DBA: A Dynamic Bloom Filter Array for Scalable Membership Representation of Variable Large Data Sets

Jiansheng Wei
Wuhan National Lab for Optoelectronics
School of Computer Science and Technology, Huazhong University of Science and Technology
Wuhan, China
jianshengwei@gmail.com

Hong Jiang
Dept. of Computer Science and Engineering
University of Nebraska-Lincoln
Lincoln, NE, USA
jiang@cse.unl.edu

Ke Zhou✉, Dan Feng
Wuhan National Lab for Optoelectronics
School of Computer Science and Technology, Huazhong University of Science and Technology
Wuhan, China
{k.zhou, dfeng}@hust.edu.cn

***Abstract*—This paper proposes a Dynamic Bloom filter Array (DBA) to represent membership for variable large data sets in storage systems in a scalable way. DBA consists of dynamically created groups of space-efficient Bloom Filters (BFs) to accommodate changes in set sizes. In each group, BFs are homogeneous and the data layout is optimized at the bit level, so that they can be accessed in parallel to achieve high query performance. DBA can effectively control its query accuracy by partially adjusting the error rate of constructing BFs, where each BF corresponds to an independent subset of the data set to facilitate element location and membership confirmation. Further, DBA supports element deletion by introducing a lazy update policy. We prototype and evaluate our DBA scheme as a scalable fast index in the MAD2 deduplication storage system. Experimental results show that DBA (with 64 BFs per group) is capable of maintaining 90% of the peek query performance while scaling up to 160 BFs. DBA is also shown to excel in performance and space efficiency by theoretical analysis and other experiments based on real-world data sets.**

Keywords-Data management; membership representation; fast index; Bloom filter

I. INTRODUCTION

Membership representation and determination is important for storage systems that manage very large data sets. For example, a key-value store needs to efficiently determine *whether a required data item exists* without having to exhaustively check all the elements. A straightforward approach that keeps an ordered full index in memory will face the following challenges when dealing with variable large data sets. First, it is cost-ineffective to maintain an ordered full index, as the logical/physical structure of the index must be frequently adjusted to accommodate the addition or deletion of elements. Commercial stores such as Microsoft's ChunkStash [1] allow complicated keys (i.e., SHA-1 hashes) that cannot be efficiently sorted. Second, as the amount of data grows, the whole index can become too large to be stored in the RAM in its entirety, as discussed in DDFS [2], causing disk access bottleneck.

In general, we argue that indexing dynamically variable large data sets in storage systems requires the membership representation scheme to achieve (1) strong scalability to adapt to the ever increasing amount of data, (2) high space efficiency to be contained in RAM for high access speed, (3) high query performance to resist the rising search complexity during the scaling-up process, (4) controllable query accuracy to resist errors introduced by compact data structures, (5) element locating capability to inform the system where to find the possible elements, and (6) element deletion capability to allow the storage system to recycle unused spaces.

Previous techniques such as *compact hash tables* [1] and Bloom filters (BFs) [3] are suitable only for fixed-cardinality data sets. To the best of our knowledge, existing scalable membership representation schemes, such as *dynamic Bloom filters* (DBF) [4], *scalable Bloom filters* [5] [6], and *incremental Bloom filters* [7], mainly focus on network applications and lack a comprehensive consideration about the above requirements of storage systems.

We propose DBA, a Dynamic Bloom filter Array aimed at representing membership for variable large data sets in storage systems in a scalable way. DBA consists of groups of space-efficient Bloom filters (BFs), and its capacity can be extended by dynamically adding new BFs. Within a group, homogeneous BFs are used and their structure is optimized at the bit level, so that dozens of BFs can be accessed in parallel to achieve high query performance. Using the compact BF structure introduces false positives, but the query accuracy can be effectively controlled by partially adjusting the error rate of the building-block BFs. Each BF is only responsible for representing an independent subset, which helps locate existing elements. Further, DBA supports element deletion by introducing a lazy update policy.

We evaluate our DBA scheme by using it as a fast index in the MAD2 [8] deduplication storage system, and it has been shown to significantly outperform existing approaches and thus be a practical scalable fast index in storage systems.

II. THE PRINCIPLES OF BLOOM FILTERS

A Bloom filter (BF) [3] [9] for representing a static set S of n elements consists of an array of m bits and a set of k independent hash functions. The bits $h_i(x)\in\{1, \ldots, m\}$ $(1\le i\le k)$ will be set to 1 for each element $x\in S$. While querying whether an item y belongs to S, there is a probability f_{BF} that y is falsely identified as a member of the set, which is called the false positive rate or error rate. Given n, if f_{BF} must be restricted to a threshold ε with minimal space overhead, the optimal number of hash functions should be $k_{opt}=\log_2(1/\varepsilon)$, and the required minimal space (in bits) is $m_{min}=\log_2 e\cdot k_{opt}\cdot n=\log_2 e\cdot\log_2(1/\varepsilon)\cdot n$. In this context, n and ε are also referred to as the designed capacity c_{BF} and the target error rate F_{BF}, respectively.

1526-7539/11 $26.00 © 2011 IEEE
DOI 10.1109/MASCOTS.2011.36

III. THE DBA SCHEME

This section presents the design of Dynamic Bloom filter Array (DBA) to carry out a practical membership representation scheme for variable large data sets in storage systems. Figure 1 shows a general framework of using DBA as a scalable fast index in a storage system. A DBA that maintains high space efficiency can be fully loaded in RAM to improve the access efficiency. On receiving query requests, the DBA scheme will be first checked to quickly filter out nonexistent items. For possible existing elements, the DBA scheme directs the query process to search the most relevant subsets to confirm membership through an in-memory cache that maps to the on-disk full index. If a fresh item arrives, its metadata will be inserted into both the DBA and the full index, and the data content will be written to the on-disk data store accordingly through an in-memory data cache. As more fresh data is written, DBA scales along with the on-disk full index, and only the updated components (i.e., BFs) of the DBA need to be synchronized to the on-disk copy. Furthermore, the DBA scheme can also be used as a global fast index [10] in a distributed storage system.

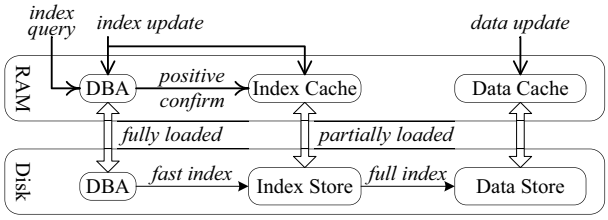

Figure 1. Using DBA as a scalable fast index.

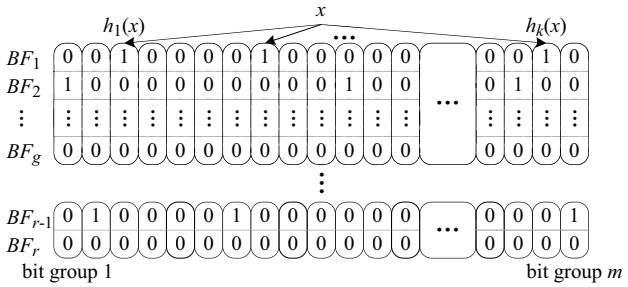

Figure 2. The bit layout of a DBA. A DBA stores the bits in a column-major order to provide parallelism in memory access, in clear contrast to a *dynamic Bloom filter* (DBF) [4] that stores the bits in a row-major order.

A DBA is constructed of groups of compact BFs for space efficiency. BFs are homogeneous within each group to share the same hash functions and reduce the computational overhead. Existing fully-filled BFs will be sealed for query only and new BF can be created dynamically to expand the capacity. Specifically, bits belonging to g different BFs but with the same offset are mapped and stored together so that they can be simultaneously accessed, as shown in Figure 2. It can be inferred that the memory-access complexity for querying an item is $O(k\times r/(2g))$ for a positive result and $O(k\times r/g)$ for a negative result, where r is the total number of BFs. The number g of BFs in a group is called group cardinality, which determines the parallelism in querying multiple BFs. In practice, g is recommended to be a fraction of r for the consideration of controlling query accuracy.

Considering a DBA that is initially designed to hold r BFs with each has a capacity of c_{BF}, the designed capacity of the DBA is $c_{DBA}=c_{BF}\times r$. Let ε_{DBA} denote the error rate threshold, and F_{BF} the target error rate of each BF, we have $1-(1-F_{BF})^r=\varepsilon_{DBA}$. The problem here is that the actual error rate can exceed ε_{DBA} if the number of elements exceeds c_{DBA}. There are two methods to control the error rate. The first is to enlarge the capacity of the BFs while keeping both r and F_{BF} constant. The second is to increase r by suppressing F_{BF}. Both methods can be implemented by rebuilding BFs in groups in the background, so that BFs in the same group can remain homogeneous and be queried in parallel.

Taking advantage of DBA's scalability, we propose to use different BFs to represent independent subsets, so that a possible existing element can be located and confirmed by simply searching the associated subset of a BF that generates a positive for it. Considering a DBA containing r BFs, the search range can be directly reduced to $1/r$ if only one of the BFs generates positive. The probability of multiple positives is relatively small and will not cause significant overhead.

Considering the fact that large storage systems usually delete stale data in batches in off-peak time, we introduce a lazy update policy for the DBA scheme to support element deletion. A DBA associates each BF with two counters that record the number of represented elements and the number of stale elements respectively. Only when the stale elements in a BF reach a predefined proportion, will a BF be refreshed to reflect the real-time membership of the subset.

IV. EVALUATIONS AND DISCUSSIONS

The DBA scheme is implemented as a fast index in a chunk server of the MAD2 [8] deduplication storage system and evaluated with 2,026,005,927 chunk fingerprints that can be further deduplicated into 83,733,597 unique items.

We construct five different configurations of DBA (i.e., 1-, 8-, 16-, 32-, and 64-group-cardinality DBA) and measure their query performance respectively. As shown in Figure 3, a DBA with greater group cardinality performs worse due to an increased management overhead at the beginning. When the number of BFs grows to 16, the advantage of parallel query takes effect, and the multi-group-cardinality DBA starts to outperform the 1-group-cardinality DBA, which is functionally equivalent to a DBF. As the fingerprint set further scales up and more BFs are created, a DBA with greater group cardinality is shown to better maintain its performance. When the number of BFs reaches 160, the 64-group-cardinality DBA maintains a query performance of 4.5×10^5 fingerprints per second (fps), in clear contrast to the 1.5×10^5 fps performance of the DBF.

According to the results in Figure 3, we can expect that a high-group-cardinality DBA will be far more scalable in capacity than a DBF while maintaining its query performance at or above the predefined threshold.

Considering a variable large data set S with a maximum cardinality c_{max}, the number of bits required by a DBA consisting of r homogeneous BFs can be derived as $m_{DBA}=\log_2 e\cdot\log_2(1/\varepsilon_{BF})\cdot c_{max}$, where $\varepsilon_{BF}=1-(1-\varepsilon_{DBA})^{1/r}$ is the error rate threshold of each member BF in the DBA. In general, a DBA inherits the high space-efficiency of a BF.

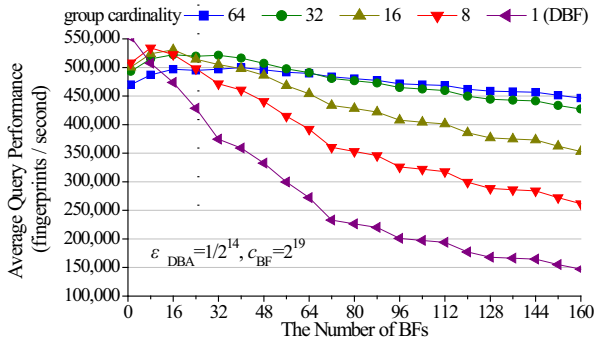

Figure 3. Average query performance of a DBA under 5 different group cardinalities. For the stability of results, the results are measured in statistics windows near the sample points.

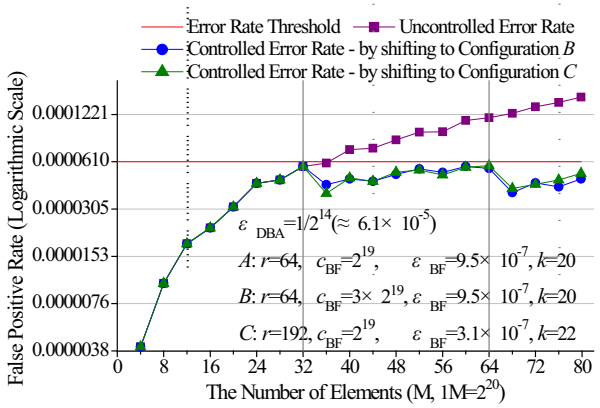

Figure 4. Error rate control capability of a DBA. The real error rate is measured by counting the false positives in statistics windows near the sample points.

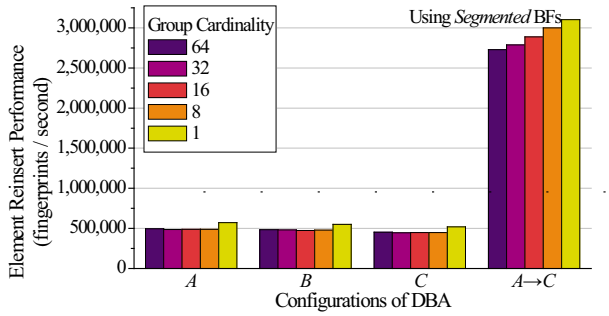

Figure 5. Partial reconstruction efficiency of a DBA.

Suppose that the maximum cardinality of the fingerprint set is unknown initially, we start with a DBA designed with Configuration *A* to measure its error rate, as shown in Figure 4. Without controlling the error rate, 160 BFs are created to hold all the fingerprints, and the error rate eventually grows to 1.6×10^{-4} that is about 2.6 times larger than the threshold. On the other hand, by partially rebuilding groups of BFs using Configuration *B* and Configuration *C* respectively, the error rate can be effectively controlled below the threshold.

Further, we measure the element reinsert performance of the DBA under all the configurations to evaluate its efficiency in supporting error rate control and element deletion, and the results are shown in Figure 5. In particular, if the DBA is implemented with *segmented* BFs [9], shifting it from Configuration *A* to Configuration *C* only needs to add two more segments along with the corresponding hash functions for each BF, and the results show a significantly higher performance.

V. Conclusions

This paper presents the DBA scheme to represent membership for variable large data sets in storage systems. Experimental results show that a DBA can maintain high query performance while using large group cardinality and the query accuracy can be effectively controlled by either enlarging the capacity or tightening the target error rate of its member BFs. Moreover, the element reinsert performance is high enough to support error rate adjustment and reflect membership update. Theoretical analysis reveals that a DBA inherits the high space-efficiency of a BF while achieving high flexibility, scalability and performance that the latter lacks. In general, a DBA outperforms existing schemes and is more applicable to representing variable large data sets as a scalable fast index in storage systems.

Acknowledgment

This work is supported in part by the National Basic Research Program (973 Program) of China under Grant No.2011CB302305, the National High Technology Research and Development Program (863 Program) of China under Grant No.2009AA01A402, and the US NSF under Grants NSF-IIS-0916859, NSF-CCF-0937993 and NSF-CNS-1016609. The authors are grateful to the anonymous reviewers for their valuable comments and suggestions.

References

[1] B. Debnath, S. Sengupta, and J. Li, "ChunkStash: Speeding up Inline Storage Deduplication using Flash Memory," *Proc. USENIX Annual Technical Conference (ATC)*, Boston, MA, USA, June 2010.

[2] B. Zhu, K. Li, and H. Patterson, "Avoiding the Disk Bottleneck in the Data Domain Deduplication File System," *Proc. 6th USENIX FAST*, pp. 269-282, San Jose, CA, USA, Feb. 2008.

[3] B. H. Bloom, "Space/Time Trade-offs in Hash Coding with Allowable Errors," *Communications of the ACM*, vol. 13, no. 7, pp. 422–426, July 1970.

[4] D. Guo, J. Wu, H. Chen, and X. Luo, "Theory and network applications of dynamic bloom filters," *Proc. 25th IEEE INFOCOM*, pp. 1–12, Barcelona, Spain, Apr. 2006.

[5] K. Xie, Y. Min, D. Zhang, J. Wen, and G. Xie, "A Scalable Bloom Filter for Membership Queries," *Proc. 50th IEEE GLOBECOM*, pp. 543–547, Washington, DC, USA, Nov. 2007.

[6] P. S. Almeida, C. Baquero, N. Preguica, and D. Hutchison, "Scalable Bloom Filters," *Information Processing Letters*, vol. 101, no. 6, pp. 255–261, Mar. 2007.

[7] F. Hao, M. Kodialam, and T. V. Lakshman, "Incremental Bloom Filters," *Proc. 27th IEEE INFOCOM*, pp. 1741–1749, Phoenix, AZ, USA, Apr. 2008.

[8] J. Wei, H. Jiang, K. Zhou, and D. Feng, "MAD2: A Scalable High-Throughput Exact Deduplication Approach for Network Backup Services," *Proc. 26th IEEE MSST*, Incline Village, NV, USA, May 2010.

[9] A. Z. Broder and M. Mitzenmacher, "Network Applications of Bloom Filters: A Survey," *Internet Mathematics*, vol. 1, pp. 485–509, 2005.

[10] Y. Hua, Y. Zhu, H. Jiang, D. Feng, and L. Tian, "Scalable and Adaptive Metadata Management in Ultra Large-scale File Systems," *Proc. 28th ICDCS*, Beijing, China, June 2008.