# Advanced Supporting Scalable and Adaptive Metadata Management in Ultra Large-Scale File Systems

Rajalakshmi.K
*Research Scholar, Department of MCA, School of Computing Science*
*VELS University, Pallavaram, Chennai-117, Tamilnadu, India.*

Veeraragavan.V
*Asst. Professor, Department of MCA, School of Computing Science*
*VELS University, Pallavaram, Chennai,-117, Tamilnadu, India.*

Dr.A Muthukumaravel
*Asst. Professor, Department of MCA, School of Computing Science*
*VELS University, Pallavaram, Chennai-117, Tamilnadu, India.*

**Abstract-This paper shows a scalable and adaptive decentralized metadata lookup scheme for ultra large-scale file systems Petabytes or even Exabytes. Our scheme logically creates metadata servers (MDS) into a multi-layered query hierarchy and exploits grouped filters Bloom to efficiently route metadata requests to desired MDSs through the hierarchy. This metadata lookup scheme can be performed at the network or memory speed. An effective workload balance algorithm is also developed in this paper for server reconfigurations. This scheme is calculated through extensive trace-driven simulations and prototype implementation in Linux. This scheme can significantly improve metadata management scalability and query efficiency in ultra large-scale storage systems.**

**Keywords – MDS, G-HBA, BF**

## I. INTRODUCTION

Metadata management is critical in scaling the overall performance of large-scale data storage systems [1]. To achieve high data throughput, many systems decouple metadata transactions from file content accesses by diverting large volumes of data traffic away from dedicated metadata servers (MDS) [2]. In such systems, a client contacts MDS first to acquire access permission and obtain desired file metadata, such as file location and attributes, and then directly accesses file content stored on data servers without going through the metadata server. While the storage demand increases exponentially in recent years, exceeding petabytes (1015) already and reaching exabytes (1018) soon, such decoupled design with a single metadata server can still become a severe performance bottleneck. It has been shown that metadata transactions account for over 50% of all file system operations [3]. In scientific or other data-intensive applications [4], the file size ranges from a few bytes to multiple terabytes, resulting in millions of pieces of metadata in directories [5]. Thus, scalable and decentralized metadata management schemes have been proposed to scale up the metadata throughput by judiciously distributing heavy management workloads among multiple metadata servers while maintaining a single writable namespace image.

One of the most important issues in distributed metadata management is to provide efficient metadata query service. Existing query schemes can be classified into two categories: probabilistic lookup and deterministic lookup. In the latter, no broadcasting is used at any point in the query process. For example, a deterministic lookup typically incurs a traversal along a unique path within a tree-structured global directory. The probabilistic approach employs lossy data representations, such as Bloom filters [6], to route a metadata request to its target MDS with a very high accuracy. Certain remedy strategy, such as broadcasting or multicasting, is needed for rectifying incorrect routing. Compared with the deterministic approach, the probabilistic one can be easily adopted in distributed systems and allows flexible workload balance among metadata servers.

A large-scale distributed file system must provide a fast and scalable metadata lookup service. In large-scale storage systems, multiple metadata servers are desirable for improving scalability. The proposed scheme in this paper, called Group-based Hierarchical Bloom filter Array (G-HBA), judiciously utilizes Bloom filters to efficiently route requests to target metadata servers. Our G-HBA scheme extends our previous Bloom filter-based architecture by considering dynamic and self-adaptive characteristics of ultra large-scale file systems.

We utilize an array of Bloom filters on each MDS to support distributed metadata management of multiple MDSs. An MDS where a file's metadata resides is called the home MDS of this file. Each metadata server further constructs

a Bloom filter to represent all files whose metadata are stored locally and then replicates this Bloom filter to all other MDSs. A metadata request from a client can randomly choose an MDS to perform membership query against its Bloom filter array that includes replicas of the Bloom filters from all other MDSs. The Bloom filter array returns a hit when exactly one filter gives a positive response. A miss takes place when zero hit or multiple hits are found in the array. Since we assume that original metadata can be stored only in one MDS, multiple hits, meaning that original metadata are found in multiple MDSs, indicate a query miss. The basic idea behind G-HBA in improving scalability and query efficiency is to decentralize metadata management among multiple groups of MDSs. We divide all N MDSs in the system into multiple groups with each group containing at most M MDSs. Note that we represent the actual number of MDSs in a group as M. By judiciously using space-efficient data structures, each group can provide an approximately complete mapping between individual files and their home MDSs for the whole storage system. While each group can perform fast metadata queries independently to improve the metadata throughput, all MDSs within one group only store a disjointed fraction of metadata and they cooperate with each other to serve an individual query. The rest of the paper is organized as follows. Section 2 presents the basic scheme of G-HBA and its design issues. The performance evaluation based on trace-driven simulations and prototype implementations are given in Section 3 and Section 4, respectively. Section 5 summarizes related work and Section 6 concludes the paper.

## II. G-HBA Design

*Group-based HBA Scheme*

In this section, we present a novel approach, called Group-based Hierarchical Bloom filter Array (G-HBA), to carry out scalable and adaptive metadata management, specifically to facilitate fast membership queries in ultra large-scale storage systems.

A query process at one MDS may involve four hierarchical levels: (1) searching the locally stored LRU BF Array (L1), (2) searching the locally stored Segment BF Array (L2), (3) multicasting to all MDSs in the same group to concurrently search all Segment BF Arrays stored in this group (L3), and (4) multicasting to all MDSs in the system to directly search requested metadata (L4). The multi-level metadata query is designed to judiciously exploit access locality and dynamically balance load among MDSs.

Each query is performed sequentially in these four levels. A miss at one level will lead to a query to the next higher level. The query starts at the LRU BF array (L1), which aims to accurately capture the temporal access locality in metadata traffic streams. If the query cannot be successfully served at L1, the query is then performed L2, as shown in Figure 1(a). The segment BF array (L2) stored on MDSs includes only θi BF replicas, with each replica representing all files whose metadata are stored on that corresponding MDS. Suppose that the total number of MDS is N, and typically θi is much smaller than N. And we have ∑Mi=1 θi = N. In this way, each MDS only maintains a subset of all replicas available in the systems. A lookup failure at L2 will lead to a query multicast among all MDSs within the current group (L3), as shown in Figure 1(b). At L3, all BF replicas avail-able in this group will be checked. At the last level of the query process, i.e., L4, each MDS directly performs lookup by searching its local BF and disk drives. If the local BF responds negatively, the requested metadata is not stored lo-cally on that MDS since the local BF has no false negatives. However, if the local BF responses positively, a disk access is then required to verify the existence of requested metadata since the local BF can potentially generate false positives.

## III. CRITICAL PATH

The critical path of a metadata query starts at L1. When the L1 Bloom filter array returns a unique hit for the mem-bership query, the target metadata is then most likely to be found at the MDS whose LRU Bloom filter generates such a unique hit. If zero or multiple hits take place at L1, imply-ing a query failure, the membership query is then performed on the L2 segment Bloom filter array, which maintains map-ping information for a fraction of the entire storage system by approximately storing θ = ₃ N−MM₃ ₃ replicas. A unique hit in any L2 segment Bloom filter array does not neces-sarily indicate a query success since (1) Bloom filters only provide probabilistic membership query and a false posi-tive may occur with a very small probability, and (2) each MDS only contains a subset of all replicas and thus is only knowledgeable of a fraction of the entire file-server map-ping. The penalty for a false positive, where a unique hit fails to correctly identify the home MDS, is that a mul-ticast must be performed within the current MDS group (L3) to solve this miss-identification. The probability of a false positive from the segment Bloom filter array of one MDS, $f_g^+$, is $f_g^+ = C\theta^1 f_0(1-f_0)^{\theta-1} = \theta(0.6185)^{m/n}(1-$

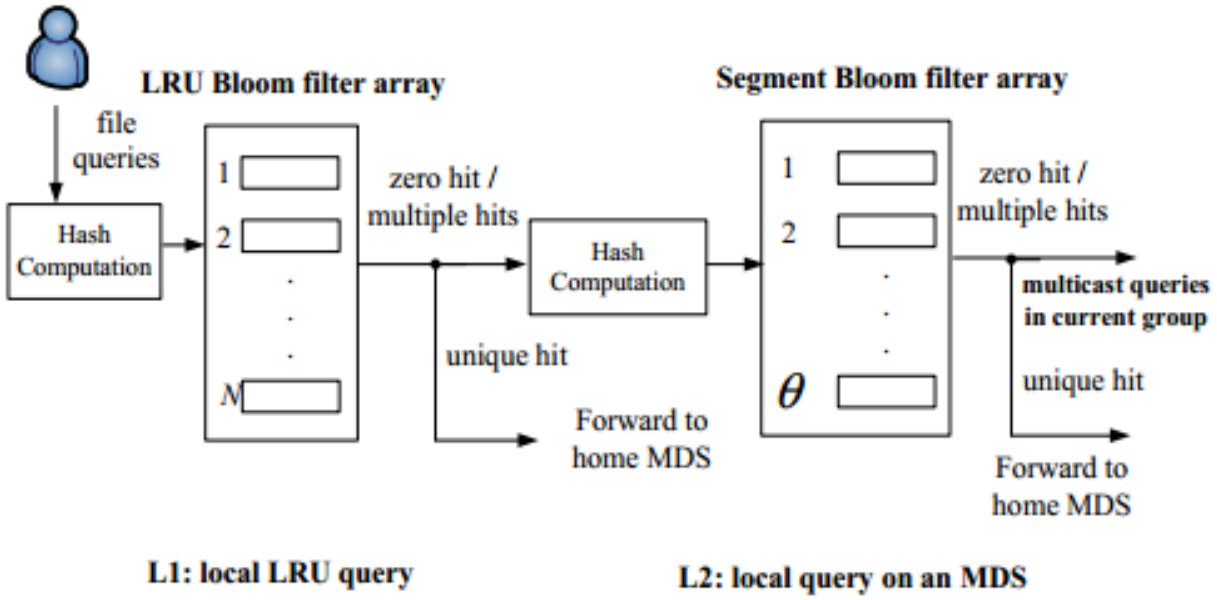
$(0.6185)^{m/n})^{\theta-1}$, where $\theta$ is the number of BF replicas stored locally on one MDS, *m/n* is the Bloom filter bit ratio, i.e., the number of bits per file, and $f_0$ is the optimal false rate in standard Bloom filters [7]. By storing only a small subset of all replicas and thus achieving significant memory space savings, the group-based approach (segment Bloom filter array) can afford to increase the number of bits per file (*m/n*) so as to significantly decrease the false rate of its Bloom filters, hence rendering $f_g^+$ sufficiently small.

- A query process at one MDS may involve four hierarchical levels: searching the locally stored LRU BF Array (L1), searching locally stored Segment BF Array (L2), multicasting to all MDSs in the same group to concurrently search all Segment BF arrays stored in this group (L3), and multicasting to all MDSs in the system to directly search requested metadata (L4). The multi-level metadata query.
- Each query is performed sequentially in these four levels. A miss at one level will lead to a query to the next higher level. Which aims to accurately capture the temporal access locality in metadata traffic streams. If the query cannot be successfully.

## IV. Dynamic and Adaptive Metadata

Bloom filters on each MDS to support distributed metadata management of multiple MDSs. An MDS where a file's metadata resides is called the home MDS of this file. Each metadata server further constructs a Bloom filter to represent all files whose metadata are stored locally and then replicates this filter to all other MDSs. A metadata request from the client can randomly choose an MDS to perform membership query against its Bloom filter array that includes replicas of the Bloom filters from the other servers. The Bloom filter array returns a hit when exactly one filter gives a positive response. A miss takes place when zero hit or multiple hits are found in the array. The basic idea behind G-HBA in improving scalability and query efficiency is to decentralize metadata management among multiple groups of MDSs. We divide all N MDSs in the system into multiple groups with each group containing at most M MDSs. Note that we represent the actual number of MDSs in a group as M. By judiciously using space-efficient data structures, each group can provide an approximately complete mapping between individual files and their home MDSs for the whole storage system. While each group can perform fast metadata queries independently to improve the metadata throughput, all MDSs within one group only store a disjointed fraction of metadata and they cooperate with each other to serve an individual query.

A simple grouping in G-HBA may introduce large query costs and does not scale well. Since each MDS only maintains partial information of the entire file system, the probability of successfully serving a metadata query by a single metadata server will decrease as the group size increases.




(a) LRU and segment Bloom filter arrays allowing the L1 and L2 queries.

Accordingly an MDS has to multicast query requests more frequently to other MDSs, incurring higher network overheads and resulting in longer query delays. Accordingly, more effective techniques are needed to improve the scalability of the group-based approach. G-HBA addresses this issue by taking advantages of the locality widely xhibited in metadata query traffic. Specifically, each MDS is designed to maintain "hot data", i.e., home MDS information for recently accessed files that are stored in an LRU Bloom filter array. Since "hot data" are typically small in size, the required storage space is relatively small.

## V. BLOOM FILTER ALGORITHM

*a.Algorithm for Insertion*

*Algorithm 1 ABF: Algorithm for Insertion*
*Require: B, the bit-vector and v, input element*
*Ensure: N, number of additional hash function*
*if all B[H1..k(v)] = 1 then*
*N ⇐ 1*
*while B[Hk+N(v)] = 1 do*
*N ⇐ N + 1*
*end while*
*B[Hk+N(v)] = 1*
*Else*
*all B[H1..k(v)] = 1*
*end if*

In this section, we describe the insert algorithm on ABF (Algorithm 1). All of these algorithms are simple toimplement. BF usually uses fixed k hash functions, but ABF uses k+N+1 independent hash functions. In algorithm 1, check whether the bit located Hk+N is set to 1 while being increment N. The bit located H(k+N+1) is set to 1.

*b.Algorithm for a Query*

*Algorithm 2 ABF: Algorithm for a Query*
*Require: B, the bit-vector and v, input element*
*Ensure: N, number of additional hash function*
*if all B[H1..k(v)] = 1 then*

*N ⇐ 1*
*while B[Hk+N(v)] = 1 do*
*N ⇐ N + 1*
*end while*
*return N*
*end if*

In algorithm 2, we first check whether all of Hk is set to 1 or not. This operation is the same as BF. Next, we check whether all of Hk+N is set to 1 or not. We get parameter N, which means that the number of additional hash function within the bit-vector is set in the same way as the insert Algorithm (Algorithm 1). In other words, N represent the quantity of elements whose accuracy depends on occupancy rate p.

When the segment Bloom filter of an MDS returns zero or multiple hits for a given metadata lookup, indicating a local lookup failure, this MDS then multicasts the query request to all MDSs in the same group, in order to resolve this failure within this group. Similarly, a multicast is necessary among all other groups, i.e., at the L4 level, if the current group returns zero or multiple hits at L3.

*c.Updating Replica*

Updating stale Bloom filter replicas involves two steps, replica identification (localization) and replica content update. Within each group, a BF replica resides exclusively on one MDS. Furthermore, the dynamic and adaptive nature of server reconfiguration, such as MDS insertion into or deletion from a group (see Section 3.1), dictates that a given replica must often migrate from one MDS to another within a group. Thus, to update a BF replica, we must correctly identify the target MDS in which this replica currently resides. This replica location information is stored in an identification (ID) Bloom filter array (IDBFA) that is maintained in each MDS, as shown in Figure 3. A unique hit in IDBFA returns the MDS ID, thus allowing the update to proceed to the second step, i.e., updating BF replica at the target MDS. Multiple hits in IDBFA lead to a light false positive penalty since a falsely identified target MDS can simply drop the update request after failing to find the targeted replica. The probability of such false positive can be extremely low. Counting Bloom filters are used in IDBFA to support server departure. Since IDBFA only maintains the information about where a replica can be accessed, the total storage requirement of IDBFA is negligible. For example, when the entire file system contains 100 MDSs, IDBFA only takes less than 0.1KB. storage on each MDS.*s*

## *V*I. OPERRATION ANALYSIS

An MDS departure triggers a similar process but in a reverse direction. It involves (1) migrating replicas previously stored on the MDS to the other MDSs within that group, (2) removing its corresponding Bloom filter from the IDBFA on each MDSs of that group, and (3) sending a message to the other groups to delete its replica. The network overhead of this design is small since group reconfiguration happens infrequently and the size. Group Splitting and Merging To further minimize the replica management overhead, we propose to dynamically perform group splitting and merging. When a new MDS is added to a group G that already has $M' = M$ MDSs, a group split operation is then triggered to divide this group into two approximately equal-sized groups, A and B. The split operation will be performed under two conditions: (1) each groups must still maintain a global mirror image of the file system and (2) workload must be balanced within each group. After splitting, A and B consist of $M - \lfloor M/2 \rfloor$ and $\lfloor M/2 \rfloor + 1$ MDSs, respectively, for a total of $(M + 1)$ MDSs. The group splitting process is equivalent to deleting $\lfloor M/2 \rfloor$ MDSs from G by applying the aforementioned MDS deletion operation $\lfloor M/2 \rfloor$ times. Each deleted MDS from G is then inserted into group B. Inversely, whenever the total size of two groups is equal to or less than the maximum allowed group size M due to MDS departures, these groups are then merged into a single group by using the light-weight migration scheme. This process repeats until no merging can be performed.

## VII. PERFORMANCE EVALUATION

G-HBA through trace-driven simulations and compare it with HBA [29], the state-of-theart BF-based metadata management scheme and one that is directly comparable to G-HBA. We use three publicly available traces, i.e., Research Workload (RES), Instructional Workload (INS) [3] and HP File System Traces [32]. In order to emulate the I/O behaviors in an ultra large-scale file system, we choose to intensify these workloads by a combination of spatial scale up and temporal scale-up in our simulation and also in prototype experiments presented in the next section. We decompose a trace into subtraces and intentionally force them to have disjoint group ID, user ID and working directories by appending a subtrace number in each record. The timing relationships among the requests within a subtrace are preserved to faithfully maintain the semantic dependencies among trace records

## VIII.CONCLUSION

This paper presents a scalable and adaptive metadata lookup scheme named Group-based Hierarchical Bloom filter Arrays (G-HBA) for ultra large-scale file systems. G-HBA organizes MDSs into multiple logic groups and utilizes grouped Bloom filter arrays to efficiently direct a metadata request to its target MDS. The novelty of G-HBA lies in that it judiciously limits most of metadata query and Bloom filter update traffic within in a server group. Compared with HBA, G-HBA is more scalable due to the facts that: 1) G-HBA has a much less memory space overhead and thus can potentially avoid accessing disks during metadata lookups in exabyte-scale storage systems. 2) G-HBA significantly reduces global broadcasts among all MDSs, such as Bloom filter updates. 3) G-HBA supports dynamic workload rebalancing when the server number changes, by using a simple but efficient migration strategy. Extensive trace-driven simulations and real implementations show that our G-HBA is highly effective and efficient in improving the performance, scalability and adaptability of the metadata management component.

REFERENCES

[1] R. J. Honicky and E. L. Miller, "Replication under scalable hashing: A family of algorithms for scalable decentralized data distribution," IEEE IPDPS, April 2004.

[2] L. Fan, P. Cao, J. Almeida, and A. Z.Broder, "Summary cache: a scalable wide area web cache sharing protocol," IEEE/ACM Trans. on Networking, vol. 8, no. 3, 2000.

[3] A. Chervenak, N. Palavalli, S. Bharathi, C. Kesselman, and R. Schwartzkopf, "Performance and scalability of a replica location service," HPDC, 2004.

[4] Y. Zhu, H. Jiang, and J. Wang, "Hierarchical Bloom filter arrays (HBA): A novel, scalable metadata management system for large cluster-based storage," IEEE Cluster Computing, 2004 (Journal version accepted by IEEE TPDS, 2007).

[5] A. Broder and M. Mitzenmacher, "Network applications of Bloom filters: a survey," Internet Mathematics, vol. 1, pp. 485–509, 2005.

[6] Y. Hua and B. Xiao, "A multi-attribute data structure with parallel Bloom filters for network services.," IEEE HiPC, pp. 277–288, 2006.

[7] E. Riedel, M. Kallahalla, and R. Swaminathan, "A framework for evaluating storage system security," FAST, pp. 15–30, 2002.

[8] Y. Zhu and H. Jiang, "False rate analysis of Bloom filter replicas in distributed systems," ICPP, pp. 255–262, 2006.

[9] J. Kubiatowicz, D. Bindel, Y. Chen, P. Eaton, D. Geels, R. Gummadi, S. Rhea, H. Weatherspoon, W. Weimer, C. Wells, and B. Zhao, "Oceanstore: An architecture for global-scale persistent storage.,"ACM ASPLOS, Nov.2000.

[10] S. Moon and T. Roscoe, "Metadata Management of Terabyte Datasets from an IP Backbone Network: Experience and Challenges,"NRDM, 2001.