Pacon: Improving Scalability and Efficiency of Metadata Service through Partial Consistency

Yubo Liu[1,2], Yutong Lu[1], Zhiguang Chen[1], Ming Zhao[2]
[1]*Sun Yat-sen University,* [2]*Arizona State University*

***Abstract*—Traditional distributed file systems (DFS) use centralized service to manage metadata. Many studies based on this centralized architecture enhanced metadata processing capability by scaling the metadata server cluster, which is however still difficult to keep up with the growing number of clients and the increasingly metadata-intensive applications. Some solutions abandoned the centralized metadata service and improved scalability by embedding a private metadata service in an HPC application, but these solutions are suitable for only some specific applications and the absence of global namespace makes data sharing and management difficult. This paper addresses the shortcomings of existing studies by optimizing the consistency model of client-side metadata cache for the HPC scenario using a novel partial consistency model. It provides the application with strong consistency guarantee for only its workspace, thus improving metadata scalability without adding hardware or sacrificing the versatility and manageability of DFSes. In addition, the paper proposes batch permission management to reduce path traversal overhead, thereby improving metadata processing efficiency. The result is a library (Pacon) that allows existing DFSes to achieve partial consistency for scalable and efficient metadata management. The paper also presents a comprehensive evaluation using intensive benchmarks and representative application. For example, in file creation, Pacon improves the performance of BeeGFS by more than 76.4 times, and outperforms the state-of-the-art metadata management solution (IndexFS) by more than 4.6 times.**

***Index Terms*—metadata, scalability, efficiency, distributed file system, consistency**

I. Introduction

The scalability and efficiency of metadata service have always been an important topic in the distributed file system (DFS) research. Traditional DFSes use a centralized service to manage metadata. This centralized architecture faces scalability and efficiency issues in large scale systems, especially in high performance computing (HPC) systems. There are studies (e.g., [1]–[3]) improve the metadata processing ability by scaling the metadata service cluster. But their overall throughput is still difficult to scale as the number of DFS clients increases and the metadata service gets saturated. Therefore, these systems based on the centralized architecture need to deploy a lot of metadata servers to meet the peak demand of the systems [3]. This static hardware scaling method is wasteful or unrealistic in modern HPC scenarios, because the number of clients in the HPC system can be very large. For example, a petascale super computing system may have up to hundreds of thousands of concurrent clients on the compute nodes.

Some studies proposed private metadata service to address the scalability and efficiency issues of the centralized metadata service in HPC systems. For example, BatchFS [4] and DeltaFS [5] embed the metadata service into the application. But these solutions have two critical limitations. First, they can be used for only some specific applications. One of the most suitable scenarios for them is N-N checkpoint, where there is no interaction among the processes. Their performance and scalability improvements mainly come from the elimination of consistency guarantees between the clients (processes). Second, they do not provide (in case of DeltaFS) or need extra overhead to provide (in case of BatchFS) global namespace, which makes data sharing and file system management difficult.

In summary, existing metadata management solutions (centralized and private) cannot simultaneously meet the scalability, versatility, and manageability requirements. The goal of this work is to meet all these requirements together. We find that existing DFSes use strong consistency model in the client-side metadata cache. It makes clients communicate frequently and synchronously with the metadata servers to guarantee the consistency, resulting in a scalability bottleneck. However, strong consistency is an overkill in many HPC workloads (see Section II.A for details). Our key idea is to relax the cache consistency according to the needs of HPC applications and allow clients to communicate asynchronously with metadata servers. Therefore, the pressure of metadata processing can be offloaded to a large number of client nodes.

Our first contribution is to propose a new consistency model of the client-side metadata cache for the centralized architecture, which we call partial consistency. Partial consistency splits the global namespace into multiple consistent regions according to the workspaces (directories) of HPC applications. It provides an application with strong consistency guarantee within the consistent region by using a distributed metadata cache on the clients that belong to the same application, but it relaxes the consistency among different consistent regions. It also allows applications to merge their consistent regions to support data sharing across different workspaces. In partial consistency, most metadata operations can be absorbed by the distributed cache and asynchronously committed to centralized metadata service. So it can greatly improve metadata scalability without adding hardware or sacrificing the versatility and manageability of DFS.

We also optimize the metadata processing in the distributed metadata cache. The distributed cache is responsible for permission checking of the metadata operations. The traditional approach is to traverse each level of the path and check their

1530-2075/20/$31.00 ©2020 IEEE
DOI 10.1109/IPDPS47924.2020.00105

IEEE computer society

permission information. However, path traversal is costly. In the related works, ShardFS [6] and LocoFS [7] accelerate path traversal by reducing the network overhead. ShardFS maintains the namespace structure in each metadata server, but at the cost of consistency guarantee. LocoFS keeps the directory metadata in a single metadata server, but it will cause single point of performance bottleneck and single point of failure. These optimizations make a big trade-off on path traversal overhead and other important aspects.

Our second contribution is to directly avoid path traversal during permission checking in the distributed metadata cache. In general, there are two characteristics of permission management in the HPC scenario. First, administrators create a separate system user for each HPC application, that is, all clients in the same application will use the same system user to access the DFS. Second, an HPC application can predict the permission information of its workspace. According to these characteristics, we let HPC applications predefine the permissions of the directories/files under their working directories (consistent regions) and use batch permission management instead of hierarchical permission authentication to reduce the overhead.

We implemented a library called Pacon to allow partial consistency to be adopted by any DFS. In the evaluation, we deployed Pacon on BeeGFS and compared it with native BeeGFS and the start-of-the-art metadata management solution (IndexFS [3]). In the single-application experiment, the results show that the throughput of Pacon is more than 6.5 times higher than BeeGFS and more than 2.6 times higher than IndexFS. In the multi-application experiment, Pacon's throughput can be more than one order of magnitude higher than BeeGFS and more than 1.07 times higher than IndexFS. Pacon has good scalability, and it can get more than 1 million OPS (operations per second) in file creation when the number of clients reaches 320. We also evaluated Pacon with MADbench2, a real-world HPC benchmark. The results show that Pacon does not sacrifice other important system aspects (such as the computation) in this case while achieving scalable and efficient metadata management.

The rest of the paper is organized as follows: Section II introduces the research background and motivations; Section III describes the design details of Pacon; Section IV evaluates Pacon's performance; Section V examines some related work; and Section VI concludes the paper.

II. Background and Motivations

This section first introduces the characteristics of architecture and applications in the HPC scenario. Then, it analyzes the metadata scalability and the cache consistency model. Finally, it analyzes the overhead of permission check in the DFS.

A. High Performance Computing Scenario

We summarize the characteristics of hardware architecture and applications in the HPC scenario. For the hardware architecture, an HPC system has a large number of compute nodes (clients), but the number of the storage nodes is relatively small. For example, TIANHE-II in the National Supercomputer Center in Guangzhou (NSCC-GZ) has approximately 16,000 compute nodes, but it has only hundreds of storage nodes [8].

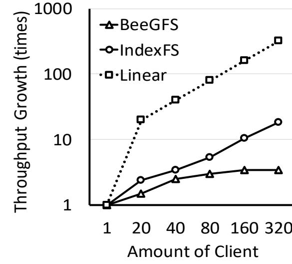


Fig. 1: Client Scalability.

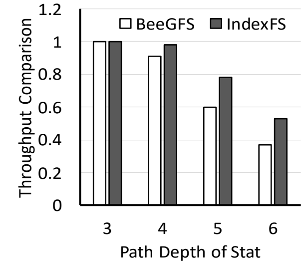


Fig. 2: Path Traversal Cost.

Applications also have some important characteristics in the HPC scenario: First, each application needs to use only a part of the namespace (one or more separate directories); Second, applications can easily predict the permission information of their workspaces. For example, in the actual use scenario of NSCC-GZ, the administrator typically creates a system user and allocates a working directory for the HPC application. So the clients of the same application will access the DFS by the same system user, and the permissions of the directories and files under the working directory are managed by the application. These characteristics of hardware and application in the HPC scenario bring new motivations for us to optimize the scalability and efficiency of metadata service.

B. Metadata Scalability and Consistency Model

The traditional approach is to use a centralized metadata service to provide a global namespace for all clients. A lot of studies improved the metadata scalability on this centralized architecture. For example, Lustre [9], BeeGFS [10], and CephFS [1] use multiple metadata servers to share the load of metadata processing; IndexFS [3] manages metadata by flattening the namespace and keeping metadata on KV stores; GlusterFS [2] distributes the metadata service on the data storage nodes. These systems can increase the scalability of metadata service to a certain extent by increasing the number of metadata servers, but the effectiveness of this approach is limited as it can hardly keep up with the rapidly growing number of clients which request intensive metadata service.

We evaluated the scalability of BeeGFS and IndexFS in file creation. BeeGFS is a widely used DFS in the HPC systems and it has higher metadata performance than Lustre in some cases [11]. IndexFS is a typical KV-based (LevelDB [12]) metadata management solution for DFS. We ran the clients on a cluster of 16 nodes. BeeGFS was setup with single MDS (metadata server). IndexFS was deployed on all client nodes and run upon the BeeGFS. We calculated the multiples of throughput when adding clients compared to the single client case. Figure 1 shows that their scalability has a lot of room for improvement. Furthermore, the centralized architecture requires a large number of metadata servers to

meet the peak demand of the system. In HPC systems, the huge difference in the number of metadata servers and clients makes this way of hardware expansion insufficient. Although IndexFS supports co-locating the metadata servers with the client (compute) nodes, it will waste the compute resource if we statically deploy a large number of metadata servers on the client nodes.

In order to support a large number of clients in some special HPC workloads, DeltaFS [5] and its predecessor, BatchFS [4] embed a private metadata service in the application to increase scalability. In fact, they are designed based on IndexFS and can be approximated as co-locating IndexFS servers with the client nodes, while leveraging the bulk insertion of IndexFS. Bulk insertion means that the clients buffer the new insertions locally and merge them to metadata servers in batches. This private metadata service approach has two critical shortcomings. First, it can be used for only some specific applications (e.g., N-N checkpoint) because the clients do not share a consistent view. Second, it does not directly provide global namespace, and cannot allow applications to share data through DFS.

In summary, the systems based on centralized metadata service have limited scalability, while the systems based on private metadata service face the versatility and manageability issues. We believe this dilemma can be solved by efficiently utilizing client metadata caching in the centralized architecture. Generally speaking, DFSes use strong consistency model in the client metadata cache. It means that each client can see the latest version of metadata at any time. According to the CAP theorem [13], choosing the strong consistency will sacrifice the availability. The strong consistency makes the client cache inefficient because many metadata operations must be synchronously applied to the metadata service. This makes the metadata service easily overloaded by metadata-intensive workloads, resulting in poor scalability.

Motivation 1. Our first motivation is to make a trade-off in cache consistency according to the characteristics of the HPC scenario discussed in Section II.A. We propose partial consistency. It weakens the consistency based on the needs of HPC applications, thus improving scalability without sacrificing other important system aspects.

C. Overhead of Path Traversal

Permission check is an important operation in the file system. The traditional way is to check the permission of each layer of the requested path. However, this path traversal is costly in DFS because it may take a lot of network I/Os. To evaluate the overhead of path traversal, we used mdtest [14] to create a namespace with 5 fanouts on the DFSes (BeeGFS and IndexFS). We increased the namespace depth and evaluated the throughput of randomly stating the leaf directories. Figure 2 shows that path traversal incurs more than 47% performance loss when the namespace depth reaches 6 compared to the case where the depth is 3. The main cause of performance loss is the network overhead caused by path traversal.

There are some studies focusing on reducing the network overhead of path traversal. For example, ShardFS [6] maintains

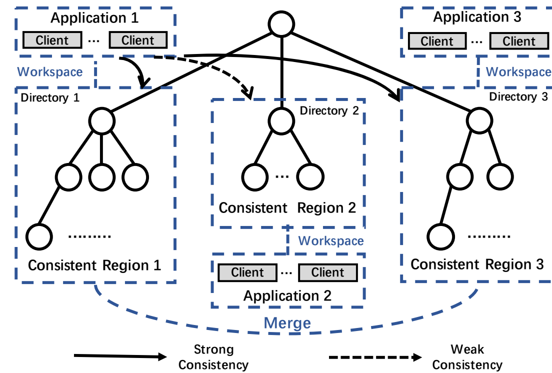


Fig. 3: Partial Consistency.

the whole namespace structure in each MDS node; LocoFS [7] stores all directory metadata on a single MDS node and the path traversal can be completed in a single node. The key idea of these studies is to reduce the number of RPCs during the path traversal. However, these optimizations also have some negative effects. For ShardFS, it needs to pay for the maintenance of the namespace information on the nodes. For LocoFS, the single directory metadata node will reduce the scalability and cause single point of failure.

Motivation 2. Our second motivation is to reduce path traversal overhead in permission checking. According to the characteristics of permission management in the HPC scenario (see Section II.A), we propose batch permission management to replace layer-by-layer checking in our system.

III. PACON DESIGN

In this section, we first introduce the partial consistency and its usage. Then we introduce the batch permission management and the main operations in Pacon. Finally, we introduce the distributed cache space management and node failure handling.

A. Partial Consistency

In order to provide high metadata scalability without sacrificing the versatility and manageability of DFS, we still use the centralized architecture, but improve scalability by optimizing the consistency model of the client-side metadata cache for the HPC scenario. Current DFSes use strong consistency model in client-side metadata cache to allow all clients to access the newest version of metadata at any time. According to the analysis in Section II.B, strong consistency model makes the client-side metadata cache inefficient and limits the scalability. However, using strong cache consistency is an overkill for many HPC workloads because they just run on some parts of the namespace. Therefore, we propose partial consistency: it provides application (clients belonging to the application) with strong consistency guarantee for only the workspace that it uses.

In partial consistency, the global namespace is split into multiple subtrees based on the workspaces of applications,

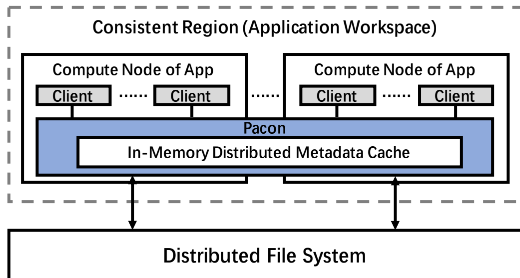


Fig. 4: Architecture of Pacon.

which we call consistent regions. We use an in-memory distributed cache on the clients that belong to the same application to cache the metadata of the workspace, so the clients can consistently access their workspace on the distributed cache (strong consistency). Requests outside the workspace will be redirected to the DFS (weak consistency). Metadata has two copies in partial consistency, one is in the distributed cache (primary copy) and the other is in the DFS (backup copy). The consistency protocol is to synchronously update the primary copy and asynchronously update the backup copy. Metadata reaches a globally consistent state when the backup copy is updated. Partial consistency also allows different applications to share a consistent view of their workspaces by merging their consistent regions together. For example, the clients of the application 1 in Figure 3 can consistently access its working directory 1 and the merged directory 3, but access to directory 2 may get inconsistent results.

We design Pacon, a library that allows partial consistency to be adopted by existing DFSes. Figure 4 shows the architecture of Pacon. Pacon runs on all client nodes and connects to the DFS (centralized metadata service). It is mainly responsible for building distributed caches on the client nodes, caching metadata, and committing metadata operations to the underlying DFS. In the initialization phase, Pacon launches a Memcached [15] cluster on the nodes where the application is running as the distributed in-memory cache. Pacon uses full path as the key to store the metadata, and distributes them in the distributed cache by DHT. Combining Memcached and DHT is a common design in cache systems, and they can be replaced with other in-memory KV databases and distributed algorithms. For metadata write operations, instead of synchronously applying the operations to the centralized metadata service, Pacon performs them on the distributed cache and pushes the operations into the commit queue. Commit processes apply metadata operations to the DFS by different commit strategies depending on the metadata operation type (see Section III.D and III.E). For metadata read operations, some of them can directly get the metadata from the distributed cache.

DFSes can get the following benefits from partial consistency. **Benefit 1:** High Scalability. Pacon still maintains a global namespace but offloads the work of the centralized metadata service to a large number of client nodes. This design can improve scalability without adding hardware or sacrificing the versatility and manageability. **Benefit 2:** Elasticity. Since the Pacon services are launched with an application's clients, it can dynamically adapt to the actual workload and save resources compared to statically deploying the services according to the system's peak demand. **Benefit 3:** High Throughput. Many metadata operations in Pacon can be asynchronously applied to the centralized metadata service, which allows the latency of the metadata servers to be hidden. In addition, we design a batch permission management method to reduce the path traversal overhead in permission check, thereby further improving the efficiency of metadata processing in the distributed cache (see Section III.C).

B. Using Pacon

Pacon provides basic file interfaces for HPC applications. An application needs to configure and initialize Pacon before running. The parameters of Pacon initialization mainly contain the path of the workspace (directory) and the network addresses of the nodes where the application runs. Thereafter, the file system operations under the working directory triggered by the application will be handled by Pacon, and the requests outside the working directory will be redirected to the underlying DFS.

There are three common use cases in the HPC scenario. Case 1: the application runs only under its working directory and does not need to interact with the other applications. In this case, the application just needs to define its consistent region in Pacon before running. Case 2: multiple applications run on non-overlapping working directory and need to share data on the DFS. In this case, applications not only need to define their consistent regions, but also need to merge them together. The merge interface of Pacon allows metadata to be consistently shared across multiple consistent regions. Case 3: applications' working directories overlap. In this case, we can consider them as running in the same large consistent region (the top one). For example, one application runs on "/A" and the other application runs on "/A/B", in which case we can consider both of them as running on "/A".

C. Batch Permission Management

Existing DFSes must perform costly path traversal to execute the permission check. As we discussed in Section II.A, we consider that an HPC application runs in a certain workspace and it can predict the permissions information of its workspace. Pacon lets HPC applications predefine the permission information of its workspace (consistent region) and keeps this information on all clients, so that permission check in the distributed cache only needs to match the permission information locally without executing path traversal.

Pacon maintains normal and special permission information for each consistent region. The normal permission defines permission information for most files and directories in the consistent region. The special permission information contains a list for the recording files/directories with different permission setting and the permission information of these items. For

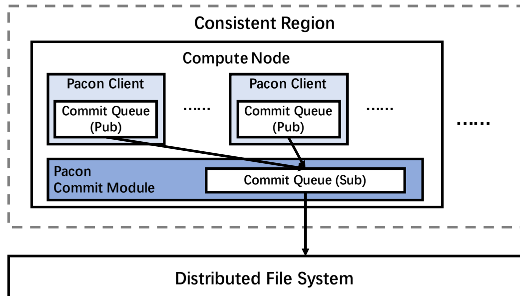


Fig. 5: Commit Queue in Pacon.

TABLE I: Main Metadata Operations in Pacon

Metadata Operation	create	mkdir	rm	getattr	rmdir	readdir
Cache Operation	put	put	update & delete	get	delete	N/A
Comm Type	async	async	async	N/A or sync (miss)	sync	sync
Commit Type	indep.	indep.	indep.	N/A or indep. (miss)	barrier	barrier

each metadata operation, Pacon first checks the request with the normal permission setting, and then checks it with each item in the special permission list. The request is approved if it can pass these checks. Then, Pacon can directly lookup the target metadata in the distributed cache by using its full path as the key without having to traverse the path layer by layer, thus reducing path traversal overhead.

If the application does not predefine the permission information, Pacon uses default permission settings similar to Linux (i.e., all directories and files in the workspace can be read, written, and executed by the creator) for the files and directories in the application's workspace. For those accesses across consistent regions, Pacon redirects them to the DFS without any permission checks, so they will be subject to the permission check of the underlying DFS. The directory/file creation operation not only requires permission authentication but also needs to check whether its parent directory exists. Pacon may need to check the parent directory on the DFS if it exists in the DFS but is not cached in the distributed cache. Pacon also allows the application to turn off the parent check, if the application itself can guarantee the correctness of the creation operations.

D. Operations in Pacon

1) Metadata Operations: Every metadata update on Pacon consists of two sub-operations. The first sub-operation is to perform the metadata operation on the distributed cache. The second sub-operation is to apply the metadata operation to the DFS by a commit module in Pacon. As shown in Figure 5, the commit queue uses the publisher-subscriber model. We use ZeroMQ [16] to implement the commit queue in our prototype. Each client in the consistent region is a publisher, and each node in the consistency region has a commit process that acts as the subscriber responsible for committing metadata operations to the DFS using the traditional file system interfaces (e.g., system calls and DFS client).

Table I shows the implementation details of the main metadata operations in Pacon, including the key operation(s) on the distributed cache ("Cache Operation"), communication type with the DFS ("Comm Type"), and the ways to submit operations to the DFS ("Commit Type"). In communication types, "async" and "sync" indicate the operation needs to commit to the DFS asynchronously and synchronously, respectively. In commit types, "indep." indicates that the commit process does not need to ensure that operations are applied to the DFS in the temporal order. On the contrary, "barrier" indicates that the commit process needs to ensure that those operations occurred before the pending operation have already been applied to the DFS. Operation commit is detailed in Section III.E.

For creating file/directory (create/mkdir) and removing file (rm), Pacon performs them on the distributed metadata cache through the in-memory KV operations and put an operation message into the commit queue. In particular, removed files are marked and their cached metadata are deleted after the operations are committed. The operation message includes the target path, operation information, and timestamp. Then, these operations can be returned without waiting for them to be applied to the DFS ("async" type). At the same time, they can be submitted without synchronization between different commit queues because these operations are "independent" ("indep." type; see Section III.E). For getting attributes (getattr), Pacon searches the target metadata in distributed metadata cache. If the target metadata is not in the distributed cache, Pacon synchronously calls the DFS interface to load it into the distributed cache (if it exists on the DFS).

For removing directory (rmdir), Pacon needs to remove all metadata under the target path on the distributed cache and the DFS. Pacon commits rmdir synchronously ("sync" type), which means the operation will only return when it is applied to the DFS. We use barrier commit ("barrier" type) to ensure its correctness. The commit process will directly discard the creation operation located in the directory being removed. Related metadata in the distributed cache will be cleaned during the recursive removing process. For listing directory (readdir), Pacon does not retrieve the metadata in the distributed cache but calls the DFS interface, and thus avoid the costly full table scan. Readdir also needs to communicate with the DFS synchronously. Pacon uses barrier before calling the readdir interface of the DFS to ensure the correctness of the listing result.

2) File operations: Files are divided into large files and small files in Pacon. The threshold for small file size can be customized by the user (it is 4KB in our prototype, including metadata and file data). Pacon stores small files with their metadata, so that applications can get both metadata and data in a single KV request. For the small files, file operations will be performed in the distributed cache. In particular, some operations require inline data to be stored on the DFS

immediately (e.g., fsync), but the target file may not have been created on the DFS because file creation in Pacon is not synchronously applied to the DFS. To address this case, Pacon uses direct I/O to temporarily write the data of the files that have not been created to the cache files. These data will be written back to their original positions after the target files are created. Pacon does not cache data of large files; it writes the file data to the DFS when the size of the file exceeds the threshold. For the large files that have been created on the DFS, all file operations will be redirected to the DFS.

3) Handling Concurrent Updates: To handle concurrent updates on metadata and small files (inline data), we do not use locks, but use the CAS (check-and-swap) interface of Memcached [15] to execute the updates. CAS works like versioning: it compares the item version with the version provided by client before updating the value, and the update operation will succeed only if these two versions are the same. When multiple write operations conflict on the same metadata or small file, Pacon will re-execute it until the update is successful.

4) Consistent Region Operations: The consistent region is defaulted to be a subtree in the namespace. To allow multiple applications to share a consistent view of their workspaces, Pacon supports merging multiple consistent regions (subtrees) together. The first step of merging consistent regions is to get the basic information (e.g., node addresses, permission information) of the consistent region that will be merged. The second step is to establish a connection between them, so that clients can access the distributed caches of another consistent region. After the merging process, Pacon will determine which consistent region (subtree) a metadata request belongs to, and access the metadata in the corresponding distributed cache. Currently, Pacon only supports read-only access to the merged consistent region.

E. Metadata Operation Commit

The commit module in Pacon is responsible for correctly applying the metadata operations to the DFS. A simple commit strategy is to commit the metadata operations in the queues completely based on the temporal order. However, this simple commit strategy will result in a high synchronization overhead. Our findings show that the temporal order is not necessary for all metadata operations. We classify metadata operations into two categories: non-dependent type and dependent type.

1) Non-dependent Type: It represents operations that only need to be committed following the namespace conventions, including creating file/directory and removing file. The basic namespace conventions are: 1) the object to be created must not exist; 2) the parent directory must be created earlier than its children; 3) the deleted object must have been created. These operations do not need to be committed in the temporal order for two reasons. The first reason is because the DFS can guarantee the namespace convention, i.e., these operations will be rejected by the DFS if they do not conform to the namespace convention. The second reason is that when the same set of operations is fully committed, the result (namespace structure) is the same regardless of the order in which these operations are committed. Pacon uses independent commit for the non-dependent operations.

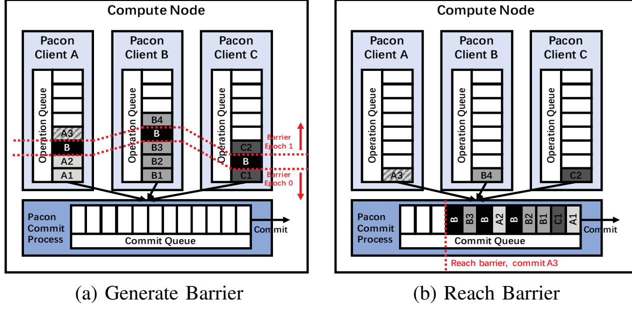
Independent Commit. For the non-dependent type, each message queue can commit these operations independently. If an operation fails to be committed to the DFS (e.g., the parent directory has not been created), we only need to resubmit the operation until it succeeds.

To prove that the independent committing can get the same result, let us consider clients that trigger only non-dependent operations during a certain period and their global time order is Seq0. Now there are two different sequences, Seq1 and Seq2, which contain the same operations as Seq0. We assume that all sequences follow the namespace conventions, so all operations can be committed. The initial state of the namespace is the same. First we prove that committing file removing independently will not affect the result. We assume that the orders of create and mkdir are the same in Seq1 and Seq2. If the conclusion is not true, it means that there is a file that exists in only one sequence. According to the namespace convention, its last state (exist or not) is opposite in Seq1 and Seq2. Since the file has the same operations in both sequences, we can deduce that the initial state of the file is the opposite in Seq1 and Seq2, which contradicts our hypothesis (same initial state), so the conclusion is established.

For the same set of operations, we have shown that the commit order of file deletion operations does not change the result if the commit order of creation operations is the same. Next we prove that the commit order of file/directory creation does not affect the result. According to the namespace conventions, these operations can be successfully committed only if they do not exist and their parent directory exists. So, the conclusion is established. The resubmitting in Pacon is used to adjust the operation order to conform the namespace conventions.

2) Dependent Type: It represents operations whose correctness will be affected by the order of other metadata operations, such as removing directory (rmdir). For example, when the commit process commits a rmdir operation, Pacon needs to ensure that all creations that occurred before the rmdir have been committed to the DFS. Therefore, these operations need to be committed in the temporal order. Pacon uses barrier commit for the dependent operations.

Barrier Commit. To ensure the correctness of the dependent type operations, we use a barrier method to make sure that the dependent operation is committed in the temporal order (i.e., those operations that occurred earlier than it have been committed when it is committed). We split the sequence of operations into multiple barrier epochs. Each operation holds a barrier epoch number to identify which barrier epoch that it belongs to, and the barrier epoch number will be increased when the client triggers a dependent operation or receives a barrier message. Commit process holds the current barrier epoch number and will only be increased after the dependent operation has been committed. The commit process will only commit those operations whose barrier epoch numbers are the



(a) Generate Barrier (b) Reach Barrier

Fig. 6: Barrier Committing. Striped squares indicate dependent operations (A3); Black squares indicate barrier message (B); Grey squares indicate no dependent operations.

same as the current barrier epoch number.

Figure 6 shows a simple example of barrier commit. A3 is a dependent operation called by client A. Before A3 is pushed into the operation queue, each client will generate a barrier message and push it into the operation queue, and then increase the barrier epoch number (as shown in Figure 6(a)). When the number of barrier messages received by the commit process is equal to the number of clients on the node (as shown in Figure 6(b)), it means that the operations of the previous barrier epoch have all been committed. So the commit process can commit the dependent operation (A3) and then increase the current barrier epoch number. Figure 6 is a single node case. In the multi-node case, the decision condition of whether an dependent operation can be committed requires all commit processes to reach the barrier.

F. Distributed Cache Space Management

The distributed metadata cache takes up only a small amount of memory resources of client nodes as the size of the metadata is very small. For example, a 500MB distributed cache space can store more than 10 million metadata without inline data. For this amount of metadata, the distributed metadata cache takes up only about 0.05% of the memory space in our testbed if the application runs on 16 nodes.

Since we assume that insufficient metadata cache is rare, so we design a simple eviction policy to make a trade-off between the overhead of memory management and hit rate. If the usage of the cache space exceeds a certain threshold, we select an entry (it can be a directory or a file) under the root directory of the consistent region and evict the metadata (committed to the DFS) under/of this entry. We use Round-Robin method to select the entry to be evicted, so that the eviction process will choose an entry that is different from the last eviction. This can alleviate cache thrashing that may be caused by the simple eviction policy to some extent.

G. Failure Recovery

Client node failure will cause the uncommitted metadata operations to be lost. However, the failure of a client node will affect only its own consistent region because the consistent regions are isolated from each other. Pacon addresses client node failure by periodically checkpointing the subtree of the consistent region on the DFS. When a node failure occurs within a consistent region, Pacon can roll back its corresponding subtree to the nearest checkpoint and rebuild the distributed cache of the consistent region. In order to adapt to the needs of different applications (such as different checkpoint intervals and implementations), we expose the interface of the checkpointing to applications. Unlike traditional checkpointing, we only need to checkpoint the application's workspace instead of the entire namespace. The checkpointing overhead equals to subtree copy. In fact, checkpoint is optional (for applications that need version rollback), and even without it, the DFS already guarantees the crash consistency of committed operations.

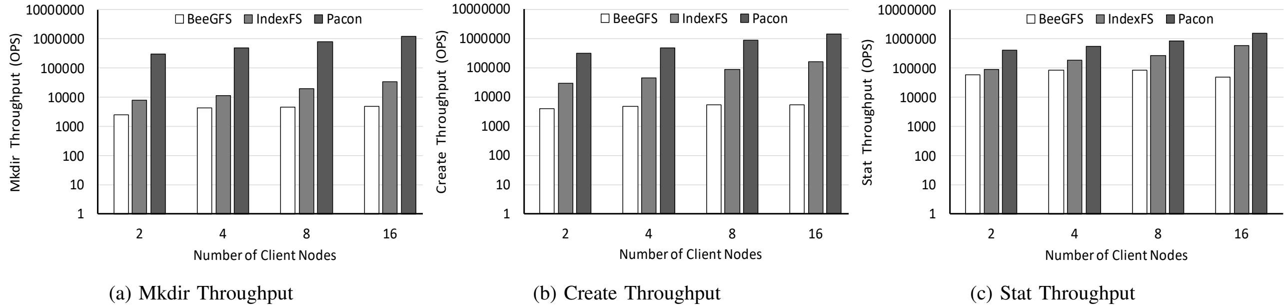
IV. Evaluation

To demonstrate the benefits that existing DFSes can get from Pacon, we deployed Pacon upon BeeGFS and compared it with native BeeGFS (for simplicity, "Pacon" is used to indicate "BeeGFS with Pacon" in this section). BeeGFS is a parallel DFS that is widely used in HPC systems. It has higher metadata performance than Lustre in some cases [11]. We also compared Pacon with IndexFS, a state-of-the-art metadata management solution. IndexFS is deployed upon BeeGFS too (the LevelDB tables are stored on BeeGFS). IndexFS has two main uses, one is to be deployed on the burst buffer nodes, and the other is to be co-located with the client nodes. Since Pacon was run on the client cluster, for fairness, we co-located IndexFS with the client nodes. We did not compare the private namespace DFSes (BatchFS and DeltaFS) because they are not general-purpose systems. BatchFS and DeltaFS can turn off the bulk insert to support non-batch applications, which can be approximated as an IndexFS deployed on the client nodes.

All experiments in this section were run on the TIANHE-II supercomputer. The client cluster has 16 nodes, each node has 2 Intel Xeon E5 CPU and 64GB RAM. The BeeGFS cluster has 1 metadata server and 3 data servers. The metadata and data servers of BeeGFS have the same hardware configuration as the client nodes, and MDS is mounted on an Intel P3600 PCIE NVMe SSD. We used mdtest [14], a metadata testing tool based on MPI [17], and MADbench2 [18], an HPC application benchmark to perform our experiments. We first evaluated the performance of Pacon in two cases: running a single application and running multiple applications. Then, we evaluated Pacon's path traversal efficiency, overhead, and scalability. Finally, we evaluated Pacon on a real-world HPC benchmark.

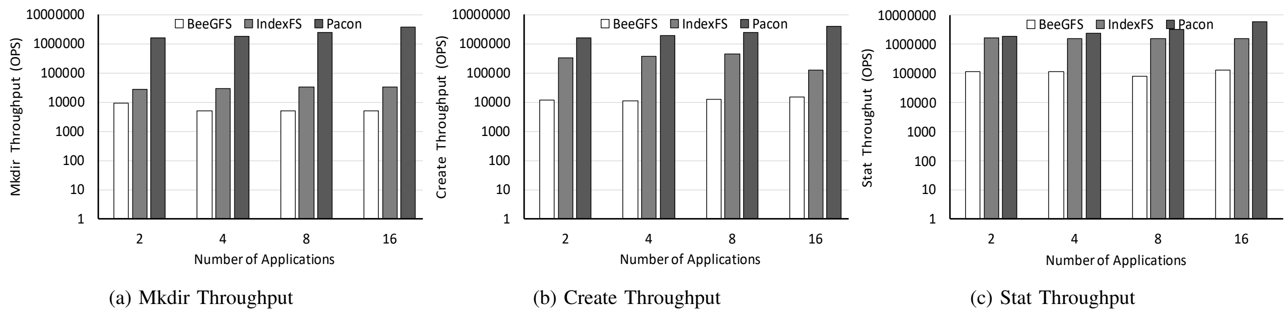
A. Single-application Case

We selected three common metadata operations in this part: creating directories, creating empty files, and randomly stating file. We ran mdtest (to simulate application) on different number of client nodes (from 2 nodes to 16 nodes), and ran 20 clients on each node. The clients concurrently create



(a) Mkdir Throughput (b) Create Throughput (c) Stat Throughput

Fig. 7: Performance of Single-application Case.



(a) Mkdir Throughput (b) Create Throughput (c) Stat Throughput

Fig. 8: Performance of Multi-application Case.

directories and files on the the same parent directory, and then randomly get the attributes of the files. The depth of the namespace is 1. For Pacon, there is only one consistent region in this experiment.

Figure 7 shows the performance comparison. Pacon can greatly speed up the metadata processing of BeeGFS. In the write operations (mkdir and create), Pacon improves performance by more than 76.4 times compared to BeeGFS. The performance improvement is mainly because Pacon buffers the metadata operations on the in-memory distributed cache and commit them to central metadata service asynchronously. It can take the throughput advantage of distributed cache (in-memory KV store) and absorb the load of metadata service in background. In the read operations (stat), Pacon has more than 6.5 times performance improvement over BeeGFS. In this experiment, all the metadata that are created by the mkdir and create operations are cached, so Pacon can quickly access metadata from the distributed metadata cache. Although BeeGFS has read cache on the client, it does not work much in random stating because the caches between client nodes are not shared.

Pacon is also better than IndexFS. For the write operations (mkdir and create), Pacon has more than 8.8 times performance improvement over IndexFS. The improvement mainly comes from partial consistency. Pacon allows write operations to return immediately after reaching the distributed cache. IndexFS is a centralized metadata service. Since IndexFS uses strong consistency in the client-side metadata cache so it cannot fully utilize the memory on the client nodes. For the read operations (stat), the throughput of Pacon is more than 2.6 times higher than IndexFS. The improvement is mainly because Pacon can read the metadata from the distributed cache but IndexFS may need to read them from the centralized metadata service. For the same reason in BeeGFS, IndexFS also cannot use client-side metadata cache efficiently in random stating.

B. Multi-application Case

In HPC systems, it is common for multiple applications to run simultaneously on the DFS. To simulate this scenario, we simultaneously ran multiple mdtest programs on different non-overlapping directories and evaluated the overall throughput. Each mdtest represents an application. The client cluster has 16 nodes, and each node contains 20 concurrent clients (totally 320 concurrent clients). We evaluated the performance of three common metadata operations (mkdir, create, random stat) with different numbers (from 2 to 16) of concurrent applications. The client nodes are evenly assigned to individual applications. For example, each application runs on 8 nodes when the number of concurrent applications is 2. In Pacon, each application is considered a consistent region. As in the previous experiment, IndexFS is deployed on each node in the client cluster.

As shown in Figure 8, Pacon's overall performance is better than BeeGFS and IndexFS: it is more than 1.07 times higher than IndexFS and more than one order of magnitude

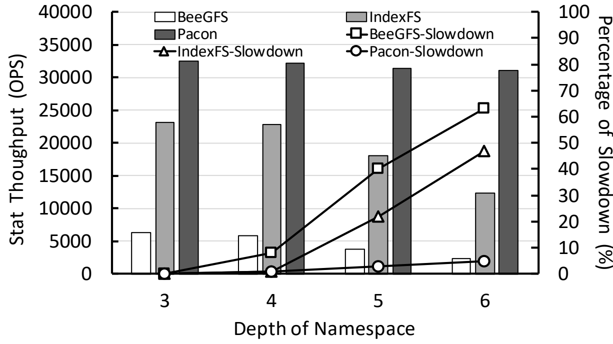


Fig. 9: Path Traversal Overhead.

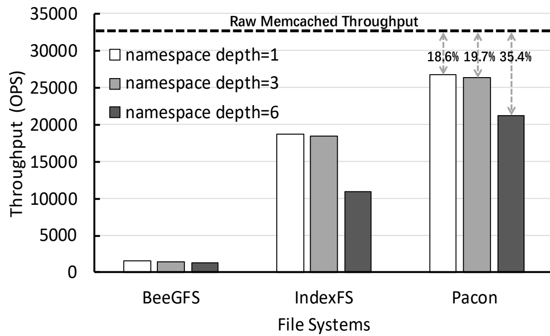


Fig. 10: Pacon Overhead.

higher than BeeGFS in these experiments. The performance improvement is mainly because Pacon has a higher throughput than other tested systems in the single-application case (as shown in the previous experiments). At the same time, partial consistency can isolate the metadata operations from different applications. In BeeGFS and IndexFS, operations need to frequently communicate with the centralized metadata service. But with Pacon, many operations only need to communicate with their distributed cache. This application-based isolation also helps group the metadata from the same application onto the running nodes of the application, thereby increasing the local metadata hits.

C. Path Traversal Analysis

Pacon uses batch permission management to avoid path traversal. This part shows the performance improvement brought by the batch permission management. We generated a namespace with different depth (from 3 to 6) and then evaluated the throughput of randomly getting directory attributes. Figure 9 shows that the stat throughput of BeeGFS and IndexFS decrease as the depth of the namespace increases. Their performance will be reduced by 63% (BeeGFS) and 47% (IndexFS) when the namespace depth is increased to 6. One of the main reasons is that the path traversal will incur a lot of network overhead. Although they cache the directory entries in clients, it is not useful in the case of random access. In Pacon, the depth of the namespace has only a slight impact

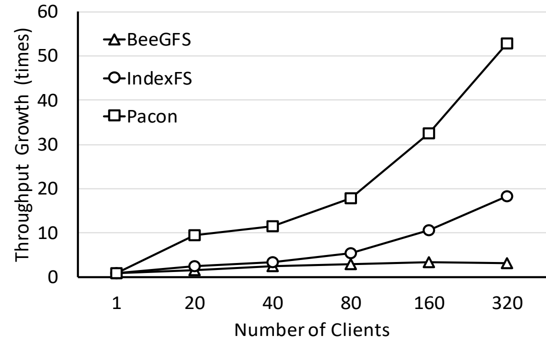


Fig. 11: Scalability.

on the performance, because Pacon can directly lookup the requested metadata.

D. Pacon Overhead

We evaluated Pacon's overhead by comparing the throughput of Pacon with raw Memcached (which is used as the distributed metadata cache in Pacon). We conducted this experiment without concurrency. For Pacon and other tested file systems, we ran mdtest with single client to create subdirectories under the same parent directory. The namespaces used in this experiment have a fanout of 5, but with different depths. For Memcached, we ran memaslap [19] with single client to evaluate the throughput of item insertion.

Figure 10 shows that the throughput of BeeGFS and IndexFS are much smaller than the in-memory KV, because they use local file system (BeeGFS) or on-disk KV system (IndexFS) to store metadata, which are more expensive than the in-memory KV system. In this experiment, Pacon can reach more than 64.6% throughput of the raw Memcached. The overhead of Pacon comes mainly from accessing the in-memory distributed KV store and pushing metadata operations into the commit message queue. Furthermore, in BeeGFS and IndexFS, path traversal may lead to amplification of metadata operations, that is, a metadata operation requires multiple network I/Os in these systems.

E. Scalability

This part shows the metadata scalability of different systems. We evaluated the throughput of file creation on BeeGFS, IndexFS, and Pacon. Each client node runs up to 20 clients, and each client concurrently creates empty files in the same parent directory. The number of client nodes increases as the number of clients increases (the number of nodes of Pacon and IndexFS will also increase). For example, when the number of clients is 20, the client cluster contains 1 node, and when the number of clients is 40, the client cluster contains 2 nodes, and so on. For each system, we normalized the results by the throughput in the single client case.

Figure 11 shows that Pacon has better scalability than BeeGFS and IndexFS. It is about 16.5 times and 2.8 times better than BeeGFS and IndexFS, respectively, when the

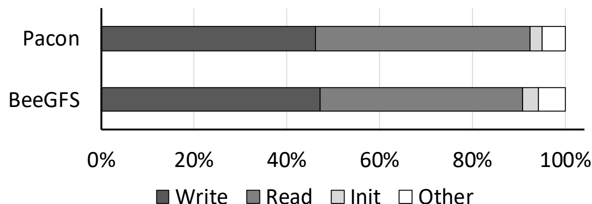


Fig. 12: Breakdown of MADbench2.

number of clients reaches 320. Compared with BeeGFS, Pacon servers run with the client nodes so it can dynamically adapt to the growth of the number of clients. Compared with IndexFS, although the number of IndexFS servers can also increase as the number of clients increases, it cannot fully utilize the client-side metadata cache, so the creation operations need to be performed on the central metadata service that is more costly than Pacon's distributed cache.

F. Real-world Application Benchmark

We used MADbench2 [18] to evaluate Pacon in the real-world HPC workload. MADbench2 is a benchmark implementation based on the behavior of MADspec, which calculates the maximum likelihood angular power spectrum of the cosmic microwave background radiation [20]. MADbench2 is used for testing the integrated performance of the I/O, computation, and communication subsystems under the stresses of a real scientific application. At the beginning, each MADbench2 process creates a file and generates (writes) the evaluation data into these files. Then, MADbench2 processes will read, write, and calculate the data in these files multiple times.

We ran MADbench2 on 16 nodes, each node has 16 working processes. In this experiment, a total of 256 files are created, each containing 4MB of data. We normalized the results by the duration of BeeGFS. Figure 12 shows that the overall runtime is almost the same on Pacon and BeeGFS, because this is a data intensive scenario. We also broke down the overhead by read, write, and initialization. Specially, the "init" part (initialization) mainly includes file creation overhead; the "other" part mainly includes computation and communication overhead. In the initialization phase, Pacon's file creation overhead is only slightly smaller than BeeGFS because this experiment is not a metadata-intensive scenario. Pacon and BeeGFS have similar read/write performance, because the file size (4MB) exceeds the small file threshold of Pacon so all read/write operations are redirected to the backend BeeGFS. This experiment shows that Pacon does not have a large impact on non-metadata operations.

V. Related Work

Central Metadata Service. There are many improvements to the central metadata service architecture. The most common way is to adopt a multi-MDS structure, which distributes metadata onto multiple metadata servers. Many DFSes support multi-MDS deployment (such as Lustre [9], CephFS [1], and BeeGFS [10]). Furthermore, CephFS [1] and GIGA+ [21] address the load balancing problem among multiple MDSes. To improve the scalability, some DFSes (such as IndexFS [3], LocoFS [7], and ShardFS [6]) flatten and store metadata on KV databases. All these related works based on the central architecture mainly improve the scalability of metadata service by extending metadata server cluster (MDS scalability), but they are difficult to scale when the number of clients is large (client scalability).

Private Metadata Service. In order to improve the client scalability, an extreme approach is to let the applications manage metadata themselves. BatchFS [4] is a DFS designed for batch jobs, a special version of IndexFS. In a batch job, metadata operations for all clients are independent (e.g., no access conflicts, no overlapping namespaces) and the files created by the batch job will not be accessed until after the batch job have done. BatchFS lets each client operate only on the local namespace snapshot during the batch job run, and then merges it into the master snapshot after the batch job is completed. For scenarios other than batch jobs, BatchFS requires additional mechanisms to handle issues such as overlapping namespaces, which increases the complexity of the system. DeltaFS [5] is a successor of BatchFS and is very similar to BatchFS's design, but it completely discards the global namespace. PFS-delegation [22] relieves metadata management bottleneck by offloading the space management to the application. These systems based on private metadata services are not general-purpose DFSes and will introduce additional namespace management overhead. Pacon has better versatility and manageability than these related works.

Configurable Metadata Service. Cudele [23] provides an API to allow user to customize the consistency and durability of the namespace. Different from Pacon, the strong consistency guarantee in Cudele is still provides by the centralized metadata service. Moreover, Cudele is not optimized for client-side metadata caching. Pacon and Cudele have different design goals: Cudele provides a simple solution for applications with different needs, and Pacon optimizes the consistency model of the client metadata cache.

VI. Conclusions

This paper addresses the scalability and efficiency of DFS metadata service with a new consistency model of client-side metadata cache. It proposes partial consistency, which splits the namespace into multiple consistent regions based on the workspaces of applications and guarantees strong cache consistency only inside the consistent regions for the applications. Therefore, Pacon can fully utilize the client-side metadata cache to improve the metadata scalability without sacrificing the versatility and manageability. In addition, it reduces the path traversal overhead of permission check in the distributed cache, which improves the efficiency of metadata processing. Pacon provides a library that can be utilized by existing DFSes to provide partial consistency. The evaluations in the paper show that Pacon can significantly improve the throughput and scalability of DFSes.

Acknowledgments

We appreciate the insightful comments from the anonymous reviewers. This work was supported by 2016YFB1000302 (National Key R&D Program of China), U.S. National Science Foundation CAREER award CNS-1619653 and awards CNS-1562837, CNS-1629888, IIS-1633381, CMMI-1610282, 61832020 (NSFC), U1611261 (NSFC), 61872392 (NSFC), 2016ZT06D211 (Program for Guangdong Introducing Innovative and Entrepreneurial Teams), 201906010008 (Pearl River S&T Nova Program of Guangzhou), 2018B030312002 (Guangdong Natural Science Foundation), and the China Scholarship Council.

References

[1] S. A. Weil, S. A. Brandt, E. L. Miller, D. D. Long, and C. Maltzahn, "Ceph: A scalable, high-performance distributed file system," in *Proceedings of the 7th symposium on Operating systems design and implementation (OSDI)*, 2006, pp. 307–320.

[2] gluster.org, "GlusterFS," https://www.gluster.org, 2019.

[3] K. Ren, Q. Zheng, S. Patil, and G. Gibson, "IndexFS: Scaling file system metadata performance with stateless caching and bulk insertion," in *Proceedings of the International Conference for High Performance Computing, Networking, Storage and Analysis (SC)*, 2014, pp. 237–248.

[4] Q. Zheng, K. Ren, and G. Gibson, "BatchFS: Scaling the file system control plane with client-funded metadata servers," in *Proceedings of the 9th Parallel Data Storage Workshop (PDSW)*, 2014, pp. 1–6.

[5] Q. Zheng, K. Ren, G. Gibson, B. W. Settlemyer, and G. Grider, "DeltaFS: Exascale file systems scale better without dedicated servers," in *Proceedings of the 10th Parallel Data Storage Workshop (PDSW)*, 2015, pp. 1–6.

[6] L. Xiao, K. Ren, Q. Zheng, and G. A. Gibson, "ShardFS vs. IndexFS: replication vs. caching strategies for distributed metadata management in cloud storage systems," in *Proceedings of the Sixth ACM Symposium on Cloud Computing (SoCC)*, 2015, pp. 236–249.

[7] S. Li, Y. Lu, J. Shu, Y. Hu, and T. Li, "LocoFS: A loosely-coupled metadata service for distributed file systems," in *Proceedings of the International Conference for High Performance Computing, Networking, Storage and Analysis (SC)*, 2017, pp. 1–12.

[8] X. Liao, L. Xiao, C. Yang, and Y. Lu, "MilkyWay-2 supercomputer: system and application," *Frontiers of Computer Science*, vol. 8, no. 3, pp. 345–356, 2014.

[9] P. Schwan *et al.*, "Lustre: Building a file system for 1000-node clusters," in *Proceedings of the 2003 Linux Symposium*, 2003, pp. 380–386.

[10] beegfs.io, "BeeGFS," https://www.beegfs.io, 2019.

[11] J. Lüttgau, M. Kuhn, K. Duwe, Y. Alforov, E. Betke, J. Kunkel, and T. Ludwig, "Survey of storage systems for high-performance computing," *Supercomputing Frontiers and Innovations*, vol. 5, no. 1, pp. 31–58, 2018.

[12] Google, "LevelDB," http://code.google.com/p/leveldb, 2018.

[13] E. Brewer, "CAP twelve years later: How the "rules" have changed," *Computer*, vol. 45, no. 2, pp. 23–29, 2012.

[14] LLNL, "mdtest," https://github.com/LLNL/mdtest, 2019.

[15] memcached.org, "Memcached," http://memcached.org, 2019.

[16] zeromq.org, "ZeroMQ," https://zeromq.org, 2019.

[17] W. Gropp, R. Thakur, and E. Lusk, *Using MPI-2: Advanced features of the message passing interface*. MIT press, 1999.

[18] J. Borrill, L. Oliker, J. Shalf, and H. Shan, "Investigation of leading HPC I/O performance using a scientific-application derived benchmark," in *Proceedings of the ACM/IEEE Conference on Supercomputing (SC)*, 2007, pp. 1–12.

[19] libmemcached.org, "memaslap," http://docs.libmemcached.org, 2019.

[20] LBL, "MADbench2," https://crd.lbl.gov/departments/computational-science/c3/c3-research/madbench2, 2014.

[21] S. Patil and G. A. Gibson, "Scale and concurrency of GIGA+: File system directories with millions of files." in *Proceedings of USENIX Conference on File and Storage Technologies (FAST)*, 2011, pp. 13–13.

[22] D. Arteaga and M. Zhao, "Towards scalable application checkpointing with parallel file system delegation," in *Proceedings IEEE Sixth International Conference on Networking, Architecture, and Storage (NAS)*, 2011, pp. 130–139.

[23] M. A. Sevilla, I. Jimenez, N. Watkins, J. LeFevre, P. Alvaro, S. Finkelstein, P. Donnelly, and C. Maltzahn, "Cudele: An API and framework for programmable consistency and durability in a global namespace," in *Proceedings of IEEE International Parallel and Distributed Processing Symposium (IPDPS)*, 2018, pp. 960–969.