

# BatchFS: Scaling the File System Control Plane with Client-Funded Metadata Servers

Qing Zheng, Kai Ren, Garth Gibson  
Carnegie Mellon University  
Email: {zhengq, kair, garth} @ cs.cmu.edu

**Abstract**—Parallel file systems are often characterized by a layered architecture that decouples metadata management from I/O operations, allowing file systems to facilitate fast concurrent access to file contents. However, metadata intensive workloads are still likely to bottleneck at the file system control plane due to namespace synchronization, which taxes application performance through lock contention on directories, transaction serialization, and RPC overheads. In this paper, we propose a client-driven file system metadata architecture, BatchFS, that is optimized for non-interactive, or batch, workloads. To avoid metadata bottlenecks, BatchFS features a relaxed consistency model marked by lazy namespace synchronization and optimistic metadata verification. Capable of executing namespace operations on client-provisioned resources without contacting any metadata server, BatchFS clients are able to delay namespace synchronization until synchronization is really needed. Our goal in this vision paper is to handle these delayed operations securely and efficiently with metadata verification and bulk insertion. Preliminary experiments demonstrate that our client-funded metadata architecture outperforms a traditional synchronous file system by orders of magnitude.

## I. INTRODUCTION

File systems define the interface between applications and the underlying storage provider. Unlike local file systems that sit directly upon a block device, most parallel file systems have adopted a layered architecture characterized by a dedicated metadata service layer backed by an object storage infrastructure serving as the persistence layer for both file data and file system metadata [1]–[4]. With namespace management decoupled from the data path, file system clients are able to stream data directly through individual object storage servers in a scale-out manner, allowing these backend servers to better utilize available hardware resources and maximize I/O bandwidth. Unfortunately, since file system clients are required to perform namespace lookups and undergo semantic checks before they can access file contents through the data path, metadata intensive workloads can still bottleneck at the file system metadata layer, which is typically designed as a centralized metadata server for ease of implementation [5]–[7].

To alleviate this bottleneck, modern file systems have been rebuilding their control planes with multiple metadata servers [3], [8]–[15], featuring techniques ranging from static namespace sharding or federation, to dynamic namespace partitioning. In addition, some of these file systems are able to perform directory splitting under different heuristics, representing different design trade-offs and facilitating fine-grained parallelism on their metadata paths [3], [12], [13], [16]. Notwithstanding a multi-server architecture, client-side metadata throughput can often be throttled by lock contention on common directories, server-side transaction ordering, and RPC overhead, let alone

other common performance issues such as load imbalance and skewed access patterns. As exascale data centers are starting to emerge, it can become even more challenging to achieve a scale-out file system for metadata-hungry applications.

A file system control plane consisting of multiple dedicated metadata servers is limited by the maximum metadata performance that this number of machines is able to deliver<sup>1</sup>. In order to better accommodate metadata-heavy massive-scale parallel HPC workloads, we propose a client-driven file system metadata architecture that allows applications to handle their own metadata operations locally mostly without server intervention. Unlike existing file systems that dedicate metadata server processes and machines to coordinate every metadata request in a centralized way, our file system can avoid inefficient RPC overheads and safeguards applications from unnecessary resource contention at the server side, effectively allowing the system to scale beyond a fixed sized control plane.

In addition, many file systems are designed for a worst-case scenario where applications rely on a strongly consistent file system to synchronize with each other. However, this is often overkill for batch applications, which are usually carefully programmed to perform tasks cooperatively with little external coordination. Our client-driven file system design exploits this opportunity and uses a relaxed consistency model to manage batch application metadata.

We envision a better file system interface that can adapt to coordinated metadata access by providing options for applications to batch metadata operations and delay semantic checks, allowing cooperating clients to pay for synchronization only when it is indeed necessary. Applications obtain capabilities to locally process namespace operations by linking to a user-level library file system, which enjoys direct access to file system metadata represented as a set of log-structured table-based data structures<sup>2</sup> stored in a shared underlying persistence layer [12], [17], [18]. Each job operates upon a file system snapshot and self-manages its namespace. By reusing server-side logic, applications are able to encode metadata mutations directly into on-disk metadata representations (as well as write-ahead log entries). When synchronization is eventually needed, a client process can submit its modified file system image to the global master image in order to merge updates. Part of this design, known as metadata bulk insertion, along with the

<sup>1</sup> Metadata servers are generally allocated entire machines because their in-memory state will consume all available resources.

<sup>2</sup> These data structures collectively represent a set of metadata mutation logs that are carefully grouped by common ACL settings in order to ensure high-level data privacy. See more details at Section II-D.

efficient underlying on-disk metadata representation, has been implemented in our previous work, IndexFS [12].

To secure global namespace semantics in a decentralized file system metadata layer, optimistic server-side namespace verification is used to establish a total order for all legitimate metadata operations bulk inserted by batch applications<sup>3</sup>. In order for this to work in a scalable way, we introduce auxiliary metadata servers, which are temporary daemon processes running on client resources with root privileges. Guarded by trusted virtual machines, these worker servers are responsible for verifying untrusted client-side file system mutations and committing them into the global master image. In addition, we envision an efficient mechanism for clients to pre-compute proofs of the correctness of their file system mutations. With these proofs, server-side namespace verification may be effectively simplified as a process of proof validation<sup>4</sup>.

By reorganizing the file system control plane into a set of decoupled and client-based metadata processing endpoints, we envision a highly scalable file system metadata architecture, BatchFS, which extends our previous work, IndexFS [12]. IndexFS features a log-structured on-disk metadata representation (Section II-D) and a simplified implementation of metadata bulk insertion (Section IV). This vision paper makes the following contributions: 1) a re-designed file system control plane harnessing client resources to achieve scalability, 2) deferred namespace synchronization enabled by efficient file system snapshots and fast metadata bulk insertion, 3) optimistic server-side metadata verification enabled by client-generated proofs, and 4) weak file system semantics targeting batch (or cooperating) applications. While we have only partially implemented BatchFS, we present its high-level design and show our preliminary results in the rest of this paper.

## II. DESIGN OVERVIEW

In general, interaction among a set of processes comes in two forms: 1) interaction among the processes of an integrated job or framework, and 2) interaction between unrelated jobs or systems, which, unfortunately, specifically use the file system as a synchronization platform. We refer to file system clients that demonstrate the former kind of interaction as *batch* clients, and the latter as *interactive* clients or *non-cooperating* clients.

### A. Assumptions

In designing BatchFS, we have been guided by common metadata access patterns and execution environments shared by batch applications in HPC data centers.

- Batch workloads usually only access a pre-constructed set of files for input, and normally generate their output files in a way that deterministically avoids name conflicts.
- Batch workloads are characterized by a natural preference for high throughput as opposed to low latency. Few place stringent response time requirements on individual metadata

<sup>3</sup> Applications anticipating or experiencing substantial interference from other non-cooperating processes can fall back to synchronous (but potentially slow) access on dedicated metadata servers.

<sup>4</sup> We focus on high-level namespace integrity, which is stronger than the low-level data integrity that is secured by the underlying storage infrastructure. Namespace verification will be discussed further in Section II-F.

requests, provided overall throughput is high enough.

- Files generated by batch jobs rarely get accessed until after these batch jobs have completed execution. There is a special case, however, for job owners monitoring output files in order to enable online user steering and fast detection of wasted resources; we will deal with this case separately in Section III.
- Batch jobs normally protect themselves against failures by stop-and-copy checkpointing, backing up their in-memory state into an underlying cluster file system before moving to the next stage. In this paper, we focus on one-file-per-process (N-N) checkpointing because one-file-per-job (N-1) checkpointing can be transformed into N-N checkpointing using a user-space translation layer such as PLFS [19].
- Most HPC data centers are equipped with dedicated storage infrastructures hosting large-scale cluster file systems, which are optimized for maximum data bandwidth but often lack a scalable metadata path [20].

### B. Interfaces

BatchFS targets the standard POSIX API. Applications invoke BatchFS routines with BatchFS's client-side user library or indirectly through FUSE [21] or MPI-IO [22]. Like its predecessor, IndexFS [12], BatchFS handles namespace related operations<sup>5</sup> but redirects all data requests to a specific underlying storage infrastructure, which holds both file contents and namespace metadata images generated by BatchFS. Different from IndexFS, BatchFS allows its clients to dynamically select between different namespace synchronization modes. This, along with its new consistency model, is discussed further in Section II-E.

### C. Architecture

A BatchFS cluster is organized as a metadata control plane layered on top of a scalable storage infrastructure serving as the underlying data plane. Rather than only having a fixed set of dedicated metadata servers like the original IndexFS, BatchFS's unique control plane features three different types of metadata processing engines: *primary* metadata servers, *auxiliary* metadata servers, and *private* metadata servers, illustrated in Figure 1.

Primary metadata servers are dedicated servers running on dedicated server nodes. These non-scaling<sup>6</sup> servers collectively manage the master image of the file system, with each server hosting a non-overlapping set of directory partitions [12], [16]. Interactive clients communicate with primary metadata servers to update the namespace synchronously, obtaining a latest view of the global namespace, albeit without scalable performance.

Both auxiliary and private metadata servers are designed to be temporary metadata processing entities that run on-demand

<sup>5</sup> In fact, as an optimization, data operations against small files (files smaller than 64KB for example) can also be served directly from BatchFS, taking advantage of its high efficient metadata processing backbone [12], [17].

<sup>6</sup> While some file systems, like IndexFS, can add additional dedicated servers, this is a slow, physical provisioning action done by an administrator. We consider this non-scalable because it is so slow and unlikely to be triggered by the needs of specific jobs.

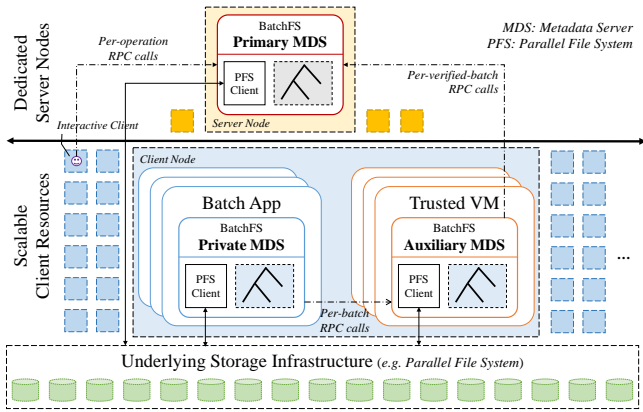


Fig. 1. *BatchFS is designed as file system metadata middleware layered on top of an existing cluster file system or an object storage platform exposing a flat namespace, which allows BatchFS to reuse the data path offered by these underlying storage substrates already optimized and tuned for maximum bandwidth. BatchFS features a client-driven metadata architecture that can shift server computation to client machines to achieve highly agile scalability.*

using client resources. We call these “client-funded metadata servers”. BatchFS library code linked into each application constitutes a private metadata server, which can be viewed as a full-fledged but untrusted metadata server associated with a file system snapshot and subject to access control provided synchronously by the underlying scalable data plane. These embedded metadata servers enable batch clients to access and modify their private namespaces locally without contacting any primary metadata servers, until they want their local mutations to be visible to non-cooperating jobs.

Auxiliary metadata servers are trusted daemon processes responsible for merging client-side namespace mutations into the global master image. Outsourcing this potentially heavy computation to auxiliary metadata servers shields primary metadata servers from becoming a performance bottleneck, and leads to better utilization of the underlying system.

#### D. Metadata Representation

In IndexFS, the global file system namespace is represented by a set of large directory entry tables with embedded inodes and possibly embedded file contents as well. Each file is mapped to a unique row inside the table. When flushed to disk, each table will be transformed into a set of log-structured table-based data structures formatted as *SSTables* (Sorted String Table). *SSTables* [23], [24] are immutable data containers that are partially ordered so that newer table entries always supersede older ones. *SSTable* serves as the physical format for metadata migration and bulk insertion [12], [17]. In addition, a tree of *SSTables* form a file system snapshot. BatchFS extends IndexFS and inherits its efficient metadata processing engine to support high-performance metadata mutation, migration, and bulk insertion.

As a new feature, BatchFS allows batch clients to request snapshots (collections of *SSTables*) of the current file system image, which they can use to establish a private copy of the snapshot namespace. *SSTables* no longer referenced in any file system snapshot can be safely discarded. In addition, files marked deleted in all *SSTables* can be purged from the underlying storage infrastructure without leaving null pointers.

To generate a snapshot for a requesting client, a primary metadata server performs an in-memory cache flush and sends the client a manifest that lists all *SSTables* comprising the current file system image. To prohibit a malicious client from accessing restricted data by scanning an entire file system snapshot inappropriately, BatchFS generates separate *SSTables* for each user-group combination so that every file referred in an *SSTable* has the same permission bits<sup>7</sup>. This way, high-level access control can be directly enforced by the underlying storage infrastructure. Note that this is designed for environments with simple ACL practices, which we expect in most HPC data centers. Similarly, user quota control is also synchronously enforced by the underlying storage infrastructure. For ease of implementation, quota management may only apply to file data, as the size of metadata is almost always dwarfed by the size of data in the file system as a whole.

#### E. Consistency Model

BatchFS features a relaxed consistency model targeting HPC applications. BatchFS clients associated with different snapshots of the file system can observe distinct namespaces. Interactive clients communicating with non-scaling primary metadata servers are kept in sync with the master image of the file system, which reflects all committed operations executed at the global namespace. Any individual metadata operation accepted by a primary metadata server is immediately committed, with its consequence immediately becoming visible to all subsequent file system calls and snapshots. On the other hand, batch workloads associated with local private metadata servers are logically disconnected from primary metadata servers. In fact, each batch client operates within its own file system image that originates from a specific snapshot of the master image. Metadata operations accepted by a private metadata server are immediately executed in the private namespace albeit remain uncommitted and invisible to other parallel file system clients, until an auxiliary server validates and publishes these changes.

Initially, BatchFS clients act as interactive clients, transmitting every metadata operation to a primary metadata server for global execution. Batch applications wishing to exploit BatchFS’s fast “asynchronous” metadata interface explicitly call BatchFS to establish their own private namespaces, converting themselves to *batch* execution mode (possibly for only a subtree of the overall namespace). BatchFS clients can switch back to the original *synchronous* execution mode whenever server coordination is needed. This entails closing the current private namespace, flushing changes to the underlying storage infrastructure, starting a new auxiliary metadata server, and asking it to merge the corresponding namespace mutations into the global file system image. We expect most batch applications to stay in *batch* execution mode until they complete a checkpoint or terminate.

#### F. Namespace Verification

To protect namespace integrity in the presence of untrusted clients, changes to a client namespace must be verified by an auxiliary metadata server before being inserted into the global

<sup>7</sup> This means a client with permission to read a directory must also be allowed to read the attributes (and embedded file data, if any) of the files under that directory.

namespace<sup>8</sup>. In BatchFS, any client namespace re-producible from a legal sequence of file system operations is considered a legitimate candidate for namespace merging. Unfortunately, in practice, it could be hard for an auxiliary metadata server to guess the valid operation sequence that happens to yield a given client namespace. To resolve this problem, BatchFS requires every client to submit, in addition to the resulting namespace changes, a “proof” of the correctness of its namespace mutations. As such, an auxiliary metadata server should be able to efficiently verify a client namespace by validating its associated proof.

A proof can be easy to construct (but unnecessarily large) if it is provided directly as the original sequence of operations executed at the client side. With such a proof, an auxiliary metadata server can perform checks simply by re-executing those operations and comparing results. To reduce the proof size, clients can reorder and merge commutative and associative file system operations. This leads to reduced verification computation, allowing client mutations to be checked faster. In the extreme, we envision a logic-based namespace certification process where clients provide formal proofs that their namespace changes respect the file system’s semantics. An auxiliary metadata server could validate these proofs without re-executing any file system operations, without using cryptography, and without consulting any external trusted entities. Once the validation succeeds, the associated namespace can be safely trusted and accepted. We refer to this as self-provable metadata representation, inspired by similar techniques in other contexts [25], [26].

To reconcile conflicts between global file system semantics and asynchronous namespace management, BatchFS employs optimistic concurrency control [27] commonly seen in database systems to order metadata operations. That is, clients do not do two-phase locking; they optimistically assume there will be no conflict. However, instead of using the generic notion of read/write set to verify its “transactions”, BatchFS applies file system semantics to verify namespace integrity and consistency. For example, a file creation can be reordered with a `chmod` if it is compatible with both the before and after permissions [28]. This resists a batch of file system operations from being unnecessarily rejected. In addition, different from a traditional database, BatchFS allows a batch of client-side namespace mutations to fail partially during the verification phase and does not necessarily roll back the whole “transaction”. This avoids a minor infraction from destroying a potentially large amount of work. Rejected namespace mutation notifications are available to users in external logs and associated files may be retained with mechanically modified names by BatchFS for users to resolve conflicts later.

### III. ADAPTATION OF JOB MONITORING APPLICATIONS

BatchFS’s design is in part motivated by the compatibility of a relaxed consistency model with batch applications. However, weak consistency could introduce problems for job monitoring utilities, which normally rely on strongly consistent file systems to achieve live user feedback, such as progress

<sup>8</sup> This verification and commit process can be delayed (potentially forever) until the first non-cooperating access, or delayed verification may be processed in the background in order to reduce (eventual) access latency.

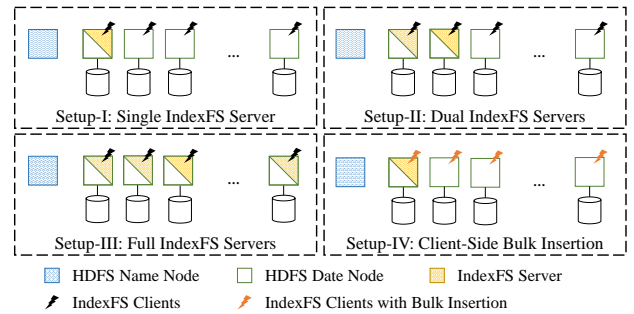


Fig. 2. *IndexFS with four different setups to model different amount of server resources. All our machines are from the NSF PRObe Kodiak cluster [30] and are configured with dual 2.6 GHz AMD Opteron processors, 8 GB of memory, two 1 TB 7200 rpm SATA disks, and a 1000 Mbps Ethernet NIC, with each running 64-bit Ubuntu 12.04 upon Linux 3.2.16. Note that this configuration was set up for ease of testing, real clusters often use dedicated hardware for the storage infrastructure and their file system servers.*

indication and data preview, because they appear to be non-cooperating processes.

In practice, live feedback, progress indication in particular, can be estimated via a proxy, such as the size of output files generated by a batch job, which can in turn simply be a stale number as opposed to an up-to-date value [29]. Based on this observation, BatchFS includes a separate metadata interface targeted at monitoring utilities. Designed with weak consistency in mind, this special interface provides applications with “metadata snapshots” (snapshots of file size for example) that do not necessarily represent a latest value but are designed to return newer versions under a sequence of repeated file system calls. With this interface, progress monitoring can be efficiently implemented without forcing batch jobs to perform unnecessary synchronization, thus ensuring high performance.

### IV. MEASUREMENTS

In this section, we report experiments done on our previous work, IndexFS [12], to show the promise of BatchFS’s file system metadata architecture. As BatchFS’s predecessor, IndexFS allows its clients to complete file creation operations locally if these files are known to be new, and will later bulk insert all these cached operations into the global namespace using a single SSTable insertion. This feature, a simple metadata bulk insertion, is extended by BatchFS to provide additional features such as ACL-specific file system snapshots, self-provable metadata representation, client-funded auxiliary metadata servers, and optimistic concurrency control. As a result, performance measurements of IndexFS showed in this section can be viewed as an optimistic projection of future BatchFS performance.

To show the efficiency of IndexFS’s metadata bulk insertion, a consistent client-side metadata write-back cache, we ran our experiments with this cache mechanism either enabled or disabled. More specifically, we measured performance on an IndexFS cluster where IndexFS was layered upon HDFS [31]. It consisted of 8 HDFS data nodes and 1 HDFS metadata node (an HDFS name node) and was configured with four different settings, as is illustrated in Figure 2.

To model metadata-intensive applications, we used a synthetic micro-benchmark tool, `mdtest` [32], to insert zero-byte

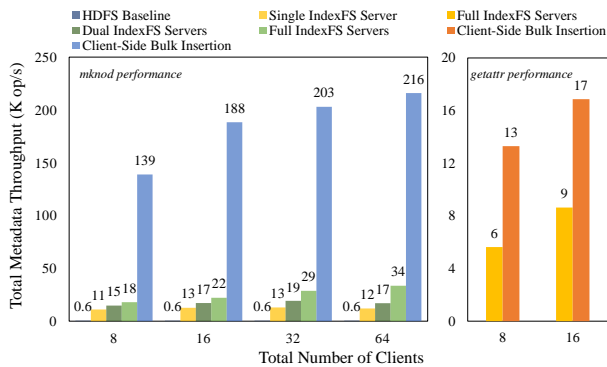


Fig. 3. With bulk insertion, BatchFS’s predecessor, IndexFS, is able to outperform HDFS by 360x and IndexFS without bulk insertion by as much as 18x. Bulk insertion also results in better metadata read performance, but metadata footprint is too large for 100% cache hits and `getattrs` may have to wait on HDFS data nodes.

files into multiple newly created directories. We generated a two-phase workload. In the first phase,  $N$  clients each create one private directory and then insert files into that directory. In the second phase, each of these clients performs `getattr` in a random order on the files created under its own directory. In total,  $M$  million files will be created and stat’d, where  $M$  is the total number of IndexFS servers or bulk insertion clients. The total number of clients,  $N$ , varies from 8 to 64.

Our experimental results are illustrated in Figure 3. In general, with the presence of a client-side metadata write-back cache, clients are able to drive an 800% to 36,000% increase in file creation throughput, in addition to better metadata read performance. This can in part be attributed to the faster metadata path, the more aggressive usage of client resources, as well as a local in-memory metadata cache provided by their embedded metadata processing modules. Moreover, compared with synchronous metadata processing that must be coordinated by a set of centralized metadata servers, allowing clients to self-manage their private namespaces indeed leads to better utilization of the underlying hardware resources, since the same system is delivering much higher throughput when bulk insertion is activated.

## V. RELATED WORK

Serverless file systems [8], [33] are often characterized by a set of symmetric file servers that are each capable of serving the whole file system namespace. This architecture is used by BatchFS’s primary and client-funded auxiliary metadata servers to achieve both high scalability and high efficiency. To improve write performance, PLFS [19], [34] devised a library-based user-level index structure capable of shaping I/O access patterns and aggregating small files, which allows it to avoid performance bottlenecks at the underlying cluster file system. PLFS inspired our approach to improving speed by decoupling sharing where possible and showed us where read performance would be of concern [35]. BatchFS’s client-side metadata write-back cache can also be used to buffer small files. In addition, BatchFS’s log-structured table-based on-disk metadata representation is general purpose and integrated with the file system.

RPC is an important but performance limiting component in distributed file systems. Mercury [36] is an efficient RPC

subsystem optimized for HPC data centers. It uses the traditional TCP/IP based network channel to send control messages but redirects bulk data to dedicated RDMA channels for fast transfer. BatchFS features a similar optimization mechanism that separates the file system control plane and data plane. In addition, BatchFS uses SSTables stored in the shared underlying storage infrastructure as a virtual communication channel to enable efficient metadata access and bulk insertion.

Modern HPC file systems often use dedicated I/O nodes for integrated data buffering and I/O forwarding [37], [38]. These techniques are orthogonal and complementary to BatchFS, which could utilize them for its private, auxiliary, or primary metadata servers.

BatchFS employs optimistic concurrency control protocol [27] commonly seen in data-intensive and database systems to increase concurrency. However, BatchFS does not enforce transaction atomicity as it allows a batch of metadata operations to fail partially during verification without rollback. In addition, BatchFS, like other weakly consistent systems [39]–[41], allows its applications to later merge conflicts and retry specific operations if the initial verification was rejected by the server.

Optimistic concurrency control and conditional operations have been used on file data in order to enable richer file system primitives for application coordination [42]–[44]. BatchFS is different from these techniques in that it focuses on metadata concurrency and targets batch applications that do not use the file system as a synchronization mechanism.

In order to secure auxiliary metadata servers, BatchFS relies on VMs to isolate these servers from adversary attacks. This can also be achieved through remote attestation, which is available in today’s software distributions [45], [46] as well as commodity security co-processors such as industry standard Trusted Platform Modules. Finally, BatchFS’s self-provable metadata representation is designed upon the observation that validating an answer is much simpler than solving the original problem [25], [26].

## VI. CONCLUSION

Metadata scalability is limited in today’s HPC parallel file systems because (1) it usually employs an RPC per operation; (2) its semantics require the status results of one operation to be known before the next is submitted; (3) its authorization enforcement requires dedicated (fixed number of) server machines; and (4) its durable representation of metadata is usually updated with random seeks for every mutation. With techniques for deferring and batching metadata operations, for reconciling concurrent weakly consistent mutations with optimistic concurrency control, for log-structured and indexed on-disk representation, for trusting code running in a user-contributed virtual machine, and for weakened semantics in high performance file systems, BatchFS proposes to allow the cooperating processes of a single job to pre-execute all of their own metadata operations on a snapshot of the file system namespace, presenting all resulting mutations back to the central file system in as few as a single batch, and providing a “proof” of the correctness and authorization of these mutations for optimistic concurrency control and selective reconciliation

as late as possible to maximize job independence and throughput. Moreover, BatchFS allows user jobs to allocate virtual machines running file system server code to scale metadata throughput with resources that local administrators would not want to dedicate to file system servers. To show the promise of this vision for scalable metadata services, BatchFS reports on a less expansive implementation of client-embedded metadata write-back caching, which shows 800% to 36,000% throughput improvement over today’s common implementations based on a single or a few dedicated metadata servers.

#### ACKNOWLEDGMENT

This research was supported in part by the DOE and Los Alamos National Laboratory, under contract number DE-AC52-06NA25396 subcontracts 161465 and 153593 (IRHPIT), the National Science Foundation under awards CNS-1042537 and CNS-1042543 (PRObe, [www.nmc-probe.org](http://www.nmc-probe.org)), and Intel as part of the Intel Science and Technology Center for Cloud Computing (ISTC-CC). We also thank the member companies of the PDL Consortium (Actifio, APC, EMC, Facebook, Google, Hewlett-Packard, Hitachi, Huawei, Intel, Microsoft, NEC, NetApp, Oracle, Samsung, Seagate, Symantec and Western Digital).

#### REFERENCES

- [1] D. Hildebrand and P. Honeyman, “Exporting storage systems in a scalable manner with pnfs,” in *MSST*, 2005.
- [2] P. Schwan, “Lustre: Building a file system for 1000-node clusters,” in *Linux Symposium*, 2003.
- [3] S. A. Weil, S. A. Brandt, E. L. Miller *et al.*, “Ceph: A scalable, high-performance distributed system,” in *OSDI*, 2006.
- [4] S. A. Weil, A. W. Leung, S. A. Brandt, and C. Maltzahn, “Rados: A scalable, reliable storage service for petabyte-scale storage clusters,” in *PDSW*, 2007.
- [5] S. Ghemawat, H. Gobioff, and S.-T. Leung, “The google file system,” in *SOSP*, 2003.
- [6] K. Shvachko, H. Kuang, S. Radia, and R. Chansler, “The hadoop distributed file system,” in *MSST*, 2010.
- [7] K. V. Shvachko, “Hdfs scalability: The limits to growth,” *login*, vol. 35, no. 2, pp. 6–16, 2010.
- [8] F. Schmuck and R. Haskin, “GPFS: A Shared-Disk File System for Large Computing Clusters,” in *FAST*, 2002.
- [9] B. Welch, M. Unangst, Z. Abbasi *et al.*, “Scalable performance of the panasas parallel file system,” in *FAST*, 2008.
- [10] P. H. Carns, W. B. L. III, R. B. Ross, and R. Thakur, “Pvfs: A parallel file system for linux clusters,” in *Linux Showcase and Conference*, 2000.
- [11] S. A. Weil, K. T. Pollack, S. A. Brandt *et al.*, “Dynamic metadata management for petabyte-scale file systems,” in *SC*, 2004.
- [12] K. Ren, Q. Zheng, S. Patil, and G. Gibson, “Scaling file system metadata performance with stateless caching and bulk insertion,” in *SC*, 2014.
- [13] “Giraffa: A distributed highly available file system,” <https://code.google.com/a/apache-extras.org/p/giraffa/>, 2013.
- [14] Y. Wang, J. Zhou, C. Ma, W. Wang, D. Meng, and J. Kei, “Clover: A distributed file system of expandable metadata service derived from hdfs,” in *CLUSTER*, 2012.
- [15] “Requirements for federated file systems,” <http://www.ietf.org/rfc/rfc5716.txt>.
- [16] S. Patil and G. Gibson, “Scale and concurrency of GIGA+: File system directories with millions of files,” in *FAST*, 2011.
- [17] K. Ren and G. Gibson, “TableFS: Enhancing metadata efficiency in the local file system,” in *USENIX ATC*, 2013.
- [18] J. Esmet, M. Bender, M. Farach-Colton, and B. C. Kuzmaul, “The TokuFS streaming file system,” in *HotStorage*, 2012.
- [19] J. Bent, G. Gibson, G. Grider, B. McClelland, P. Nowoczynski, J. Nunez, M. Polte, and M. Wingate, “PLFS: a checkpoint filesystem for parallel applications,” in *SC*, 2009.
- [20] S. Lang, P. Carns, R. Latham *et al.*, “I/o performance challenges at leadership scale,” in *SC*, 2009.
- [21] “FUSE,” <http://fuse.sourceforge.net/>.
- [22] P. Corbett and *et al.*, “Overview of the mpi-io parallel i/o interface,” in *Input/Output in Parallel and Distributed Computer Systems*. Springer, 1996, pp. 127–146.
- [23] F. Chang, J. Dean, S. Ghemawat, W. C. Hsieh, D. A. Wallach, M. Burrows, T. Chandra, A. Fikes, and R. E. Gruber, “BigTable: a distributed storage system for structured data,” in *OSDI*, 2006.
- [24] LevelDB, “A fast and lightweight key/value database library,” <http://code.google.com/p/leveldb/>.
- [25] G. C. Necula and P. Lee, “Safe kernel extensions without run-time checking,” in *OSDI*, 1996.
- [26] B. Braun, A. J. Feldman, Z. Ren, S. Setty, A. J. Blumberg, and M. Walfish, “Verifying computations with state,” in *SOSP*, 2013.
- [27] P. A. Bernstein and N. Goodman, “Concurrency control in distributed database systems,” *ACM Computing Surveys (CSUR)*, vol. 13, no. 2, pp. 185–221, 1981.
- [28] A. T. Clements, M. F. Kaashoek, N. Zeldovich, R. T. Morris, and E. Kohler, “The scalable commutativity rule: Designing scalable software for multicore processors,” in *SOSP*, 2013.
- [29] R. Ross, “Hec posix i/o api extensions,” [www.pdsi-scidac.org/docs/sc06/hec-posix-extensions-sc2006-workshop.pdf](http://www.pdsi-scidac.org/docs/sc06/hec-posix-extensions-sc2006-workshop.pdf).
- [30] G. Gibson, G. Grider, A. Jacobson, and W. Lloyd, “Probe: A thousand-node experimental cluster for computer systems research.”
- [31] HDFS, “Hadoop file system,” <http://hadoop.apache.org/>.
- [32] “mdtest: HPC benchmark for metadata performance,” <http://sourceforge.net/projects/mdtest/>.
- [33] T. E. Anderson, M. D. Dahlin, J. M. Neeffe, D. A. Patterson, D. S. Roselli, and R. Y. Wang, “Serverless network file systems,” in *SOSP*, 1995.
- [34] A. Torres and D. Bonnie, “Small File Aggregation with PLFS,” <http://permalink.lanl.gov/object/tr?what=info:lanl-repo/lareport/LA-UR-13-22024>, 2013.
- [35] J. He, J. Benth, G. Grider, G. Gibson, C. Maltzahn, and X.-H. Sun, “Discovering structure in unstructured i/o,” in *PDSW*, 2012.
- [36] J. Soumagne, D. Kimpe, J. Zounmevo, M. Chaarawi, Q. Koziol, A. Afsahi, and R. Ross, “Mercury: Enabling remote procedure call for high-performance computing,” in *CLUSTER*, 2013.
- [37] N. Ali, P. Carns, K. Iskra, D. Kimpe, S. Lang, R. Latham, R. Ross, L. Ward, and P. Sadayappan, “Scalable i/o forwarding framework for high-performance computing systems,” in *CLUSTER*, 2009.
- [38] N. Liu, J. Cope, P. Carns, C. Carothers, R. Ross, G. Grider, A. Crume, and C. Maltzahn, “On the role of burst buffers in leadership-class storage systems,” in *MSST*, 2012.
- [39] J. J. Kistler and M. Satyanarayanan, “Disconnected operation in the coda file system,” in *SOSP*, 1991.
- [40] D. B. Terry, M. M. Theimer, K. Petersen, A. J. Demers, M. J. Spreitzer, and C. H. Hauser, “Managing update conflicts in bayou, a weakly connected replicated storage system,” in *SOSP*, 1995.
- [41] G. DeCandia, D. Hastorun, M. Jampani, G. Kakulapati, A. Lakshman, A. Pilchin, S. Sivasubramanian, P. Vosshall, and W. Vogels, “Dynamo: Amazon’s highly available key-value store,” in *SOSP*, 2007.
- [42] S. Lang, R. Latham, R. Ross, and D. Kimpe, “Interfaces for coordinated access in the file system,” in *IASDS*, 2009.
- [43] P. Carns, R. Ross, and S. Lang, “Object storage semantics for replicated concurrent-writer file systems,” in *IASDS*, 2010.
- [44] P. Carns, K. Harms, D. Kimpe, R. Ross, J. Wozniak, L. Ward, M. Curry, R. Klundt, G. Danielson, C. Karakoyunlu, J. Chandy, B. Settlemyer, and W. Gropp, “A case for optimistic coordination in hpc storage systems,” in *PDSW*, 2012.
- [45] T. Garfinkel, B. Pfaff, J. Chow, M. Rosenblum, and D. Boneh, “Terra: A virtual machine-based platform for trusted computing,” in *SOSP*, 2003.
- [46] E. G. Sirer, W. de Bruijn, P. Reynolds, A. Shieh, K. Walsh, D. Williams, and F. B. Schneider, “Logical attestation: An authorization architecture for trustworthy computing,” in *SOSP*, 2011.