Hierarchical Bloom Filter Arrays (HBA): A Novel, Scalable Metadata Management System for Large Cluster-based Storage

Yifeng Zhu, Hong Jiang and Jun Wang
Department of Computer Science and Engineering
University of Nebraska, Lincoln, NE, USA
Email: {yzhu, jiang, wang}@cse.unl.edu

Abstract

An efficient and distributed scheme for file mapping or file lookup scheme is critical in decentralizing metadata management within a group of metadata servers. This paper presents a novel technique called HBA (Hierarchical Bloom Filter Arrays) to map file names to the servers holding their metadata. Two levels of probabilistic arrays, i.e., Bloom Filter Arrays, with different accuracies are used on each metadata server. One array, with lower accuracy and representing the distribution of the entire metadata, trades accuracy for significantly reduced memory overhead, while the other array, with higher accuracy, caches partial distribution information and exploits the temporal locality of file access patterns. Extensive trace-driven simulations have shown our HBA design to be highly effective and efficient in improving performance and scalability of file systems in clusters with 1,000 to 10,000 nodes (or super-clusters).

1 Introduction

Rapid advances in general-purpose communication networks have motivated the deployment of inexpensive components to build competitive cluster-based storage solutions to meet the increasing demand of scalable computing [4, 5, 7, 15, 27, 30]. In the recent years, the bandwidth of these networks has been increased by two orders of magnitude [3, 8, 12], which greatly narrows the performance gap between them and the dedicated networks used in commercial storage systems, such as fiber channels. The significant improvement in network bandwidth offers an appealing opportunity to provide cost-effective high-performance storage services by aggregating the existing storage resources on each commodity PC in a computing cluster with such networks if a scalable scheme is in place to efficiently virtualize these distributed resources into a single-disk image. The key challenge in realizing this objective lies in the potentially huge number of nodes (in thousands) in such a cluster. Currently clusters with thousands of nodes are already in existence and clusters with even larger number of nodes are expected in the near future.

Since all I/O requests can be classified into two categories, the user data requests and the metadata requests, the scalability of accessing both data and metadata has to be carefully maintained to avoid any potential performance bottleneck along all data paths. To divert the high volume of user data traffic to bypass any single centralized component, the functions of data and metadata managements are usually decomposed and the metadata is stored separately on different nodes away from the user data. While previous work on cluster-based storage mainly focuses on optimizing the scalability and efficiency of user data accesses by using a RAID style striping [7, 31], caching [28], scheduling [18, 32] and networking [29], very little attention has been drawn to the scalability of the metadata management.

Yet, the efficiency of the metadata management is critical for the overall performance of cluster-based storage systems. It not only provides file attributes and data block addresses, but also synchronizes concurrent updates, enforces access control, supports recovering and maintains consistency between user data and file metadata. A study on the file system traces collected in different environments over a course of several months shows that requests targeting at the metadata can account for up to 83% of the total number of I/O requests [11]. Under such skewed load to metadata, a centralized metadata management system certainly will not scale well with the cluster size. As the number of files or I/O requests increases, the throughput of metadata operations on a single metadata server can be severely limited.

This paper proposes a novel scheme, called *Hierarchical Bloom Filter Array* (HBA), to evenly distribute the tasks of metadata management onto a group of metadata servers. A Bloom filter is a succinct data structure for probabilistic membership query. We identify that a straightforward adoption of Bloom filers is impractical due to the memory space overhead when the number of files is very large. By exploiting the temporal access locality of the file access pattern, we use a small Bloom filter array with a high accuracy at the first level of the hierarchy to capture the destination metadata server information of some frequently accessed files to keep high management efficiency while reducing the memory overhead. At the second level of the hierarchy, a pure Bloom array (PBA), with lower accuracy in favor of memory efficiency, is used to maintain the destination metadata information of all files. Extensive trace-driven simulations have shown HBA to be capable of offering significant performance and cost advantages over PBA alone or pure global LRU lists.

This paper has the following technical contributions:

- It analyzes the performance of the pure Bloom filter approach by using both theoretical models and trace simulations. The efficiency and scalability of this approach are examined under different workloads and cluster configurations.

- It proposes and evaluates a hybrid approach that uses hierarchical structures. It explores the impacts of different parameters to optimize the tradeoff between the efficiency of metadata distribution and management, and the memory and network overhead.

- It compares both HBA and PBA schemes using two artificially scaled-up large file system traces that emulate file systems of up to 1300 nodes and 710 active users.

- HBA attempts to optimize the tradeoff between the efficiency and the network and memory overhead. To achieve high metadata look-up efficiency, PBA with high accuracy must be used and thus it suffers severely from large memory overhead. On the other hand, to maintain the same efficiency, a scheme using only pure LRU lists has to be updated very frequently and thus it suffers from enormously large network traffic overhead. HBA employs a hierarchical structure integrating a PBA with a lower accuracy (to significantly reduce memory overhead) with a pure LRU scheme with a lower update frequency to achieve a good tradeoff between the efficiency and the memory and network overhead. As a result, HBA achieves a high efficiency without suffering either memory or network overhead.

The rest of the paper is organized as follows. Section 2 outlines the existing approaches to decentralizing the metadata management in large cluster-based file systems. Section 3 describes the proposed architecture and the design objectives. Section 4 presents in detail the design of the HBA scheme. The simulation methodology is presented in Section 5, while the performances of our design are evaluated in Section 6. Section 8 concludes the paper.

2 Related Work and Comparisons of Decentralization Schemes

Centralized metadata management is employed in many cluster-based storage systems. Google file system (GFS) [15] uses only one metadata server. Experiments for GFS, conducted in a storage cluster with 100 nodes, indicate that this single metadata server is not a performance bottleneck under the specific workload in a data searching environment, where the number of files stored in GFS is modest and file access patterns are less complicated than the workload in a general file system. In the GFS study, only a few million files were expected and the accesses to these files were almost read-only, once initially written. PVFS [7], a RAID-0 style parallel file system, also employs a single metadata server design to provide a cluster-wide shared namespace. As the performance is the most important objective of PVFS, some expensive but indispensable functions, such as the concurrent control and consistency maintenance between data and metadata, are not fully designed and implemented. In CEFT-PVFS [30, 31, 32, 33] an extension of PVFS to incorporate a RAID-10 style parallel I/O and dynamic load-balancing, the metadata server synchronizes the concurrent updates and this synchronization enforcement can limit the overall throughput under the workload of intensive concurrent metadata updates. While Lustre [4] also uses a centralized metadata management in its current design, undergoing efforts are being made to further improve its scalability by decentralizing the metadata management.

Static namespace partition is a simple way of distributing metadata operations to a group of metadata servers. A common partition technique has been to divide the directory tree during the process of installing or mounting and store the information at some well-known locations. Some distributed file systems, such as NFS [21], AFS [20], and Coda [25], follow this approach. This scheme works well only when file access patterns are uniform, resulting in a balanced workload. Unfortunately, access patterns in general file systems are highly skewed [9, 14, 23, 26] and thus this partition scheme can lead to highly imbalanced workload if files in some particular subdirectories become more popular.

Another decentralized scheme is to implement a globally replicated mapping table. There is a salient tradeoff between the space requirement and the granularity and flexibility of distribution. A fine-grained table allows more flexibility metadata placement. In an extreme case, if the table records the home server of the metadata for each file, then the metadata of a file can be placed on any metadata server. However, the memory space requirement for this approach makes it unattractive for large scale storage systems. A back-of-the-envelope calculation shows that it would take as much as 1.8 GB memory to store such a table with one hundred million entries if 16 bytes and 2 bytes are used to represent the file name and metadata server ID respectively. In addition, searching an entry in such a huge table consumes a large number of precious CPU cycles. To reduce the memory space overhead, xFS [1] proposes a coarse-grained table that maps a group of files to a metadata server. To keep a good tradeoff, it is suggested in XFS that the number of entries in a table should be an order of magnitude larger than the total metadata server number.

Modulus-based hashing is also a widely used decentralized scheme. This approach hashes a symbolic pathname of a file to a digital value and assigns its metadata to a server according to the modulus value with respect to the total metadata server number. In practice, the likelihood of serious skew of metadata workload is almost negligible in this scheme since the number of frequently accessed files is usually much larger than the number of metadata servers. However, a serious problem with this scheme is that metadata needs to be migrated to new servers after renaming of a file or directory and additions or deletions of metadata servers. Although the size of the metadata of a file is small, a large number of files may be involved during a directory renaming. In particular, the metadata of all files has to be relocated if a metadata server joins or leaves. This could lead to both disk and network traffic surges and cause serious performance degradation. While LazyHybrid [5] is proposed to reduce the influence of metadata migration by incorporating a small table that maps disjointed hash ranges to server IDs and other techniques, such as lazy updating, the migration overhead can still overweight the benefits from load balancing in a busy distributed storage system.

3 Architectural Considerations and Design Objectives

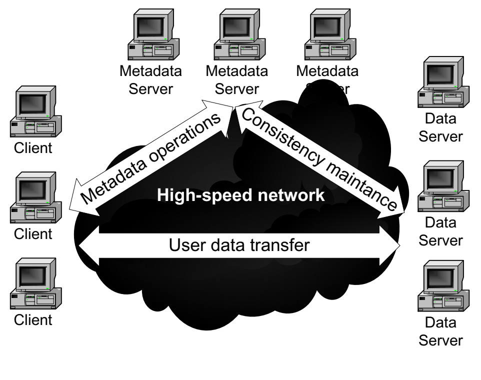

Figure 1. Cluster-based storage architecture.

In this paper, we focus on a generic cluster where a number of commodity PCs are connected by a high-bandwidth low-latency switched network. Each node has its own storage devices. There are no functional differences between all cluster nodes. The role of clients, metadata servers, and data servers can be carried out by any node and a node may not be dedicated to a specific role. It can act in multiple roles simultaneously. Figure 1 shows the architecture of a generic cluster targeted in this paper.

In this study, we concentrate on the scalability and flexibility aspects of metadata management. Some other important issues, such as consistency maintenance, synchronization of concurrent accesses, file system security and protection enforcement, free space allocation (or garbage collection), balancing the space utilizations, management of the striping of file contents, and incorporation of fault tolerance, are beyond the scope of this study. Instead, the following objectives are considered in our design:

- Single shared namespace. All storage devices are virtualized into a single image and all clients share the same view of this image. This requirement simplifies the management of user data and allows a job to run on any node in a cluster.
- Scalable service. The throughput of a metadata management system should scale with the computation power of a cluster. It should not become a performance bottleneck under high I/O access rate. This requires the system to have a low management overhead.
- Zero metadata migration. Although the size of the metadata is small, the number of files in a system can be enormously large. In a metadata management system that requires metadata to migrate to other servers in responses to the file system's evolution, such as renaming of files or directory, or the topology changes involving server additions or departures, the computational overhead of checking whether a migration is needed and the network traffic overhead due to metadata migration may be prohibitively large, hence limiting the efficiency and scalability.
- Balancing the load of metadata accesses. The management is evenly shared among multiple metadata servers to best leverage the available throughput of these severs.
- Flexibility of storing the metadata of a file on any metadata server. This flexibility provides the opportunity for fine-grained load balance, simplifies the placement of metadata replicas, and facilitates some performance optimizations, such as metadata prefetching [19, 24]. In a distributed system, metadata prefetching requires the flexibility of storing a group of sequentially accessed files on the same physical location to save the number of metadata retrievals.

4 Hierarchical Bloom Filter Arrays

4.1 Bloom Filters

A Bloom filter is a fast and space-efficient method to support probabilistic membership queries. It was invented by Burton Bloom in 1970 [2] and widely used for web caching [13], network routing [6], and prefix matching [10]. A Bloom filter aims to reduce the memory requirement of representing a set of elements at the nominal cost of a very small probability of false hits. A false hit occurs when a Bloom filter positively confirms the membership of an element while this element actually does not belong to this set. The probability that a false hit occurs is called the false positive rate or false hit rate.

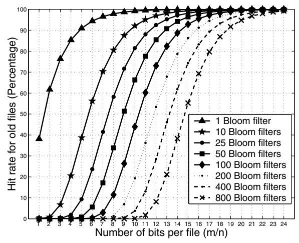

Figure 2. Theoretical hit rates for existing files.

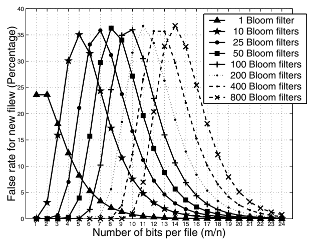

Figure 3. Theoretical false hit rates for new files.

4.2 Hierarchical Bloom Array Design

4.2.1 Pure Bloom Filter Array (PBA) Approach

A straightforward extension of the Bloom filter approach to decentralizing metadata management onto multiple metadata servers is to use an array of Bloom filters on each metadata server. The metadata of each file is stored on some metadata server, called the home metadata server. In this design, each metadata server builds a Bloom filter that represents all files whose metadata is stored locally and then replicates this filter to all the other metadata servers. Including the replicas of the Bloom filters from all the other servers, a metadata server stores all filters in a array. When a client initiates a metadata request, the client randomly chooses a metadata server and asks this server to perform the membership query against this array. The Bloom filter array is said to have a hit if exactly one filter gives a positive response. A miss is said to have occurred whenever there is no hit or more than one hit found in the array. The desired metadata can be found on the hit Bloom Filter with a very high probability.

When an existing file is searched, a false positive hit from any Bloom filter can lead to multiple hits and thus causes the search to fail. If all Bloom filters are perfectly updated, the hit rate for an existing file is the probability that all Bloom filters have no false positive hits, given as follows

$$hit_{oldfiles} = (1-f)^p = (1-(0.6185)^{m/n})^p \qquad (1)$$

where m is the length of a Bloom filter, n is the number of files that a single metadata contains and p is the total number of metadata servers and f is the optimal false rate of a single Bloom filter. Figure 2 shows the relationship between $hit_{oldfile}$ and m/n under different numbers of metadata servers.

For new files, a false hit happens when exactly one Bloom filter gives a false positive response. The false positiveness will be discovered eventually when the desired metadata actually does not exist on the falsely identified metadata server. The false hit rate can be expressed as

$$\begin{aligned} false_{newfile} &= pf(1-f)^{p-1} \\ &= p(0.6185)^{m/n}(1-(0.6185)^{m/n})^{p-1} \end{aligned}$$

Given a constant p, $false_{newfile}$ reaches its maximum value $(1-\frac{1}{p})^{p-1}$ when $f = \frac{1}{p}$, i.e., $m/n = 2.0792 \ln p$. This maximum value approaches asymptotically to $e^{-1} \approx 0.3679$. This trend of $false_{newfile}$ with respect to m/n under different numbers of metadata servers is given in Figure 3. This trend shows a special characteristic of a Bloom filter array that is different from that of a Bloom filter. While in a Bloom filter, increasing the filter length always reduces its false hit rate, *the false hit rate of a Bloom filter array actually increases with the filter size until reaching its maximum false hit rate*. This observation is important in optimizing the bit/file ratio for Bloom filter arrays.

While optimizing the tradeoff between the space efficiency and the response accuracy, more weights are put on improving $hit_{oldfile}$ than decreasing $false_{newfile}$ since almost all I/O requests are targeted at existing files in a typical file system,. This biased optimization might not work well in any special environment where the operations of file creation account for a considerably high percentage of the total file accesses.

This approach provides a flexible metadata placement, has no migration overhead, and balances the metadata workload. PBA does not rely on any property of a file to place its metadata and thus allows the system to place any metadata on any server. This makes it feasible to group metadata with strong locality together for prefetching, a technology that has been widely used in conventional file systems [19, 24]. During evolvement of the file system and the cluster topology, not all metadata needs to migrate to new locations. When a file or directory is renamed, only the Bloom filters associated with all the involved files or subdirectories need to be updated. While a metadata server leaves or joins the system, a single associated Bloom filter is added or deleted from the Bloom arrays on all other metadata servers. Since each client randomly chooses a metadata server to lookup for the home metadata server of a file, the query workload is balanced on all metadata servers.

The major disadvantage of this approach is that storing the Bloom filter array requires a large memory space. For example, if there are 200 metadata servers in a supercluster, 16 bits per file are required in each Bloom filter to maintain a hit rate around 90% for old files and a false hit rate 10% for new files. If there are 500 million files stored in this cluster, the Bloom filter array would take around $16 \times 500Mb = 1GB$ memory space on each metadata server. This memory requirement is underestimated since, in practice, the hit rates can be lower than the theoretical results. This implies that an even higher bit/file needs to be employed. In a web caching design system [13], a ratio of 32 is suggested.

4.2.2 Hierarchical Bloom Array (HBA) Design

To achieve a sufficiently high hit rate in the pure Bloom filter array approach described above, the high memory overhead may make this approach impractical due to the fact that a large bit/hit ratio needs to be employed to achieve a high hit rate when the number of metadata server is large. In this section, we present the design of the Hierarchical Bloom Array (HBA) to reduce the memory overhead while achieving a competitively high hit rate.

The novelty of the HBA design lies in its judicious exploitation of the fact that in a typical file system, a small fraction of files absorb most of the I/O activities. Ref. [14] discovered that 66% of all files had not been access in over a month in a UNIX environment, indicating that the entire I/O accesses were focused on at most 34% of the file system. Ref. [26] found that 0.1% of the total space used by the file system received $30 - 60\%$ of the I/O activity. Ref. [9] show that most files in UNIX file systems were inactive and only $3.6\% - 13\%$ of the file-system data was used in a given day, and only $0.2 - 3.6\%$ of the I/O activity went to the least active 75% of the file system. A recent study [27] on a file system trace collected in December 2000 from a mid-sized file server found that only 2.8% and 24.2% percentages of files that were accessed during a continuous course of 12 hours and 10 days, respectively.

Figure 4 shows the structure of the HBA design on each metadata server, including two levels of Bloom filter arrays. In the design, each metadata server maintains a LRU (Least-Recently-Used) list that caches names of recently visited files whose metadata is stored on that metadata server. Each Bloom filter at the first level, called a LRU BF, represents all the files cached in the LRU list of the corresponding metadata server. Each LRU BF is globally replicated among all metadata servers. Whenever an overflow happens in the LRU list, an eviction based on the LRU replacement policy triggers both an addition and deletion operations to its corresponding LRU BF. Only when the portion of changes made to a LRU BF has exceeded some threshold, will the LRU BF be multicast to all the metadata servers to update all its replicas. Since the number of entries in LRU is relatively small, it is affordable to use a large bit/file ratio to achieve a low false hit rate. In addition, the Bloom filters in the second level represent the metadata distributions of all metadata servers. Since the totally number of files is typically very large, a small bit/file ratio is used to reduce the memory overhead. A miss in the first level array leads to a query to the second level. An unsuccessful query in the second level array will cause a broadcast to be issued to all the other metadata severs. It must be noted that the penalty for a miss or a false hit can be very expensive, relative to the hit time, since it entails, among other things, a broadcast over the interconnection network, a query on a second metadata server and an acknowledgement across the network.

To perform a query into the Bloom filters, the file names are transformed into digital indexes into of the Bloom array by first calculating the MD5 signature of the full pathname and then hashing the MD5 signature into indices by using the universal hash functions [22]. Without calculating the MD5 of the full pathname, files with the same name will be hashed to the same location, even if they are in different directories. The MD5 approach is chosen because its available fast implementation. The universal hash functions are employed to keep the independence of hash indices, a requirement of Bloom filter to minimize the false hit rate.

Locating the metadata by hashing the full pathname will complicate the access control since all parent directories are bypassed. The same technique used in [5] can be employed here to deal with the access control issue. Two UNIX style access permission codes, including the permission code of the file per se and the intersection of access permissions of all parent directories, are maintained in the metadata of each file and checked for each file access. A file is only accessible only when both codes permit. A downside of this solution is that populating the permission changes of a directory to its children may potentially result in a large number of network messages.

5 Trace Driven Simulation

5.1 File System Traces

To the best of our knowledge, there are no publicly available file system traces that have been collected from a large-scale cluster with thousands of nodes. To emulate the I/O

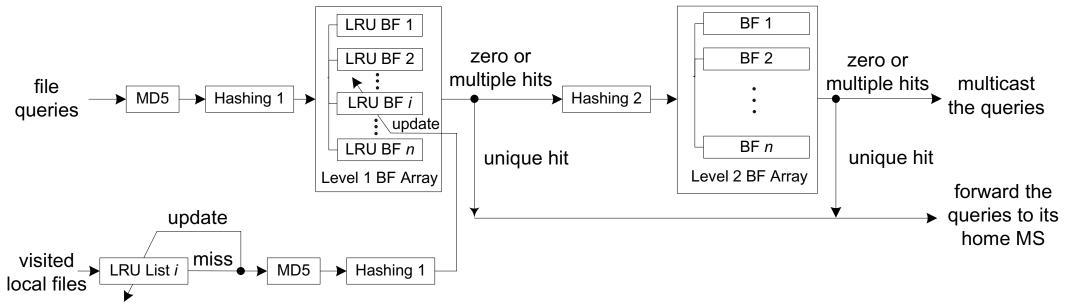

Figure 4. Scheme of hierarchical Bloom filter array on the metadata server node i.

Table 1. Comparisons of the original RES trace fragment and two scaled-up ones.

	Original	TIF = 50	TIF =100
Hosts	13	650	1300
Active files	4212	0.24 million	0.42 million
New files	301	14192	30103
Requests	0.014 million	6.9 million	14 million
Total files	0.66 million	3.3 million	66 million

Table 2. Comparisons of the original HP traces with a scaled-up one.

	Original	TIF = 40
Active users	18	710
Active files	0.057 million	2.26 million
New files	0.004 million	0.16 million
Requests	0.47 million	19 million
Total files	4.0 million	160.0 million

behaviors of such a large system and facilitate a meaningful simulation, we artificially scale up the workload presented in the RES trace collected at University of California Berkeley in 1997 and the HP file system trace collected at the HP Lab in December 2001.

Throughout January 1997, the RES trace [11] was collected on a cluster of 13 machines used by an academic research group consisting of 50 users. The HP file system trace [23] is a 10-day trace of all file system accesses to several disk arrays with a total of 500 GB of storage. These arrays were attached to a 4-way HP-UX time-sharing server and were used by 236 users. Since both the RES and HP traces collected all I/O requests at the file system level, any requests not related to metadata operations, such as read, write, and execution, are filtered out in our simulation.

To scale up the workload collected in these environments to emulate the workload in a large cluster with thousands of nodes, we divided each daily trace collected from 8:00am to 16:00pm, which were usually the busiest period during a day, into four fragments, with each fragment including two hours of I/O accesses. The time stamps of all events in each fragment are equally shifted so that this fragment starts at time instant zero. Replaying multiple time-shifted fragments simultaneously increases the I/O arrival rate while keeping a similar histogram of file system calls. In addition, the number of files stored and the number of files actively visited were scaled up proportionally by adding the date and fragment number as a prefix path to all filenames. We believe that replaying a large number of processed fragments together can emulate the workload of a large cluster since I/O requests are self-similar in nature at both the disk-level [16] and file level [17]. Note that the number of fragments replayed concurrently is referred to as Trace Intensifying Factor (TIF) throughout the rest of this paper. The characteristics of the original traces and their scaled-up ones are summarized in Table 1 and Table 2.

5.2 Trace Driven Simulation

We have developed a trace-driven simulator to emulate the behaviors of the metadata management system on the metadata servers. Some trace events that are not directly related to the metadata are filtered out in the simulation. For example, since the metadata is usually accessed through the system calls such as open, close, and stat, the data read and write events do not retrieve or modify the relevant metadata in a typical file system and thus they are skipped in the simulation.

6 Performance Evaluation

We simulate the metadata servers using the two traces introduced previously and measure the performance in terms of hit rates, as well as the memory and network overhead. Since the decentralized schemes of table-based mapping and modulus-based hashing are simple and straightforward and their performances were already discussed quali-

Table 3. Impact of the propagation thresholds on the hit rate in PBA.

Thresholds (%)	100	10	0.001	0.00001
Hit Rate(%)	81.110	81.112	81.17	82.417

tatively, the simulation study in this paper will be focused on the schemes of PBA, HBA and pure LRU BF to obtain quantitative comparisons and conclusions.

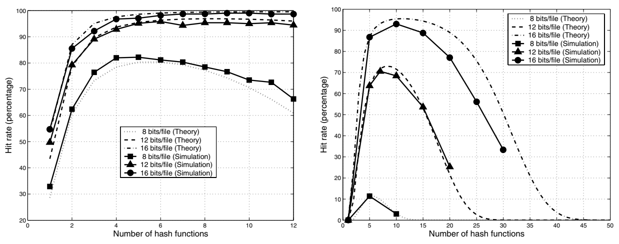

Figure 5. Comparisons the theoretical and simulation results of the hit rates of PBA in a cluster with 10 (left figure) and 100 (right figure) metadata servers (TIF = 100).

6.1 RES Traces

6.1.1 Pure Bloom Filter Array Approach

Figure 5 depicts the relationships, obtained by theoretical analysis and simulation, between the hit rates and the computation cost in terms of the number of hash functions used in the pure Bloom filter array (PBA) approach. In these simulations, 100 trace fragments were replayed simultaneously in a cluster with 10 and 100 metadata servers, respectively, and the Bloom filters used different combinations of the bit/file ratio and the number of hash functions. The PBA approach achieves its best hit rate when the number of hash functions optimizes a single Bloom filter. In the simulations presented in the rest of this paper, the number of hash functions is always kept at a value that optimizes the hit rate for a given bit/file ratio. The close agreement between the theoretical and simulation results lends more confidence and credence to our theoretical analysis and simulation results. More importantly, these experiments show that to maintain a high hit rate in a large cluster with 100 or more metadata servers, a large bit/file ratio, such as 16 bits/file, becomes necessary.

Table 3 shows the impact of the propagation threshold, the percentage of bits in a Bloom filter that must be changed before updating its copies in other metadata servers, on the hit rates in the scenario of 10 metadata servers and a bit/file ratio of 8. With the decrease of the threshold, the hit rate increases slightly. This unexpected low sensitivity of hit rate to threshold is due to the fact that the frequency of file renaming, creation or deletion is very low in the RES trace. We might underestimate the impact of the propagation threshold since the events of directory renaming cannot be fully and truthfully simulated for the given trace. The original file or directory names in the RES trace are hashed to a single level of namespace to protect the privacy and thus the hierarchical directory tree can not be reconstructed from the trace. Hence it is infeasible to truly simulate a directory renaming.

6.1.2 Hierarchical Bloom Filter Array Approach

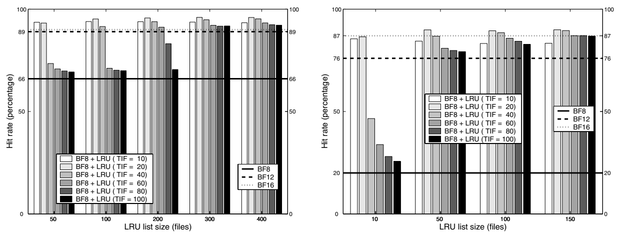

Figure 6. Comparisons of hit rates of HBA under various LRU size and TIFs in RES traces in a cluster with 10 (left figure) and 100 (right figure) metadata servers.

Figure 6 shows the hit rate of HBA with different sizes of LRU lists in a cluster with 10 and 100 metadata servers, respectively, when the TIF increases gradually from 10 to 100. In HBA, the two levels of Bloom filter arrays adopt different bit/file ratios, giving rise to different accuracies. While the second level Bloom filer array, which stores the distribution information of all files, employs a bit/file ratio of 8, the LRU BF in the first level adopts a bit/file ratio of 20. The bars in these figures represent the average hit rate of all metadata servers in the HBA approach. For the convenience of comparisons, the optimal hit rates in PBA with different bit/file ratios are also given in these figures and are shown as horizontal lines. In HBA, a small LRU can significantly improve the overall hit rates. The LRU lists with sizes of 300, 100 and 50 file entries in the clusters with 10 and 100 metadata servers, respectively, can boost the hit rates of HBA with 8 bits/file to, or higher than those of PBA of the same configurations but with 16 bits/file. In the real applications of HBA, the size of a LRU list can adaptively increase from some small initial value until a satisfying hit rate is achieved.

Table 4 gives the relative storage space overhead of various HBA and PBA configurations, normalized to that of PBA with 8 bits/file. On each metadata server, the extra

Table 4. Relative space overhead normalized to PBA with a ratio of 8.

Server #	PBA12	PBA16	HBA (LRU size)
10	1.5	2.0	1.0001136 (300 entries)
50	1.5	2.0	1.0001894 (100 entries)
100	1.5	2.0	1.0001894 (50 entries)

overhead of HBA introduced by a LRU list and a LRU Bloom filter is only a negligible portion of the space requirement of the PBA 8. This is because that millions of files are stored in the Bloom filter array in PBA, but only hundreds or thousands of files are stored in the LRU list and LRU Bloom filter. A HBA that achieves the same hit rate as a PBA with 16 bits/file requires only 50% of the space required by that PBA.

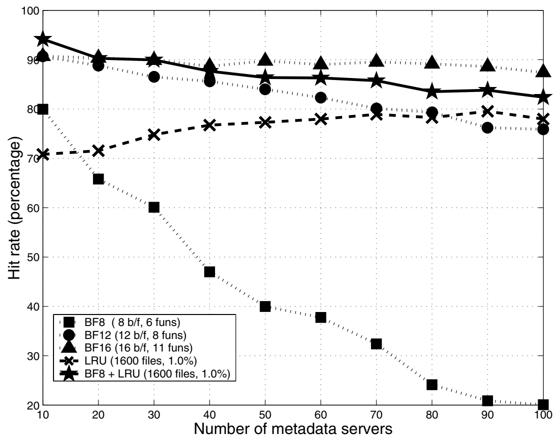

Figure 7. Hit rate comparisons between LRU BFs, HBA with a ratio of 8 and PBA with ratios of 8, 12 and 16 under different number of metadata servers (TIF = 40).

7 HP File System Traces

Figure 7 shows the hit rates of the PBA, a LRU list and the HBA when the number of metadata servers changes from 10 to 100 with a step of 10. The HBA combines the LRU list with a size of 1600 entries and a Bloom filter array with a bit/file ratio of 8. In these experiments, 40 trace fragments are replayed simultaneously and there are a total of 710 active users in the traces. When the number of metadata severs increases, the load on each metadata server decreases accordingly, thus slightly increasing the hit rate of LRU lists. In the experiments of less than 30 metadata servers, the hit rate of HBA is slightly better that of PBA

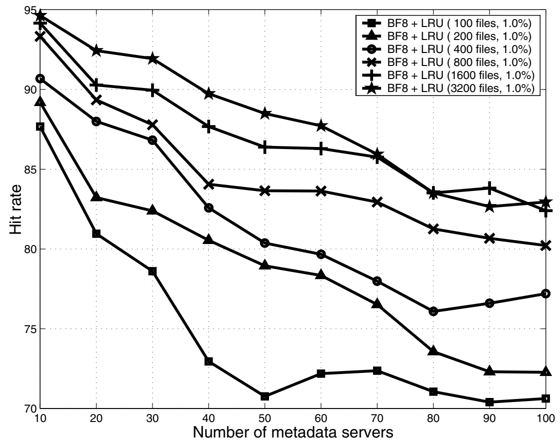

Figure 8. Hit rate comparisons of HBA with different LRU sizes under various number of metadata servers (TIF = 40).

Table 5. Relative space overhead normalized to PBA with a ratio of 8 in HP traces.

Server #	PBA 8	PBA 12	PBA 16	HBA
20	1.0	1.5	2.0	1.0002
40	1.0	1.5	2.0	1.0004
60	1.0	1.5	2.0	1.0006
80	1.0	1.5	2.0	1.0008
100	1.0	1.5	2.0	1.0010

with a bit/file ratio of 16. Although HBA is around $1-5.7\%$ worse than PBA with a bit/file ratio of 16 when the number of metadata servers increases from 30 to 100, the hit rate is still $1.5 - 9.9\%$ better than PBA with a bit/file ratio of 12 and $17.7 - 330\%$ better than PBA with a bit/file ratio of 8.

The impact of the LRU size on the overall hit rate of HBA is presented in Figure 8. It is shown that the benefit of increasing the LRU size is significant initially but diminishes gradually. Doubling the LRU size from 1600 to 3200 only results in up to 2% improvement in the hit rate. As indicated previously, in the real implementations of HBA, the size of LRU can be dynamically determined by gradually increasing from some initial value until a predefined hit rate goal is satisfied.

Table 5 presents the relative memory requirement normalized to the PBA with a bit/file ratio of 8 when the number of metadata servers changes from 10 to 100. The extra memory overhead introduced in the HBA by the LRU and LRU BF is up to 0.1% and only takes tens of KBs.

There is a clear tradeoff between the network traffic overhead and the hit rate in HBF. With a smaller propagation threshold, the LRU BFs are updated more frequently so that

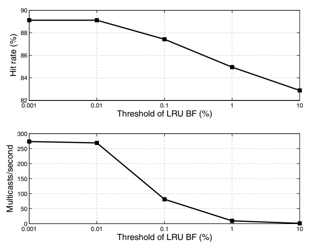

Figure 9. Tradeoff between hit rate and network overhead (50 metadata servers; 1600 entries in a LRU; TIF = 40;).

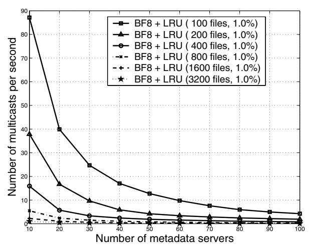

Figure 10. Network overhead of HBA with different LRU sizes (TIF = 40, LRU BF Threshold = 1%).

the likelihood of having a hit in a LRU BF is increased; but the updating traffic takes away some network bandwidth. Figure 9 shows the relationship between the hit rate and the number of multicast messages per second in the entire cluster when the propagating threshold increases from 0.001% to 100%. A threshold of 1% is found to have a good balance of this tradeoff. Figure 10 gives the network traffic under this threshold when both the number of metadata servers and the size of a LRU list changes. When the size of an LRU is larger than 1600, the total network traffic overhead introduced by HBA in most cases of metadata configurations are less than 1 multicast per second. We believe that this overhead is marginal in a modern network.

8 Conclusion

This paper first analyzed the efficiency of using a pure Bloom filter array (PBA) scheme to represent the metadata distribution of all files and accomplish the metadata distribution and management in cluster-based storage systems with thousands of nodes. Both theoretical analytical and simulation results indicated that this approach did not scale well with the increase of the number of metadata servers and have very large memory overhead when the number of files is large.

By exploiting temporal access locality of file access patterns, this paper then proposed a hierarchical scheme, HBA, that maintains two levels of Bloom filter arrays, with the one at the first level succinctly representing the metadata location of most recently visited files on each metadata server while the one at the second level maintaining metadata distribution information of all files with lower accuracy in favor of memory efficiency. The level 1 array has small size but a high accuracy and greatly compensates for the low efficiency of metadata distribution and significantly reduces the memory requirement of the level 2 array that follows the pure Bloom filter array approach. Our extensive trace-driven simulations showed that the HBA scheme can achieve an efficacy comparable to that of PBA, but at only 50% of memory cost and slightly higher network traffic overhead (multicast). On the other hand, HBA incurred much less network traffic overhead (multicast) than the pure LRU BF approach. Moreover, simulation results show that the network traffic overhead introduced by HBA is minute in modern fast networks.

Compared with other existing solutions to decentralizing the metadata management, the hierarchical scheme retains much of their advantages while avoiding their disadvantages. It not only reduces the memory overhead, but also balances the metadata management workload, allows a fully associative placement of metadata of files, requires no metadata migration during file or directory naming and node additions or deletions. Our immediate future work is to implement this scheme in a real system and evaluate the efficiency in a production environment.

9 Acknowledgement

This work is supported by an NSF Grant (EPS-0091900), a Nebraska University Foundation Grant (26-0511-0019), a 2003 Layman Fund and an Academic Priority Grant of University of Nebraska - Lincoln (UNL). Work was completed using the Research Computing Facility at UNL. We greatly appreciate Dong Li for developing preliminary trace transformation codes and valuable comments and discussions. We thank the HP labs and the University of California Berkeley for providing us the file system traces.

References

[1] T. Anderson, M. Dahlin, J. Neefe, D. Pat-terson, D. Roselli, and R. Wang. Serverless network file systems. In *In Proceedings of the 15th Symposium on Operating System Principles. ACM*, pages 109–126, Copper Mountain Resort, Colorado, December 1995.

[2] B. H. Bloom. Space/time trade-offs in hash coding with allowable errors. *Communications of the ACM*, 13(7):422–426, 1970.

[3] N. J. Boden, D. Cohen, R. E. Felderman, A. E. Kulawik, C. L. Seitz, J. N. Seizovic, and W.-K. Su. Myrinet: A gigabit-per-second local area network. *IEEE Micro*, 15(1):29–36, 1995.

[4] P. Braam. Lustre white paper. available at `http://www.lustre.org/docs/whitepaper.pdf`, Dec. 2003.

[5] S. A. Brandt, L. Xue, E. L. Miller, and D. D. E. Long. Efficient metadata management in large distributed file systems. In *20th IEEE / 11th NASA Goddard Conference on Mass Storage Systems and Technologies (MSST 2003)*, pages 290–298, San Diego, CA, Apr. 2003.

[6] A. Broder and M. Mitzenmacher. Network applications of bloom filters: A survey. In *Proceedings of 40th Annual Allerton Conference on Communication, Control and Computing*, Monticello, Illinois, USA, Oct. 2002.

[7] P. H. Carns, W. B. Ligon III, R. B. Ross, and R. Thakur. PVFS: A parallel file system for Linux clusters. In *Proceedings of the 4th Annual Linux Showcase and Conference*, pages 317–327, Atlanta, GA, 2000. USENIX Association. Best Paper Award.

[8] D. H. Carrere. Linux local and wide area network adapter support: (from 10 mbps to gigabit ethernet, token ring, frame relay, and slow packet). *Int. J. Netw. Manag.*, 10(2):103–112, 2000.

[9] V. Cate and T. Gross. Combining the concepts of compression and caching for a two-level file system. In *Proceedings Architectural Support for Programming Languages and Operating Systems (ASPLOS)*, pages 200–211, Santa Clara, CA, Apr. 1991.

[10] S. Dharmapurikar, P. Krishnamurthy, and D. E. Taylor. Longest prefix matching using bloom filters. In *Proceedings of ACM Special Interest Group on Data Communications (SIGCOM'03)*, Aug. 2003.

[11] J. R. L. Drew Roselli and T. E. Anderson. A comparison of file system workloads. In *Proceedings of the Annual USENIX Technical Conference*, San Diego, California, June 2000.

[12] C. Eddington. Infinibridge: An infiniband channel adapter with integrated switch. *IEEE Micro*, 22(2):48–56, 2002.

[13] L. Fan, P. Cao, J. Almeida, and A. Z. Broder. Summary cache: A scalable wide-area web cache sharing protocol. *IEEE/ACM Transactions on Networking*, 8(3):281–293, 2000.

[14] R. Floyd. Short-term file reference patterns in a UNIX environment. Technical Report TR-177, Computer Science Department, University of Rochester, Rochester, NY, Mar. 1986.

[15] S. Ghemawat, H. Gobioff, and S.-T. Leung. The Google file system. In *Proceedings of the nineteenth ACM Symposium on Operating Systems Principles*, pages 29–43, 2003.

[16] M. E. Gomez and V. Santonja. Analysis of self-similarity in I/O workload using structural modeling. In *Proceedings of the 7th International Symposium on Modeling, Analysis and Simulation of Computer and Telecommunication Systems*, page 234. IEEE Computer Society, 1999.

[17] S. D. Gribble, G. S. Manku, D. Roselli, E. A. Brewer, T. J. Gibson, and E. L. Miller. Self-similarity in file systems. In *Proceedings of the 1998 ACM SIGMETRICS joint international conference on Measurement and modeling of computer systems*, pages 141–150, 1998.

[18] W. L. III and R. Ross. Server-side scheduling in cluster parallel I/O systems. *Calculateurs Paralleles Journal, Special Issue on Parallel I/O for Cluster Computing*, Oct. 2001.

[19] M. K. McKusick, W. N. Joy, S. J. Leffler, and R. S. Fabry. A fast file system for unix. *ACM Trans. Comput. Syst.*, 2(3):181–197, 1984.

[20] J. H. Morris, M. Satyanarayanan, M. H. Conner, J. H. Howard, D. S. Rosenthal, and F. D. Smith. Andrew: a distributed personal computing environment. *Communications of the ACM*, 29(3):184–201, 1986.

[21] B. Pawlowski, C. Juszczak, P. Staubach, C. Smith, D. Lebel, and D. Hitz. NFS version 3: Design and implementation. In *Proceedings of the Summer 1994 USENIX Technical Conference*, pages 137–151, 1994.

[22] M. V. Ramakrishna. Practical performance of bloom filters and parallel free-text searching. *Communications of the ACM*, 32(10):1237–1239, 1989.

[23] E. Riedel, M. Kallahalla, and R. Swaminathan. A framework for evaluating storage system security. In *Proceedings of the USENIX Conference on File and Storage Technology*, pages 15–30, Mar. 2002.

[24] M. Rosenblum and J. K. Ousterhout. The design and implementation of a log-structured file system. In *Proceedings of the thirteenth ACM symposium on Operating systems principles*, pages 1–15, 1991.

[25] M. Satyanarayanan, J. Kistler, P. Kumar, M. Okasaki, E. Siegel, and D. Streere. Coda: A highly available file system for distributed workstation environments. *IEEE Transactions on Computers*, 39(4), April 1990.

[26] C. Staelin. *High performance file system design*. PhD thesis, Department of Computer Science, Princeton University, Oct. 1991.

[27] H. Tang and T. Yang. An efficient data location protocol for self-organizing storage clusters. In *Proceedings of ACM/IEEE SuperComputing (SC 03)*, Phoenix, AZ, Nov. 2003.

[28] M. Vilayannur, A. Sivasubramaniam, M. Kandemir, R. Thakur, and R. Ross. Discretionary caching for I/O on clusters. In *Proceedings of the Third IEEE/ACM International Symposium on Cluster Computing and the Grid*, pages 96–103, Tokyo, Japan, May 2003.

[29] J. Wu, P. Wyckoff, and D. Pandac. PVFS over InfiniBand: Design and performance evaluation. In *Proceedings of International Conference on Parallel Processing (ICPP)*, pages 125–132, Oct. 2003.

[30] Y. Zhu, H. Jiang, X. Qin, D. Feng, and D. Swanson. Design, implementation, and performance evalaution of a cost-effective fault-tolerant parallel virtual file system. In *The International Workshop on Storage Network Architecture and Parallel I/Os, in conjunction with The IEEE Twelfth International Conference on Parallel Architectures and Compilation Techniques (PACT)*, New Orleans, LA, Sept. 2003.

[31] Y. Zhu, H. Jiang, X. Qin, D. Feng, and D. Swanson. Improved read performance in a Cost-Effective, Fault-Tolerant Parallel Virtual File System (CEFT-PVFS). In *Proceeding of IEEE/ACM Workshop on Parallel I/O in Cluster Computing and Computational Grids, in conjunction with IEEE International Symposium on Cluster Computing and the Grid (CCGRID)*, pages 730–735, Tokyo, Japan, May 2003.

[32] Y. Zhu, H. Jiang, X. Qin, D. Feng, and D. Swanson. Scheduling for improved write performance in a cost-effective, fault-tolerant parallel virtual file system (CEFT-PVFS). In *Proceedings of The 4th LCI International Conference on Linux Clusters*, San Jose, California, June 2003.

[33] Y. Zhu, H. Jiang, X. Qin, and D. Swanson. A case study of parallel I/O for biological sequence search on linux clusters. In *Proceedings of IEEE International Cluster Computing*, pages 308–315, Hong Kong, China, Dec. 2003.