HBA: Distributed Metadata Management for Large Cluster-Based Storage Systems

Yifeng Zhu, *Member*, *IEEE*, Hong Jiang, *Member*, *IEEE*,
Jun Wang, *Member*, *IEEE*, and Feng Xian, *Student Member*, *IEEE*

Abstract—An efficient and distributed scheme for file mapping or file lookup is critical in decentralizing metadata management within a group of metadata servers. This paper presents a novel technique called Hierarchical Bloom Filter Arrays (HBA) to map filenames to the metadata servers holding their metadata. Two levels of probabilistic arrays, namely, the Bloom filter arrays with different levels of 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, whereas the other array, with higher accuracy, caches partial distribution information and exploits the temporal locality of file access patterns. Both arrays are replicated to all metadata servers to support fast local lookups. We evaluate HBA through extensive trace-driven simulations and implementation in Linux. Simulation results show our HBA design to be highly effective and efficient in improving the performance and scalability of file systems in clusters with 1,000 to 10,000 nodes (or superclusters) and with the amount of data in the petabyte scale or higher. Our implementation indicates that HBA can reduce the metadata operation time of a single-metadata-server architecture by a factor of up to 43.9 when the system is configured with 16 metadata servers.

Index Terms—Distributed file systems, file system management, metadata management, Bloom filter.

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 [1], [2], [3], [4], [5], [6]. In the recent years, the bandwidth of these networks has been increased by two orders of magnitude [7], [8], [9], which greatly narrows the performance gap between them and the dedicated networks used in commercial storage systems. This significant improvement offers an appealing opportunity to provide cost-effective high-performance storage services by aggregating 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 numbers of nodes are expected in the near future.

- *Y. Zhu is with the Department of Electrical and Computer Engineering, University of Maine, 5708 Barrows Hall, Orono, ME 04473. E-mail: zhu@eece.maine.edu.*
- *H. Jiang and F. Xian are with the Department of Computer Science and Engineering, University of Nebraska-Lincoln, 268 Avery Hall, Lincoln, NE 68588. E-mail: {jiang, fxian}@cse.unl.edu.*
- *J. Wang is with the School of Electrical Engineering and Computer Science, University of Central Florida, Harris Center ENGRIII-320, Orlando, FL 32816. E-mail: jwang@eecs.ucf.edu.*

Manuscript received 28 June 2006; revised 2 Mar. 2007; accepted 20 June 2007; published online 12 Oct. 2007.
Recommended for acceptance by A. Pietracaprina.
For information on obtaining reprints of this article, please send e-mail to: tpds@computer.org, and reference IEEECS Log Number TPDS-0173-0606.
Digital Object Identifier no. 10.1109/TPDS.2007.70788.

Since all I/O requests can be classified into two categories, that is, user data requests and 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 metadata is stored separately on different nodes away from user data. Although previous work on cluster-based storage mainly focuses on optimizing the scalability and efficiency of user data accesses by using a RAID-style striping [3], [10], caching [11], scheduling [12], [13], and networking [14], little attention has been drawn to the scalability of metadata management.

Yet, the efficiency of 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 from node failures, and maintains consistency between user data and file metadata. A study on the file system traces collected from different environments over a course of several months shows that requests targeting metadata can account for up to 83 percent of the total number of I/O requests [15]. Under such skewed loads 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 (MS) can be severely limited.

This paper proposes a novel scheme, called *Hierarchical Bloom Filter Arrays* (HBA), to evenly distribute the tasks of metadata management to a group of MSs. A Bloom filter (BF) is a succinct data structure for probabilistic membership query. Our analysis led us to the conclusion

that a straightforward adoption of BFs is impractical due to the memory space overhead when the number of files is very large. By exploiting the temporal locality of metadata accesses, we propose the use of two levels of BF arrays. At the first level, a small array with a high accuracy is used to capture the destination MS information of frequently accessed files to keep high management efficiency while reducing the memory overhead. At the second level, an array with lower accuracy in favor of memory efficiency is used to maintain the destination metadata information of all files. Full pathnames are used for BF hashing. Both arrays are replicated to all MSs to facilitate fast local lookups. As the system evolves, as a result of events such as file creation and deletion, workload variations, and changes in server configurations, a BF is updated locally, and the changes are propagated to its remote replicas periodically to improve the lookup accuracy. Our trace-driven simulations show that our HBA design has a strong scalability in decentralizing metadata management. Our preliminary results are published in [16]. This paper incorporates our experimental results based on a real implementation in Linux.

This paper has the following technical contributions:

- It analyzes the performance of the pure BF array (PBA) 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 trade-off between the efficiency of metadata distribution and management, and the memory and network overhead.
- It compares both the HBA and PBA schemes using two artificially scaled-up large file system traces that emulate file systems of up to 1,300 nodes and 710 active users.
- HBA attempts to optimize the trade-off between the efficiency and the network and memory overhead. To achieve high metadata lookup 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 of lower accuracy (to significantly reduce memory overhead) with a pure LRU scheme of lower update frequency to achieve a good trade-off between the efficiency and the memory and network overhead. As a result, HBA achieves a high efficiency without significant memory or network overhead.

The rest of this paper is organized as follows: Section 2 outlines the existing approaches to decentralizing 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 and performance evaluation are presented in Sections 5 and 6, respectively. Section 7 describes our prototype implementation in Linux and discusses the experimental results. Section 8 concludes this paper.

2 Related Work and Comparison of Decentralization Schemes

Many cluster-based storage systems employ centralized metadata management. Experiments in GFS show that a single MS is not a performance bottleneck in a storage cluster with 100 nodes under a read-only Google searching workload. PVFS [3], which is a RAID-0-style parallel file system, also uses a single MS design to provide a clusterwide shared namespace. As data throughput is the most important objective of PVFS, some expensive but indispensable functions such as the concurrent control between data and metadata are not fully designed and implemented. In CEFT [6], [10], [13], [17], which is an extension of PVFS to incorporate a RAID-10-style fault tolerance and parallel I/O scheduling, the MS synchronizes concurrent updates, which can limit the overall throughput under the workload of intensive concurrent metadata updates. In Lustre [1], some low-level metadata management tasks are offloaded from the MS to object storage devices, and ongoing efforts are being made to decentralize metadata management to further improve the scalability.

Some other systems have addressed metadata scalability in their designs. For example, GPFS [18] uses dynamically elected "metanodes" to manage file metadata. The election is coordinated by a centralized token server. OceanStore [19], which is designed for LAN-based networked storage systems, scales the data location scheme by using an array of BFs, in which the ith BF is the union of all the BFs for all of the nodes within i hops. The requests are routed to their destinations by following the path with the maximum probability. Panasas ActiveScale [20] not only uses object storage devices to offload some metadata management tasks but also scales up the metadata services by using a group of directory blades. Our target systems differ from the three systems above. Although GPFS and Panasas ActiveScale need to use their specially designed commercial hardware, our target systems only consist of commodity components. Our system is also different from OceanStore in that the latter focuses on geographically distributed storage nodes, whereas our design targets cluster-based storage systems, where all nodes are only one hop away.

The following summarizes other research projects in scaling metadata management, including table-based mapping, hash-based mapping, static tree partitioning, and dynamic tree partitioning.

2.1 Table-Based Mapping

Globally replicating mapping tables is one approach to decentralizing metadata management. There is a salient trade-off between the space requirement and the granularity and flexibility of distribution. A fine-grained table allows more flexibility in metadata placement. In an extreme case, if the table records the home MS for each individual file, then the metadata of a file can be placed on any MS. 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

TABLE 1
Comparison of HBA with Existing Decentralization Schemes

	Hashing based Mapping	Table based Mapping	Static Tree Partition	Dynamic Tree Partition	HBA
Example Systems	Lustre [1], Vesta [30], InterMezzo [31]	xFS [32]	NFS [22], AFS [23], Coda [24], Sprite [33]	OBFS [29]	HBA
Load Balance	Yes	No	No	Need load monitor	Yes
Migration Cost	Large	0	0	Large	0
Lookup Time	O(1)	$O(\log n)$	Slow	$O(\log d)$	O(1)
Memory Overhead	0	$O(n)$	$O(1)$	$O(d)$	$O(n)$
Directory Operations	Slow	Medium	$O(1)$	$O(1)$	Fast

n and d are the total number of files and partitioned subdirectories, respectively.

as 1.8 Gbytes of memory space to store such a table with 10^8 entries when 16 bytes are used for a filename and 2 bytes for an MS ID. In addition, searching for an entry in such a huge table consumes a large number of precious CPU cycles. To reduce the memory space overhead, xFS [21] proposes a coarse-grained table that maps a group of files to an MS. To keep a good trade-off, it is suggested that in xFS, the number of entries in a table should be an order of magnitude larger than the total number of MSs.

2.2 Hashing-Based Mapping

Modulus-based hashing is another 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 number of MSs. 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 MSs. However, a serious problem arises when an upper directory is renamed or the total number of MSs changes: the hashing mapping needs to be reimplemented, and this requires all affected metadata to be migrated among MSs. Although the size of the metadata of a file is small, a large number of files may be involved. In particular, the metadata of all files has to be relocated if an MS joins or leaves. This could lead to both disk and network traffic surges and cause serious performance degradation. LazyHybrid [2] is proposed to reduce the impact of metadata migration by updating lazily and also incorporating a small table that maps disjoint hash ranges to MS IDs. The migration overhead, however, can still overweigh the benefits of load balancing in a heavily loaded system.

2.3 Static Tree Partitioning

Static namespace partition is a simple way of distributing metadata operations to a group of MSs. A common partition technique has been to divide the directory tree during the process of installing or mounting and to store the information at some well-known locations. Some distributed file systems such as NFS [22], AFS [23], and Coda [24] 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 [25], [26], [27], [28] and, thus, this partition scheme can lead to a highly imbalanced workload if files in some particular subdirectories become more popular than the others.

2.4 Dynamic Tree Partitioning

Weil et al. [29] observe the disadvantages of the static tree partition approach and propose to dynamically partition the namespace across a cluster of MSs in order to scale up the aggregate metadata throughput. The key design idea is that initially, the partition is performed by hashing directories near the root of the hierarchy, and when a server becomes heavily loaded, this busy server automatically migrates some subdirectories to other servers with less load. It also proposes prefix caching to efficiently utilize available RAM on all servers to further improve the performance. This approach has three major disadvantages. First, it assumes that there is an accurate load measurement scheme available on each server and all servers periodically exchange the load information. Second, when an MS joins or leaves due to failure or recovery, all directories need to be rehashed to reflect the change in the server infrastructure, which, in fact, generates a prohibitively high overhead in a petabyte-scale storage system. Third, when the hot spots of metadata operations shift as the system evolves, frequent metadata migration in order to remove these hot spots may impose a large overhead and offset the benefits of load balancing.

2.5 Comparison of Existing Schemes

Table 1 summarizes the existing state-of-the-art approaches to decentralizing metadata management and compares them with the HBA scheme, which will be detailed later in this paper. Each existing solution has its own advantages and disadvantages. The hashing-based mapping approach can balance metadata workloads and inherently has fast metadata lookup operations, but it has slow directory operations such as listing the directory contents and renaming directories. In addition, when the total number of MSs changes, rehashing all existing files generates a prohibitive migration overhead. The table-based mapping method does not require any metadata migration, but it fails to balance the load. Furthermore, a back-of-the-envelope calculation shows that it would take as much as 1.8 Gbytes of memory to store such a table with 100 million files. The static tree balance approach has zero migration overhead, small memory overhead, and fast directory

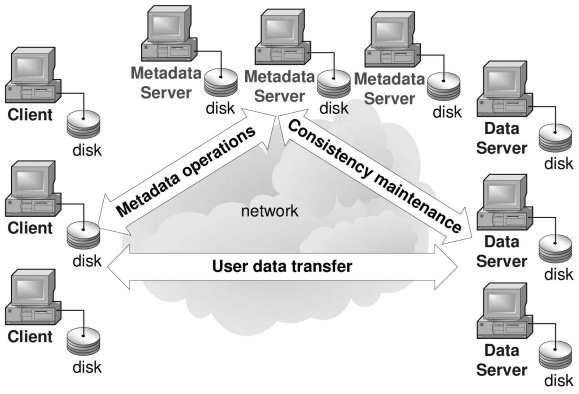


Fig. 1. Cluster-based storage architecture.

operation. However, it cannot balance the load, and it has a medium lookup time, since hot spots usually exist in this approach. Similar to the hashing-based mapping, dynamic tree partition has fast lookup operations and small memory overhead. However, this approach relies on load monitors to balance metadata workloads and thus incurs a large migration overhead. To combine their advantages and avoid their disadvantages, a novel approach, called HBA, is proposed in this paper to efficiently route metadata requests within a group of MSs. The detailed design of HBA will be presented later in this paper.

3 Architectural Considerations and Design Objectives

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, MSs, and data servers can be carried out by any node. A node may not be dedicated to a specific role. It can act in multiple roles simultaneously. Fig. 1 shows the architecture of a generic cluster targeted in this study.

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 of 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 computational power of a cluster. It should not become a performance bottleneck under high I/O access workloads. This requires the system to have low management overhead.
- *Zero metadata migration.* Although the size of 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 response to the file system's evolution such as renaming of files or directories, or topology changes involving server arrivals or departures, the computational overhead of checking whether a migration is needed and the network traffic overhead due to metadata migration may be prohibitively high, hence limiting the efficiency and scalability.
- *Balancing the load of metadata accesses.* The management is evenly shared among multiple MSs to best leverage the available throughput of these severs.
- *Flexibility of storing the metadata of a file on any MS.* 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 [34], [35]. 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 BF is a lossy but succinct and efficient data structure to represent a set $\mathcal{S}$, which processes the membership query, "Is x in $\mathcal{S}$?" for any given element x with a time complexity of $O(1)$. It was invented by Burton Bloom in 1970 [36] and has been widely used for Web caching [37], network routing [38], and prefix matching [39]. The storage requirement of a BF falls several orders of magnitude below the lower bounds of error-free encoding structures. This space efficiency is achieved at the cost of allowing a certain (typically nonzero) probability of *false positives* or *false hits*; that is, it may incorrectly return a "yes," although x is actually not in $\mathcal{S}$.

4.2 Hierarchical Bloom Filter Array Design

4.2.1 Pure Bloom Filter Array Approach

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

We denote this simple approach as *PBA*. PBA allows a flexible metadata placement, has no migration overhead, and balances metadata workloads. 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

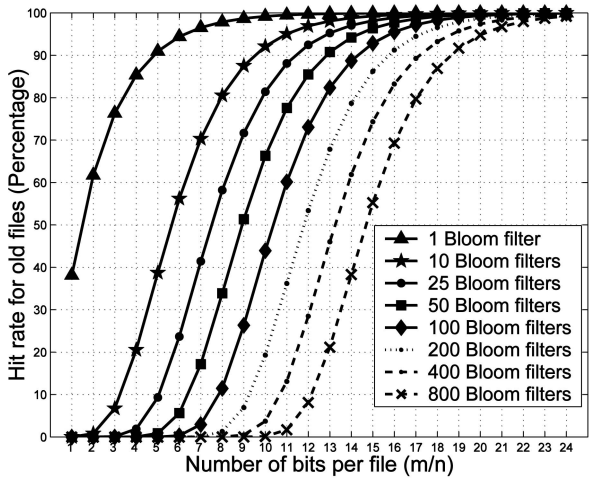


Fig. 2. Theoretical hit rates for existing files.

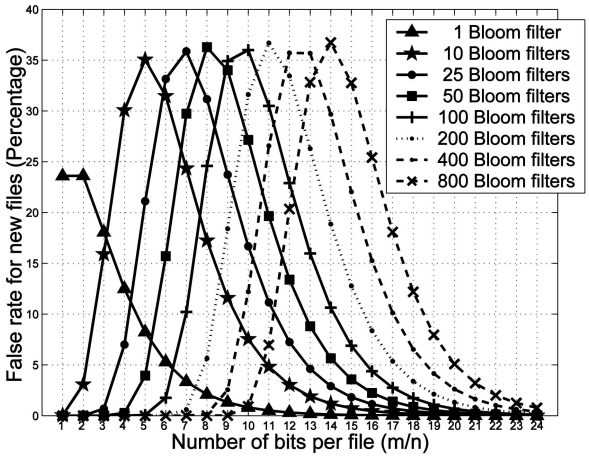


Fig. 3. Theoretical false-hit rates for new files.

it feasible to group metadata with strong locality together for prefetching, a technique that has been widely used in conventional file systems [34], [35]. During the evolution of file system and its cluster topology, not all metadata needs to migrate to new locations. When a file or directory is renamed, only the BFs associated with all the involved files or subdirectories need to be updated. Although a MS leaves or joins the system, a single associated BF is added or deleted from the Bloom arrays on all other MSs. Since each client randomly chooses a MS to look up for the home MS of a file, the query workload is balanced on all MSs.

The following theoretical analysis shows that the accuracy of PBA does not scale well when the number of MSs increases. When an existing file is searched against a group of BFs, a false-positive hit from any filter can lead to multiple hits and accordingly causes the search to fail. Assuming that all BFs are perfectly updated, the expected hit rate for an existing file is the probability that all BFs have no false-positive hits, given as follows:

$$hit_{oldfile} = (1-f)^{p-1} = \left(1-(0.6185)^{m/n}\right)^{p-1}, \qquad (1)$$

where m is the length of a BF in bits, n is the number of files that a single MS represents, p is the total number of MSs, and f is the optimal false rate of a single BF, as analyzed in [40]. Fig. 2 shows the relationship between $hit_{oldfile}$ and m/n under different numbers of MSs.

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

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

Given a constant p, $false_{newfile}$ reaches its maximum value $(1-\frac{1}{p})^{p-1}$ when $f=\frac{1}{p}$, that is, $m/n = 2.0792 \ln p$. 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 MSs is given in Fig. 3. This trend shows a special characteristic of a single BF array that is different from that of a single BF. Although in a BF, increasing the filter length always reduces its false-hit rate, *the false-hit rate of a BF 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 BF arrays.

While optimizing the trade-off between the space efficiency and the query accuracy, more weight is 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.

The above analysis shows that the accuracy of PBA, even when optimized for existing file and new file lookup, degrades quickly with the increase in the number of BFs, that is, the number of MSs. This leads to the major disadvantage of PBA. To achieve satisfactory hit rates, BFs with large sizes need to be used, thus increasing the memory space requirement on each MS. For example, if there are 200 MSs in a supercluster, 16 bits per file are required in each BF to maintain a hit rate of approximately 90 percent for old files and a false-hit rate of 10 percent for new files. If there are 500 million files stored in this cluster, the BF array would take around $16 \times 500\ \text{Mbits} = 1\ \text{Gbyte}$ of memory space on each MS. This memory requirement is underestimated, since in practice, the hit rates can be lower than the theoretical estimation. This implies that an even higher bit/file ratio needs to be employed. In a Web caching design system [37], a ratio of 32 bits per object is suggested.

4.2.2 Hierarchical Bloom Filter Array Design

To achieve a sufficiently high hit rate in the PBA described above, the high memory overhead may make this approach impractical. A large bit-per-file ratio needs to be employed in each BF to achieve a high hit rate when the number of MSs is large. In this section, we present a new design called HBA to optimize the trade-off between memory overhead and high lookup accuracy.

The novelty of HBA lies in its judicious exploitation of the fact that in a typical file system, a small portion of files absorb most of the I/O activities. Floyd [26] discovered that 66 percent of all files had not been accessed in over a month

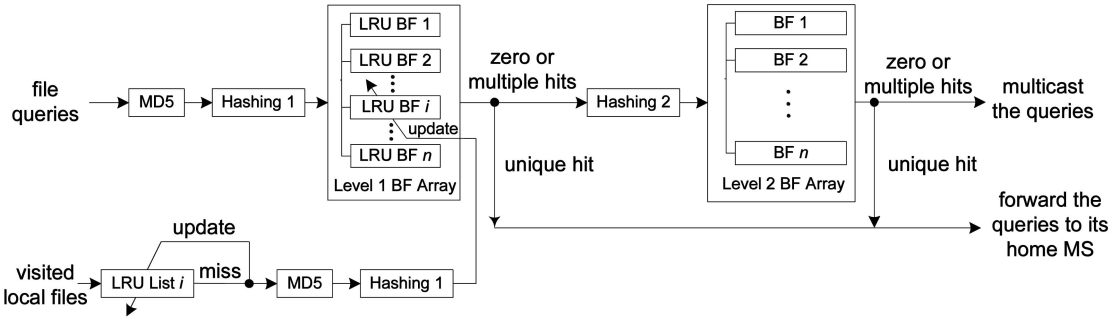


Fig. 4. Structure of HBA on the MS node i. n is the total number of MSs.

in a Unix environment, indicating that the entire I/O accesses were focused on at most 34 percent of the file system. Staelin [28] found that 0.1 percent of the total space used by the file system received 30 percent to 60 percent of the I/O activity. Cate and Gross [25] showed that most files in Unix file systems were inactive, and only 3.6 percent to 13 percent of the file system data was used in a given day, and only 0.2 percent to 3.6 percent of the I/O activity went to the least active 75 percent part of the file system. A recent study [5] on a file system trace collected in December 2000 from a medium-sized file server found that only 2.8 percent and 24.2 percent of files were accessed during a continuous course of 12 hours and 10 days, respectively.

Fig. 4 shows the structure of the HBA design on each MS, which includes two levels of BF arrays. In the design, each MS maintains a Least Recently Used (LRU) list that caches names of recently visited files whose metadata is stored locally on that MS. Each BF at the top level, called an *LRU BF*, represents all the files cached in the LRU list of the corresponding MS. Each LRU BF is globally replicated among all MSs. Whenever an overflow happens in the LRU list, an eviction based on the LRU replacement policy triggers both an addition operation and a deletion operation to its corresponding LRU BF. Only when the amount of changes made to a LRU BF has exceeded some threshold will the LRU BF be multicast to all the MSs to update all its replicas. Since the number of entries in LRU is relatively small, it is affordable to use a high bit/file ratio to achieve a low false-hit rate. In addition, the BFs in the lower level represent the metadata distributions of all MSs. Since the total number of files is typically very large, a low bit/file ratio is used to reduce the memory overhead. A miss in the top level array leads to a query to the lower level. An unsuccessful query in the lower level array will cause a broadcast to be issued to all the other metadata severs. Note 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 MS, and an acknowledgment across the network.

To perform a query into the BFs, filenames are transformed into digital indices 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 [41]. The MD5 approach is chosen because of its available fast implementation. The universal hash functions are employed to keep the independence of hash indices, a requirement for BFs 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 [2] 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 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.

Our HBA scheme can support the five objectives described in Section 3:

1. The two-layered BF arrays on each MS form an integral component to facilitate file lookup in a single shared namespace.
2. The simulation results presented in Section 6 indicate that HBA can achieve comparable lookup accuracy, with only 50 percent of the memory overhead of PBA. Our experimental results based on a real implementation with multiple MSs present superlinear speedup over a single MS, as will be described in detail in Section 7. This superlinear speedup, although complicated in its causes (including effects such as reduced working set and network contention as a result of decentralized management), is clearly related to the effectiveness of the decentralized metadata management.
3. When a file or directory is renamed or the MS configuration changes, no metadata migration is needed to maintain the lookup correctness. Instead, only local BFs are updated, and changes are propagated to their remote replicas if needed. To minimize the number of broadcasts while renaming an upper directory, the names of files and subdirectories (not including the files of subdirectories) are recorded as part of the metadata of their parent directory.
4. Random placement is chosen in HBA to initially place metadata on a server. When the total number of files are significantly larger than the number of MSs, the file query workload can be coarsely balanced among all MSs. When a new MS is added, our design is currently insufficient to offload a partial workload to the new server, and some future research work is needed.
5. Similar to PBA, HBA allows the metadata of a file to be placed on any MS without any restriction.

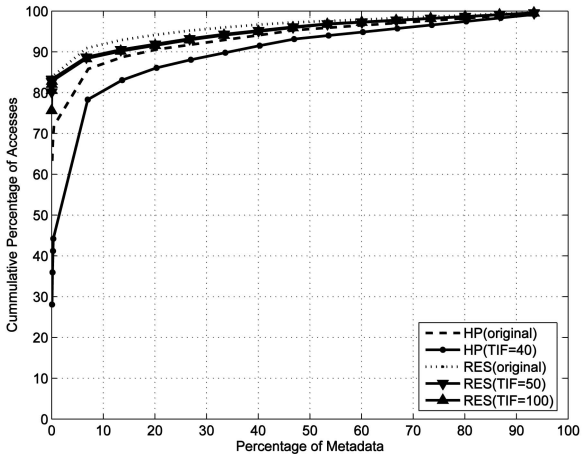


Fig. 5. Cumulative distributions of metadata access.

TABLE 2
Comparison 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 3
Comparison 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

5 Trace-Driven Simulation

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, let alone those also containing sufficient amount of metadata operations. Most available traces only focus on recording data read and write operations [42], [43]. To emulate the I/O behavior of such a large system and facilitate a meaningful simulation, we intentionally scale up the workload presented in the RES trace collected at the University of California, Berkeley, in 1997 and in the HP file system trace collected at the Hewlett-Packard Laboratories in December 2001.

Throughout January 1997, the RES trace [15] was collected on a cluster of 13 machines used by an academic research group consisting of 50 users. The HP file system trace [27] is a 10-day trace of all file system accesses to several disk arrays, with a total of 500 Gbytes of storage. These arrays were attached to a four-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 emulate the workload of a large cluster with thousands of nodes, we choose to scale up the workloads in the RES and HP traces available to us. The conventional approach to scale up workloads is to simply replay I/O traces at an accelerated rate by dividing the time stamps by a constant speedup factor. When a large number of users are running concurrently on shared file servers, this approach cannot appropriately reflect the increase in working set and user number. Accordingly, we choose to stress MSs by a combination of spatial scale-up and temporal scale-up in our simulation study. We decompose a trace into subtraces and add a subtrace number in each trace record in order to have disjoint group ID, user ID, and working directories. Specifically, we divide each daily trace collected from 8:00 a.m. to 4:00 p.m., which was usually the busiest period during the day, into four 2-hour subtraces. The timing relationships among the requests within a subtrace are preserved to faithfully maintain the semantic dependencies among trace records. These subtraces are replayed concurrently by setting the same start time. As a result, the metadata traffic can be spatially and temporally scaled up by a different factor, depending on the number of subtraces replayed simultaneously. Note that the combined trace maintains similar histogram of file system calls. The percentages of all system calls such as *open*, *close*, and *fstate* remain unchanged. The intensified HP and RES traces retain similar distributions in cumulative metadata accesses, as shown in Fig. 5. A point $(x; y)$ in the cumulative distribution curve indicates that x percentage of file or directory metadata receives y percentage of the total metadata accesses. The results show that 7 percent of metadata absorbs 78 percent to 92 percent of metadata traffic, which is consistent with the results of workload studies summarized in Section 4.2.2; that is, a small portion of files absorb most of the I/O activities. The number of subtraces replayed concurrently is denoted as the *Trace Intensifying Factor* (TIF). Tables 2 and 3 summarize the characteristics of the original and scaled-up traces.

We have developed a trace-driven simulator to emulate the behavior of the metadata management system on MSs. Some trace events that are not directly related to metadata are filtered out in the simulation. For example, since 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 MSs by using the two traces introduced in Section 5 and measure the performance in terms of hit rates and the memory and network overhead. Since the decentralized schemes of table-based mapping and modulus-based hashing are simple and straightforward and their performance was already discussed qualitatively, the simulation study in this paper will be focused on the schemes of PBA, HBA, and pure LRU BF to obtain quantitative comparison and conclusions.

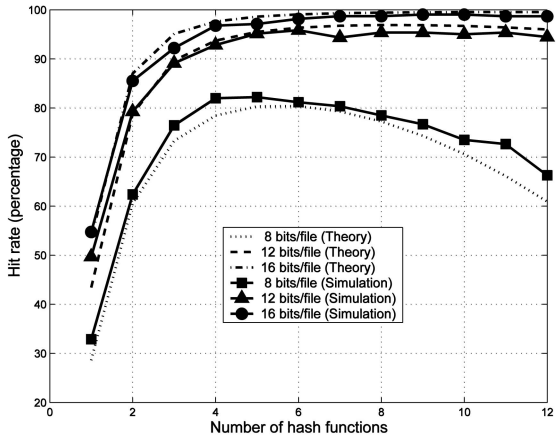


Fig. 6. Comparison of the theoretical and simulation results of the hit rates of PBA in a cluster with 10 MSs.

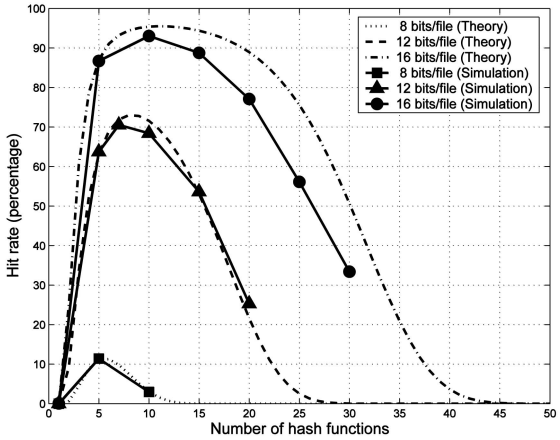


Fig. 7. Comparison of the theoretical and simulation results of the hit rates of PBA in a cluster with 100 MSs.

6.1 Research Workload Traces

6.1.1 Pure Bloom Filter Array Approach

Figs. 6 and 7 depict the relationships, as obtained by our theoretical analysis and simulation, between the hit rates and the computation cost in terms of the number of hash functions used in the PBA approach. In these simulations, 100 trace fragments were replayed simultaneously in a cluster with 10 and 100 MSs, respectively, and the BFs 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 BF. 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 MSs, a large bit/file ratio such as 16 bits/file becomes necessary.

Table 4 shows the impact of the propagation threshold, that is, the percentage of bits in a BF that must be changed before updating its replicas in other MSs, on the hit rates in the scenario of 10 MSs and a bit/file ratio of 8. With the decrease in 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 cannot be reconstructed from the trace. Hence, it is infeasible to truly simulate a directory renaming.

TABLE 4
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

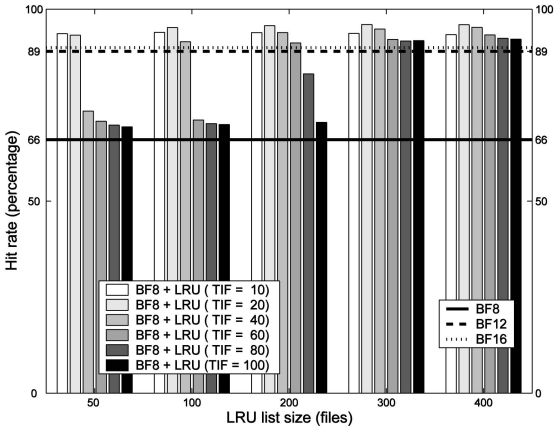


Fig. 8. Comparison of hit rates of HBA under various LRU sizes and TIFs in the RES traces in a cluster with 10 MSs.

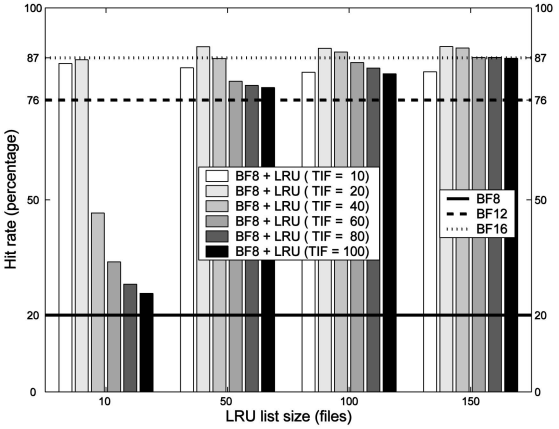


Fig. 9. Comparison of hit rates of HBA under various LRU sizes and TIFs in the RES traces in a cluster with 100 MSs.

6.1.2 Hierarchical Bloom Filter Array Approach

Figs. 8 and 9 show the hit rate of HBA with different sizes of LRU lists in a cluster with 10 and 100 MSs, respectively, when the TIF increases gradually from 10 to 100. In HBA, the two levels of BF arrays adopt different bit/file ratios, giving rise to different accuracies. Although the second-level BF 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 rates of all MSs in the HBA

TABLE 5
Relative Space Overhead Normalized to PBA with a Bits/File 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)

approach. For convenience of comparison, 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 MSs, respectively, can boost the hit rates of HBA with a bits/file ratio of 8 or higher than those of PBA of the same configurations but with a bits/file ratio of 16. In 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 5 gives the relative storage space overhead of various HBA and PBA configurations, normalized to that of PBA with a bits/file ratio of 8. On each MS, the extra overhead of HBA introduced by an LRU list and an LRU BF is only a negligible portion of the space requirement of its PBA counterpart. This is because millions of files are stored in the BF array in PBA, but only hundreds or thousands of files are stored in the LRU list and LRU BF. An HBA that achieves the same hit rate as a PBA with a bits/file ratio of 16 requires only 50 percent of the space required by PBA.

6.2 Hewlett-Packard Laboratories File System Traces

Fig. 10 shows the hit rates of PBA, LRU list, and HBA when the number of MSs changes from 10 to 100, with a step of 10. HBA combines an LRU list with a size of 1,600 entries and a BF 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 MS decreases accordingly, thus slightly increasing the hit rate

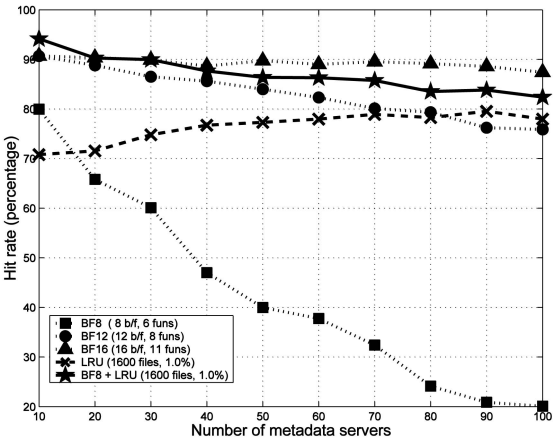


Fig. 10. Hit rate comparison between LRU BFs, HBA with a bits/file ratio of 8, and PBA with bits/file ratios of 8, 12, and 16 under different numbers of MSs $(\mathrm{TIF} = 40)$.

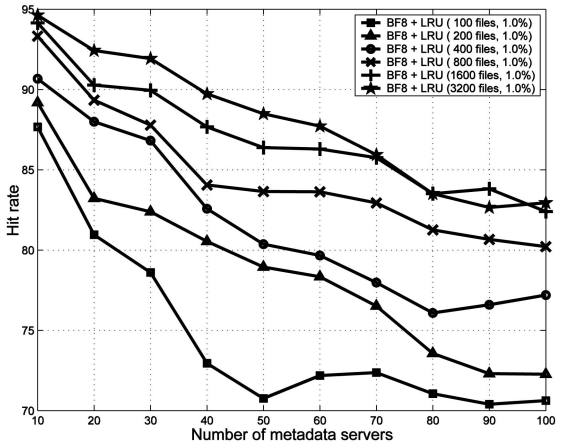


Fig. 11. Hit rate comparison of HBA with different LRU sizes under various number of MSs $(\mathrm{TIF} = 40)$.

of LRU lists. In the experiments with less than 30 MSs, the hit rate of HBA is slightly better than that of PBA with a bit/file ratio of 16. Although the hit rate of HBA is around 1 percent to 5.7 percent lower than that of PBA with a bit/file ratio of 16 when the number of MSs increases from 30 to 100, it is still 1.5 percent to 9.9 percent higher than that of PBA with a bit/file ratio of 12 and 17.7 percent to 330 percent higher 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 Fig. 11. It is shown that the benefit of increasing the LRU size is significant initially but diminishes gradually. Doubling the LRU size from 1,600 to 3,200 only results in up to 2 percent of improvement in the hit rate. As indicated previously, in 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 reached.

Table 6 presents the relative memory requirement normalized to PBA with a bit/file ratio of 8 when the number of MSs changes from 10 to 100. The extra memory overhead introduced in HBA by LRU and LRU BF is up to 0.1 percent and only takes tens of kilobytes.

There is a clear trade-off between the network traffic overhead and hit rate in HBA. With a smaller propagation threshold, LRU BFs are updated more frequently so that the likelihood of having a hit in an LRU BF is increased, but the updating traffic takes away some network bandwidth. Fig. 12 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 percent to 100 percent. A threshold of 1 percent is found to have a good

TABLE 6
Relative Space Overhead Normalized to PBA with a Ratio of 8 in the 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

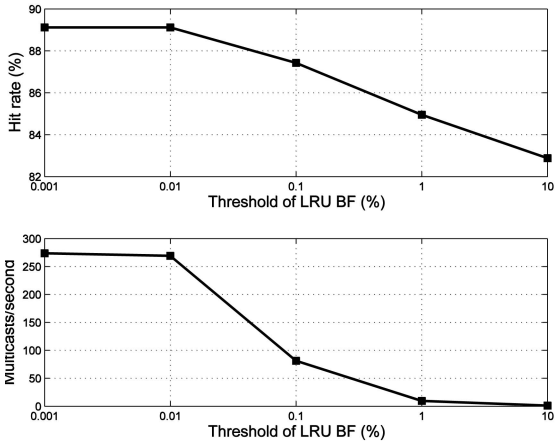


Fig. 12. Trade-off between hit rate and network overhead (50 MSs, 1,600 entries in LRU, and $\mathrm{TIF} = 40$).

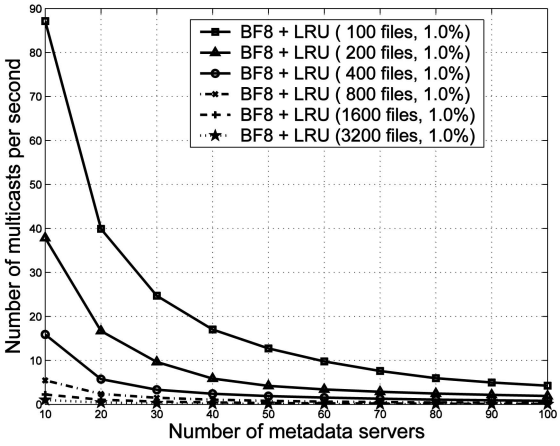


Fig. 13. Network overhead of HBA with different LRU sizes ($\mathrm{TIF} = 40$, and LRU BF Threshold = 1 percent).

balance of this trade-off. Fig. 13 gives the network traffic under this threshold when both the number of MSs and the size of LRU list changes. When the size of LRU is larger than 1,600, the total network traffic overhead introduced by HBA in most cases of MS configurations are less than 1 multicast per second. This overhead is marginal in a modern network.

7 Implementation of Hierarchical Bloom Filter Arrays

We implement the HBA prototype on the Linux kernel 2.4.21 as an I/O daemon running on each MS. All internal communications use TCP/IP, and the request forwarding between MSs is implemented by using IP-IP encapsulation [44], [45]. Currently, our client-side component is not able to directly intercept the user system I/O calls. A system call trapping mechanism, similar to the one used in PVFS [3], has not been implemented yet. Instead, we provide a C library that includes functions analogous to the Unix/POSIX functions such as *HBA_open*, *HBA_close*, and *HBA_stat*. We run experiments on the Sandhills cluster that has 40 nodes, each equipped with dual AMD processors. Due to the limitations of available hardware resource and client-side I/O interfaces, we could not evaluate our design by running real large-scale applications, in which thousands of clients

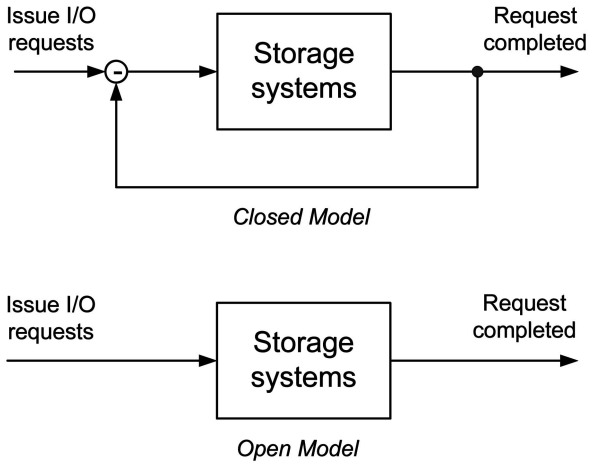


Fig. 14. Two models to replay I/O traces in real systems.

access the MSs concurrently. To overcome this limitation, we choose to use I/O traces to drive the client applications, in which the events are replayed through the library functions.

A challenging issue in replaying the trace is to realistically model timing effects that specify how fast the trace should be replayed. This difficulty stems from the fact that the arrival of subsequent I/O requests usually depends upon the completion of previous I/Os, and such dependency information cannot be easily extracted from a system and recorded in the traces [46]. There are two widely used models, that is, the *closed model* and the *open model*, that have been employed to solve the timing issue, as shown in Fig. 14. In the open model, I/O requests are issued at predetermined times specified in the traces, without considering the performance of storage systems. This model tends to ignore the dependencies among requests and continuously issue new requests, regardless whether their previous dependent requests have finished or not. Clearly, this does not truly reflect requests' behavior in real systems. In addition, the open model is impractical for long-time traces, since the experiments require the same amount of time as the trace collected. In the closed model, on the other hand, new requests are only issued when their previous dependent requests are being served. Although this model considers the load feedback from storage systems, it tends to smooth out the burstiness and reduce the number of outstanding requests.

In this work, we choose to use the closed model because of three main reasons. First, metadata operations are highly dependent on one another. For example, locating a file needs searching the metadata of parent directories recursively until reaching the root. Second, metadata operations are synchronous [15], [47], which implies that subsequent processes have to wait until current I/Os are finished. Third, our traces are collected over a period of 1 month, and it is not practical for us to replay the traces with the exact speed at which the trace was collected.

All metadata is stored in the local file system of each MS. Similar to PVFS, the metadata attributes of a file are stored as a metadata file with the same full pathname on its home MS. The metadata attributes of each file are split into two parts: some attributes such as ownership and creation time are exactly the attributes of its metadata file and are

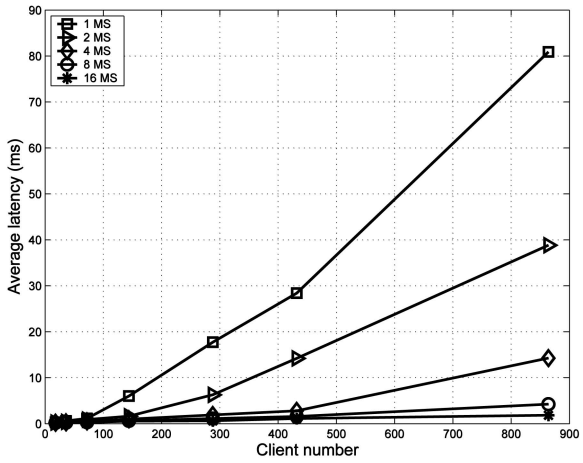


Fig. 15. Average latency of metadata operations.

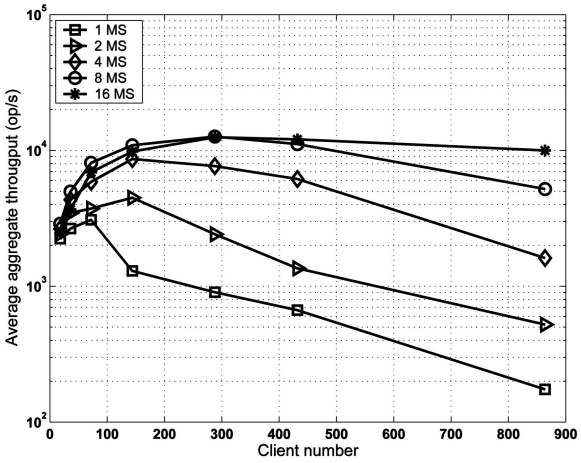


Fig. 16. Average throughput of metadata operations.

managed through local file systems, whereas the other attributes, including file size and data locations, are stored as the content of the metadata file, and they are managed by the HBA server daemons.

We store directory metadata in a way different from PVFS. Although both PVFS and HBA follow conventional file system hierarchical directory structures, HBA stores the names of files and subdirectories as part of the metadata of their parent directories. The names of files in all subdirectories are not included in the metadata of their parent directories. Note that we only store them as an unsorted list instead of the B-tree or extensible hashing used in enterprise file systems [48], [49]. This is because HBA does not rely on directory metadata to support file lookup. Directory metadata is used to reduce the overhead of directory renaming and list operations such as "*ls*." When a directory is renamed, the home MS of that directory iterates the list and updates corresponding BFs that might be stored on other servers. Recursively, the home MSs of subdirectories also perform the same operations. An improvement to our current design could be explained as follows: When the target directory is close to the root or has a large number of files, the home MS can first retrieve the latest BFs from other servers, then perform renaming in the filter array stored locally, and finally propagate the changes in their replicas in other MSs. The attributes of subdirectories and files are not included to avoid frequent updates. Thus, an "*ls -l*" command can potentially induce a flood of network traffic to multiple servers in our current implementation.

We choose to use the larger traces, that is, the HP traces, instead of the RES traces, to evaluate our design. Before the experiments, all MSs and the PBA are initially populated with existing files with inferred attributes from the traces. The traces are scaled up with a factor of 40 by the same approach presented previously in Section 5. In order to stress the MSs, we use up to 20 client nodes to concurrently replay the subtraces. These client nodes do not serve as any MS. There is a separate thread for each subtrace. When the total number of subtraces exceeds 20, multiple threads may run simultaneously on all client nodes. All records in a subtrace are played back in a sequential order, and we do not use multithreading to explicitly emulate the parallelism of different users within each subtrace. All MSs are initially populated randomly.

Figs. 15 and 16 compare the average latency and the average aggregate throughput of metadata operations in a conventional single-metadata-server design with those in the HBA schemes with 2, 4, 8, and 16 MSs, respectively, where the aggregate throughput represents the total number of metadata operations that can be completed by all MSs per time unit. The results are measured under various numbers of clients, that is, the total number of users in subtraces replayed simultaneously. Each measurement is performed in a quiet environment, with no other applications running, and is repeated three times. The arithmetic averages are reported here, and the maximum deviation from the median was always below 10.6 percent of the reported value. From the simulation results, we can draw the following observations.

1. When the workload increases, the throughput of all configurations increases initially and then decreases dramatically after saturating the servers' processing capacities. Compared with the single-metadata-server system, the peak throughput of the HBA schemes with 2, 4, 8, and 16 MSs is 1.7, 2.8, 3.8, and 4.4 times higher, respectively.
2. The latency reduction becomes more significant with the increase in workload. Under the heaviest workload studied in our experiments, a configuration of 16 MSs reduces the response time of a single-metadata-server architecture by a factor of 43.9. Fig. 17 shows the speedups of our HBA scheme under different workloads, where speedup is defined as

$$\text{speedup of HBA} = \frac{\text{average latency in HBA}}{\text{average latency in a single metadata server}}. \quad (2)$$

   Under heavy workloads, our HBA scheme can exhibit a superlinear speedup.
3. Under light workloads, the HBA scheme with 16 MSs surprisingly performs up to 14.7 percent inferior to the scheme with eight MSs in the average aggregate throughput and 13.6 percent inferior in

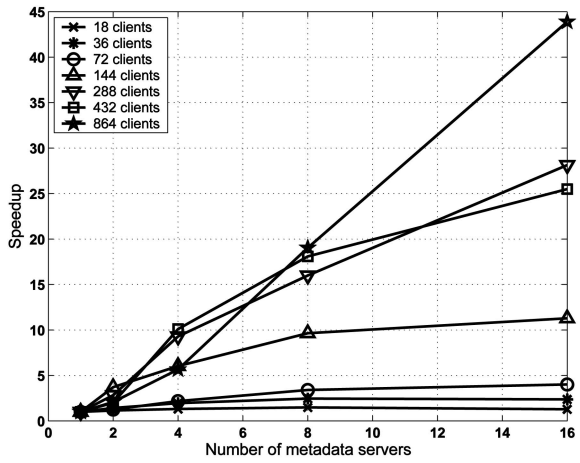


Fig. 17. Speedup under different workloads.

the average response time. When the number of MSs increases, the probability of false positives in a BF array also increases, which increases the number of multicasts to resolve the false positives. Such increased overhead generated by extra multicasts offsets the benefits of load sharing when the workload is not heavy.

In summary, the experimental results based on our prototype implementation indicate that the HBA scheme maintains a strong scalability in increasing the aggregate throughput and reducing the latency of metadata operations. We might underestimate the benefits of the HBA design, since a closed model is employed to replay the traces in real systems. In such models, I/O is issued only when its previous I/Os have completed. This certainly decreases the number of outstanding I/O requests and hence reduces the queuing time of I/O requests. Applications may be able to achieve higher speedups than the ones reported in our experiments due to HBA's ability to quickly absorb (or dissipate) requests waiting in the queues.

8 Conclusion

This paper has analyzed the efficiency of using the 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 our theoretic analysis and simulation results indicated that this approach cannot scale well with the increase in the number of MSs and has very large memory overhead when the number of files is large.

By exploiting the temporal access locality of file access patterns, this paper has proposed a hierarchical scheme, called HBA, that maintains two levels of BF arrays, with the one at the top level succinctly representing the metadata location of most recently visited files on each MS and the one at the lower level maintaining metadata distribution information of all files with lower accuracy in favor of memory efficiency. The top-level array is small in size but has high lookup accuracy. This high accuracy compensates for the relatively low lookup accuracy and large memory requirement in the lower level array. Our extensive trace-driven simulations show that the HBA scheme can achieve an efficacy comparable to that of PBA but at only 50 percent of memory cost and slightly higher network traffic overhead (multicast). On the other hand, HBA incurs 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. We have implemented our HBA design in Linux and measured its performance in a real cluster. The experimental results show that the performance of HBA is very promising. Under heavy workloads, HBA with 16 MSs can reduce the metadata operation time of a single-metadata-server architecture by a factor of up to 43.9.

Compared with other existing solutions to decentralizing metadata management, HBA retains most 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, and does not require metadata migration during file or directory renaming and node additions or deletions.

We have also designed an efficient and accurate metadata prefetching algorithm to further improve metadata operation performance [50]. We are incorporating this prefetching scheme into HBA.

There are several limitations in this research work:

- Our workload trace does not include metadata operations of parallel I/Os used in scientific applications. This is mainly due to the lack of long-term traces that include sufficient amount of metadata operations.
- When a new MS is added, a self-adaptive mechanism is needed to automatically rebalance the metadata spatial distribution. For the purpose of this paper, we assume that such a mechanism is available.

Acknowledgments

This work is partially supported by the US National Science Foundation (NSF) Funds (CCF 0621493, 0621526, 0754951, CNS 0619430, 0723093, 0646910, 0646911, and DRL 0737583), US Department of Energy (DoE) Early Career Award DE-FG02-07ER25747, Maine NASA Space Grant, Chinese NSF 973 Project under Grant 2004cb318201, and equipment donations from SUN. The authors greatly appreciate Dong Li for developing preliminary trace transformation codes, HP Labs and the University of California, Berkeley, for providing them the file system traces, and the reviewers for their constructive comments and suggestions.

References

[1] P.J. Braam, "Lustre White Paper," http://www.lustre.org/docs/whitepaper.pdf, 2005.

[2] S.A. Brandt, L. Xue, E.L. Miller, and D.D.E. Long, "Efficient Metadata Management in Large Distributed File Systems," *Proc. 20th IEEE Mass Storage Symp./11th NASA Goddard Conf. Mass Storage Systems and Technologies (MSS/MSST '03)*, pp. 290-298, Apr. 2003.

[3] P.H. Carns, W.B. Ligon III, R.B. Ross, and R. Thakur, "PVFS: A Parallel File System for Linux Clusters," *Proc. Fourth Ann. Linux Showcase and Conf.,* pp. 317-327, 2000.

[4] S. Ghemawat, H. Gobioff, and S.-T. Leung, "The Google File System," *Proc. 19th ACM Symp. Operating Systems Principles (SOSP '03),* pp. 29-43, 2003.

[5] H. Tang and T. Yang, "An Efficient Data Location Protocol for Self-Organizing Storage Clusters," *Proc. ACM/IEEE Conf. SuperComputing (SC '03),* p. 53, Nov. 2003.

[6] Y. Zhu and H. Jiang, "CEFT: A Cost-Effective, Fault-Tolerant Parallel Virtual File System," *J. Parallel and Distributed Computing,* vol. 66, no. 2, pp. 291-306, Feb. 2006.

[7] 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,* vol. 15, no. 1, pp. 29-36, 1995.

[8] D.H. Carrere, "Linux Local and Wide Area Network Adapter Support," *Int'l J. Network Management,* vol. 10, no. 2, pp. 103-112, 2000.

[9] C. Eddington, "Infinibridge: An Infiniband Channel Adapter with Integrated Switch," *IEEE Micro,* vol. 22, no. 2, pp. 48-56, 2002.

[10] Y. Zhu, H. Jiang, X. Qin, D. Feng, and D. Swanson, "Exploiting Redundancy to Boost Performance in a RAID-10 Style Cluster-Based File System," *Cluster Computing: The J. Networks, Software Tools and Applications,* vol. 9, no. 4, pp. 433-447, Oct. 2006.

[11] M. Vilayannur, A. Sivasubramaniam, M. Kandemir, R. Thakur, and R. Ross, "Discretionary Caching for I/O on Clusters," *Proc. Third IEEE/ACM Int'l Symp. Cluster Computing and the Grid (CCGRID '03),* pp. 96-103, May 2003.

[12] W.B. Ligon III and R.B. Ross, "Server-Side Scheduling in Cluster Parallel I/O Systems," *Calculateurs Paralleles,* special issue on parallel I/O for cluster computing, Oct. 2001.

[13] 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)," *Proc. Fourth LCI Int'l Conf. Linux Clusters,* June 2003.

[14] J. Wu, P. Wyckoff, and D. Pandac, "PVFS over InfiniBand: Design and Performance Evaluation," *Proc. Int'l Conf. Parallel Processing (ICPP '03),* pp. 125-132, Oct. 2003.

[15] D. Roselli, J.R. Lorch, and T.E. Anderson, "A Comparison of File System Workloads," *Proc. Ann. Usenix Technical Conf.,* June 2000.

[16] Y. Zhu, H. Jiang, and J. Wang, "Hierarchical Bloom Filter Arrays (HBA): A Novel, Scalable Metadata Management System for Large Cluster-Based Storage," *Proc. IEEE Int'l Conf. Cluster Computing (CLUSTER '04),* pp. 165-174, Sept. 2004.

[17] Y. Zhu, H. Jiang, X. Qin, and D. Swanson, "A Case Study of Parallel I/O for Biological Sequence Search on Linux Clusters," *Proc. IEEE Int'l Cluster Computing (CLUSTER '03),* pp. 308-315, Dec. 2003.

[18] F. Schmuck and R. Haskin, "GPFS: A Shared-Disk File System for Large Computing Clusters," *Proc. First Usenix Conf. File and Storage Technologies (FAST '02),* pp. 231-244, Jan. 2002.

[19] J. Kubiatowicz, D. Bindel, Y. Chen, S. Czerwinski, P. Eaton, D. Geels, R. Gummadi, S. Rhea, H. Weatherspoon, C. Wells, and B. Zhao, "Oceanstore: An Architecture for Global-Scale Persistent Storage," *Proc. Ninth Int'l Conf. Architectural Support for Programming Languages and Operating Systems (ASPLOS '00),* pp. 190-201, 2000.

[20] D. Nagle, D. Serenyi, and A. Matthews, "The Panasas ActiveScale Storage Cluster: Delivering Scalable High Bandwidth Storage," *Proc. ACM/IEEE Conf. Supercomputing (SC '04),* p. 53, 2004.

[21] T. Anderson, M. Dahlin, J. Neefe, D. Pat-terson, D. Roselli, and R. Wang, "Serverless Network File Systems," *Proc. 15th ACM Symp. Operating System Principles (SOSP '95),* pp. 109-126, Dec. 1995.

[22] B. Pawlowski, C. Juszczak, P. Staubach, C. Smith, D. Lebel, and D. Hitz, "NFS Version 3: Design and Implementation," *Proc. Usenix Summer Technical Conf.,* pp. 137-151, 1994.

[23] J.H. Morris, M. Satyanarayanan, M.H. Conner, J.H. Howard, D.S. Rosenthal, and F.D. Smith, "Andrew: A Distributed Personal Computing Environment," *Comm. ACM,* vol. 29, no. 3, pp. 184-201, 1986.

[24] M. Satyanarayanan, J.J. Kistler, P. Kumar, M.E. Okasaki, E.H. Siegel, and D.C. Steere, "Coda: A Highly Available File System for Distributed Workstation Environments," *IEEE Trans. Computers,* vol. 39, no. 4, Apr. 1990.

[25] V. Cate and T. Gross, "Combining the Concepts of Compression and Caching for a Two-Level File System," *Proc. Fourth Int'l Conf. Architectural Support for Programming Languages and Operating Systems (ASPLOS '91),* pp. 200-211, Apr. 1991.

[26] R. Floyd, "Short-Term File Reference Patterns in a Unix Environment," Technical Report TR-177, Computer Science Dept., Univ. of Rochester, Mar. 1986.

[27] E. Riedel, M. Kallahalla, and R. Swaminathan, "A Framework for Evaluating Storage System Security," *Proc. First Usenix Conf. File and Storage Technology (FAST '02),* pp. 15-30, Mar. 2002.

[28] C.H. Staelin, "High Performance File System Design," PhD dissertation, Dept. Computer Science, Princeton Univ., Oct. 1991.

[29] S.A. Weil, K.T. Pollack, S.A. Brandt, and E.L. Miller, "Dynamic Metadata Management for Petabyte-Scale File Systems," *Proc. ACM/IEEE Conf. Supercomputing (SC '04),* p. 4, 2004.

[30] P.F. Corbett and D.G. Feitelson, "The Vesta Parallel File System," *ACM Trans. Computer Systems,* vol. 14, no. 3, pp. 225-264, 1996.

[31] P.J. Braam and P.A. Nelson, "Removing Bottlenecks in Distributed Filesystems: Coda and InterMezzo as Examples," *Proc. Linux Expo,* May 1999.

[32] T.E. Anderson, M.D. Dahlin, J.M. Neefe, D.A. Patterson, D.S. Roselli, and R.Y. Wang, "Serverless Network File Systems," *ACM Trans. Computer Systems,* vol. 14, no. 1, pp. 41-79, 1996.

[33] M.N. Nelson, B.B. Welch, and J.K. Ousterhout, "Caching in the Sprite Network File System," *ACM Trans. Computer Systems,* vol. 6, no. 1, pp. 134-154, 1988.

[34] M.K. McKusick, W.N. Joy, S.J. Leffler, and R.S. Fabry, "A Fast File System for Unix," *ACM Trans. Computer Systems,* vol. 2, no. 3, pp. 181-197, 1984.

[35] M. Rosenblum and J.K. Ousterhout, "The Design and Implementation of a Log-Structured File System," *Proc. 13th ACM Symp. Operating Systems Principles (SOSP '91),* pp. 1-15, 1991.

[36] B.H. Bloom, "Space/Time Trade-Offs in Hash Coding with Allowable Errors," *Comm. ACM,* vol. 13, no. 7, pp. 422-426, 1970.

[37] L. Fan, P. Cao, J. Almeida, and A.Z. Broder, "Summary Cache: A Scalable Wide-Area Web Cache Sharing Protocol," *IEEE/ACM Trans. Networking,* vol. 8, no. 3, pp. 281-293, 2000.

[38] A. Broder and M. Mitzenmacher, "Network Applications of Bloom Filters: A Survey," *Proc. 40th Ann. Allerton Conf. Comm., Control and Computing,* Oct. 2002.

[39] S. Dharmapurikar, P. Krishnamurthy, and D.E. Taylor, "Longest Prefix Matching Using Bloom Filters," *Proc. ACM SIGCOMM '03,* pp. 201-212, Aug. 2003.

[40] J.K. Mullin, "A Second Look at Bloom Filters," *Comm. ACM,* vol. 26, no. 8, pp. 570-571, 1983.

[41] M.V. Ramakrishna, "Practical Performance of Bloom Filters and Parallel Free-Text Searching," *Comm. ACM,* vol. 32, no. 10, pp. 1237-1239, 1989.

[42] F. Wang, Q. Xin, B. Hong, S.A. Brandt, E.L. Miller, D.D.E. Long, and T.T. McLarty, "File System Workload Analysis for Large Scale Scientific Computing Applications," *Proc. 21st IEEE Mass Storage Symp./12th NASA Goddard Conf. Mass Storage Systems and Technologies (MSS/MSST '04),* Apr. 2004.

[43] M.P. Mesnier, M. Wachs, R.R. Sambasivan, J. Lopez, J. Hendricks, G.R. Ganger, and D. O'Hallaron, "//TRACE: Parallel Trace Replay with Approximate Causal Events," *Proc. Fifth Usenix Conf. File and Storage Technologies (FAST '07),* pp. 153-167, Feb. 2007.

[44] C. Perkins, *IP Encapsulation within IP,* IBM, 1996.

[45] L. Aversa and A. Bestavros, "Load Balancing a Cluster of Web Servers Using Distributed Packet Rewriting," technical report, 1999.

[46] W.W. Hsu and A.J. Smith, "The Performance Impact of I/O Optimizations and Disk Improvements," *IBM J. Research and Development,* vol. 48, no. 2, pp. 255-289, 2004.

[47] C. Ruemmler and J. Wilkes, "Unix Disk Access Patterns," *Proc. Usenix Winter Technical Conf.,* pp. 405-502, 1993.

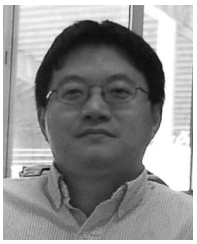
[48] *XFS: A High-Performance Journaling Filesystem,* http://oss.sgi.com/projects/xfs/, Feb. 2007.

[49] *Red Hat Global File System,* http://www.redhat.com/software/rha/gfs/, Feb. 2007.

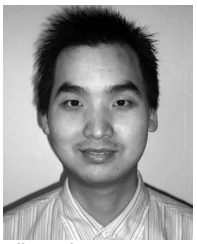
[50] P. Gu, Y. Zhu, H. Jiang, and J. Wang, "Nexus: A Novel Weighted-Graph-Based Group Prefetching Algorithm for Metadata Servers in Petabyte Scale Storage Systems," *Proc. Sixth IEEE Int'l Symp. Cluster Computing and the Grid (CCGRID '06),* pp. 409-416, May 2006.

Yifeng Zhu received the BSc degree in electrical engineering from Huazhong University of Science and Technology, Wuhan, China, in 1998 and the MS and PhD degrees in computer science from the University of Nebraska, Lincoln, in 2002 and 2005, respectively. He is currently an assistant professor in the Department of Electrical and Computer Engineering, University of Maine. His research interests include parallel I/O storage systems, supercomputing, energy-aware memory systems, and wireless sensor networks. He served as the program chair of SNAPI '07 and the committee of various international conferences, including ICDCS, ICPP, and NAS. He received Best Paper Award at IEEE CLUSTER '07 and several research and education grants from the US National Science Foundation HECURA, ITEST, REU, and MRI. He is a member of the ACM, the IEEE, the IEEE Computer Society, and the Francis Crowe Society.

Hong Jiang received the BSc degree in computer engineering from Huazhong University of Science and Technology, Wuhan, China, in 1982, the MASc degree in computer engineering from the University of Toronto in 1987, and the PhD degree in computer science from the Texas A&M University, College Station, in 1991. Since August 1991, he has been with the University of Nebraska, Lincoln, where he is currently a professor in the Department of Computer Science and Engineering. His research interests include computer architecture, computer storage systems and parallel I/O, parallel/distributed computing, cluster and grid computing, performance evaluation, real-time systems, middleware, and distributed systems for distance education. He has more than 135 publications in major journals and international conference proceedings in these areas, and his research has been supported by the US National Science Foundation (NSF), US Department of Defense (DoD), and the State of Nebraska. He is a member of the IEEE, the ACM, and the ACM SIGARCH.



Jun Wang received the BEng degree in computer engineering from Wuhan University (for merly Wuhan Technical University of Surveying and Mapping), the MEng degree in computer engineering from Huazhong University of Science and Technology, Wuhan, China, and the PhD degree in computer science and engineering in 2002 from the University of Cincinnati. He is currently an assistant professor in the School of Electrical Engineering and Computer Science, University of Central Florida. His research interests include I/O architecture, file and storage systems, parallel and distributed computing, cluster and P2P computing, and performance evaluation. He has received several major research grants from the US National Science Foundation and the US Department of Energy Early Career Principal Investigator Award Program. He is a member of the IEEE, the ACM, the Usenix, and the SNIA.



Feng Xian received the bachelor's degree from Chongqing University of Posts and Telecommunications, Chongqing, China, in 1999 and the master's degree from Huazhong University of Science and Technology, Wuhan, China, in 2003. He is currently working toward the PhD degree in the Department of Computer Science and Engineering, University of Nebraska, Lincoln. His research interests include memory management and programming languages, distributed systems, and network security. He is a student member of the IEEE and the ACM SIGPLAN.