symposium on Discrete Algorithms, pp. 30–39, 2004.

[14] Deng, Fan; Rafiei, Davood , Approximately Detecting Duplicates for Streaming Data using Stable Bloom Filters, Proceedings of the ACM SIGMOD Conference, pp. 25–36, 2006.

[15] James K. Mullin and Daniel J. Margoliash, A tale of three spelling checkers, Software, Practice and Experience, pp. 625- 630, June 1990