Revisiting the Metadata Architecture of Parallel File Systems

Nawab Ali*, Ananth Devulapalli†, Dennis Dalessandro†, Pete Wyckoff†, and P. Sadayappan*
*Department of Computer Science and Engineering
The Ohio State University, Columbus, OH 43210 USA
Email: {alin, saday}@cse.ohio-state.edu
†Ohio Supercomputer Center
1224 Kinnear Road, Columbus, OH 43212 USA
Email: {ananth, dennis, pw}@osc.edu

***Abstract*—As the types of problems we solve in high-performance computing and other areas become more complex, the amount of data generated and used is growing at a rapid rate. Today many terabytes of data are common; tomorrow petabytes of data will be the norm. Much work has been put into increasing capacity and I/O performance for large-scale storage systems. However, one often ignored area is metadata management.**

Metadata can have a significant impact on the performance of a system. Past approaches have moved metadata activities to a separate server in order to avoid potential interference with data operations. However, with the advent of object-based storage technology, there is a compelling argument to recouple metadata and data. In this paper we present two metadata management schemes, both of which remove the need for a separate metadata server and replace it with object-based storage.

I. Introduction

Increasingly, computational efforts rely on the processing and generation of large data sets. It is not uncommon today for scientific applications to generate data of the order of many terabytes [1], [2]. The rate at which applications create and use large-scale data sets has spurred the design of petabyte-scale file systems which consist of potentially billions of files [3].

One of the current trends in file system research is a focus on the I/O throughput of large-scale file systems [4], [5], [6], [7]. The performance of metadata operations, such as querying file attributes, enforcing access control, and maintaining consistency, is often ignored. However, the performance of metadata operations is intrinsic to the scalability and overall capability of large-scale storage systems. Often dismissed as a negligible cost, the metadata load in reality can be a major bottleneck [8], [9], [10].

The recent introduction of the ANSI standard [11] for object-based storage devices (OSDs) offers new opportunities in file systems research. An OSD is a storage device that exports an object-based interface to applications rather than the block-based interface of traditional disks. OSDs manage the internal data layout decisions, implement user-defined data attributes, and also provide a fine-grained security policy. From the perspective of the file system, a file is no longer a linear array of blocks but rather an object, or a set of objects. By abstracting the layout and attribute specifics from file systems, OSDs can potentially simplify their design.

978-1-4244-4208-9/08/$25.00 ©2008 IEEE

While OSDs offer a richer operational interface than traditional disks, they do not fulfill all the requirements of a parallel file system. Previous work [12], showed that I/O operations can be moved from dedicated I/O servers to OSDs. In this current work we offload metadata operations (with the exception of directory operations) from a dedicated metadata server to OSDs. Moving the data and metadata operations to an OSD-based I/O sub-system, allows for a reduced number of servers and a simpler design of parallel file systems. This reduces the complexity associated with managing these systems and allows them to scale with respect to I/O bandwidth and storage space.

The main contribution of our paper is to make the argument that recoupling data and metadata using OSDs creates a more modular and scalable architecture. We present two novel ways of offloading metadata operations to an OSD-based I/O sub-system along with a PVFS-based implementation of our designs. Finally, we present results from microbenchmarks and applications. Even with the overhead associated with software OSDs, the performance of our OSD-based metadata management techniques is comparable to, and on certain workloads better than that of dedicated PVFS metadata servers.

II. Background

This section provides background information on the technologies and file systems we have used in our research.

A. Parallel Virtual File System

PVFS [4] is a parallel file system designed to provide applications with high-performance I/O capabilities. A typical PVFS system consists of clients, I/O servers, and metadata servers. PVFS stripes a file across multiple I/O servers in parallel, thereby providing applications high-bandwidth access to the file system. The metadata server (MDS) handles the metadata operations such as access control, directory traversals, and file lookups.

B. Object-Based Storage Devices

Object-based storage devices (OSDs) are an extension of traditional block-based disks. Instead of representing data as a linear array of blocks, OSDs export data as objects. Another important distinction between an OSD and a traditional

block-based disk is the role of the operating system in data placement. In traditional block-based disks, the host operating system is responsible for block allocation and data placement. In contrast, with object-based storage, the disk abstracts the details of organizing the data on the storage medium from the operating system. Figure 1 shows this difference between block-based and object-based disks. By removing the responsibility of block allocation and data placement, it is possible to simplify the design of file systems. PVFS, for instance, uses dedicated I/O servers to handle writing data to disk. The introduction of OSDs eliminates the need for such I/O servers.

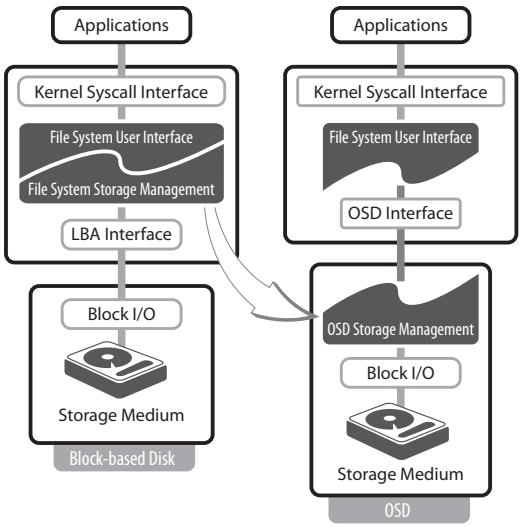

Fig. 1: Comparison of Block-based and Object-based storage models.

Currently, there are no commodity object-based disks available. However, a related work [12] provides a software OSD based on the OSD specification [11].

III. System Design

There are two main entities that make up the OSD architecture: the initiator which translates client requests into OSD commands, and the target which executes and responds to OSD commands. The target and the initiator communicate using the iSCSI protocol. The following sections explore the initiator and target of the OSD software implementation [12].

A. OSD Initiator

The initiator library exports the OSD command interface to clients. Initiators create and send requests, while the targets process and respond to those requests. According to the OSD specification [11], all client requests are translated into complex Command Descriptor Blocks. Due to this complexity, a generic initiator library is provided to help with the process. It is used either directly by clients or through middleware such as a file system. The implementation of the initiator library involved exploiting the new "bsg" interface in the Linux kernel for bi-directional SCSI commands as required by the OSD specification. Other kernel modifications include support for I/O vectors to minimize data copying and support structures for bi-directional commands in the SCSI mid-layer and iSCSI transport. Effort has been made to make the interface friendly to userspace applications.

B. OSD Target

The OSD target is made up of a number of components which include transport, iSCSI/SCSI layer, OSD command processor, attribute, and data management backends.

The OSD specification [11] defines an object as an ordered set of bytes associated with a unique identifier. Additionally, each object has a set of mandatory and user-defined attributes, or metadata about the object. Some of the predefined attributes are familiar from most file system designs, such as size and modification time.

The transport layer provides a generic interface for multiple transports like TCP/IP or InfiniBand. The iSCSI/SCSI layer deals with iSCSI session management and SCSI header parsing. The OSD command processing engine demarshals OSD commands, executes the commands, and hands over the marshaled response back to the iSCSI/SCSI layer, which is dispatched back to the initiator. The attribute management backend is implemented using SQLite [13], a light-weight database, for faster and scalable implementation of attributes. Details about design and implementation of the attribute storage is explored in a previous publication [14]. Finally the data storage backend relies on the underlying file system. Presently, the OSD target supports all of the object manipulation, I/O, attribute manipulation, and multi-object commands.

IV. Offloading MDS Operations to OSDs

One of the challenges in offloading metadata operations to the disk is identifying the correct storage paradigm. In this section we present two different OSD-based metadata storage techniques and examine their advantages and drawbacks.

Name	Size	Explanation
Uid	4 bytes	User id
Gid	4 bytes	Group id
Permissions	4 bytes	File permissions
Creation time	6 bytes	File creation time
Modification time	6 bytes	Last mod. time
Access time	6 bytes	Last access time
Distribution	Variable	Layout type
Data Handle Array	Variable	Data file handles

TABLE I: File system metadata components.

In our current PVFS implementation, there are 8 components to the metadata of a file, as shown in Table I. The first six are typical of any file system: owner and access permissions of the file and timestamps of file activity. The timestamps are supported natively by the OSD, and we need not explicitly update them, while the owner and permission information is specific to the file system. The last two attributes

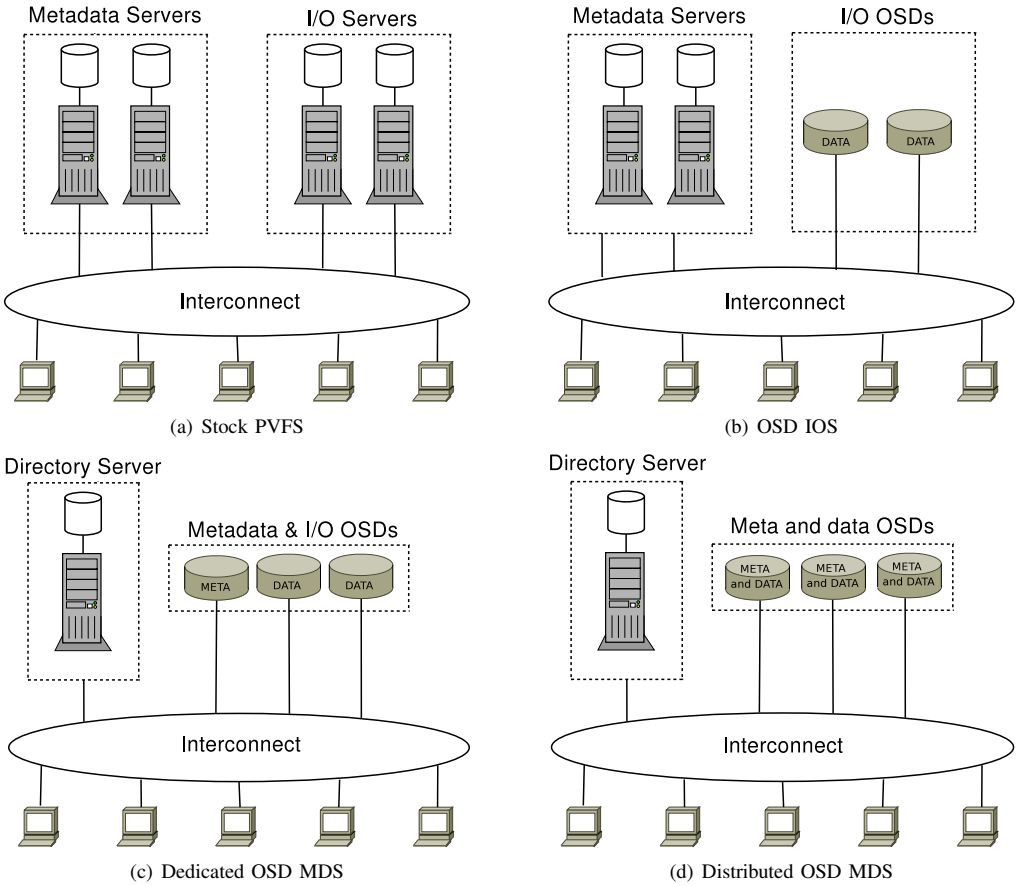

Fig. 2: Architecture of different metadata schemes.

contain information about the layout of the file contents. The *distribution* is a small structure that contains the name of a layout type along with its parameters; for instance, "simple_stripe" specifies that the bytes in the file are striped across the data servers. The data handle array is a list of handles (disk and object identifiers) that specify where the data referenced in the distribution is stored. Using this information, a client can directly access the disks that have the desired data.

A. Stock PVFS and OSD IOS

Figure 2 shows four different metadata distribution schemes. In the top-left is unmodified (or "stock") PVFS, which uses dedicated servers for storing both data and metadata. These are sometimes combined in practice, but presented separately here for clarity. Previous work [12] replaced the I/O servers with directly-attached disks, as is shown in the top-right corner of the figure. This change does not affect metadata, which is still stored in dedicated PVFS servers.

B. Dedicated OSD-based MDS

The simplest metadata offloading technique is to use a dedicated, OSD-based metadata server (MDS) for all metadata operations. Much like the transition that substituted OSDs for servers in the data storage path, this modification replaces PVFS metadata servers with dedicated metadata OSDs, except for directory operations. Figure 2 shows this change in the bottom-left. To accomplish this with an OSD-based storage system, we employ a single, dedicated OSD to handle the metadata operations. The MDS is essentially an OSD that stores the metadata as attributes of a zero-length file. While we could store the metadata information as the contents of an object, it is both easier to access and potentially faster to implement when stored as separate attributes of an object. Storing metadata as attributes avoids the overhead of parsing the file. It also enables the use of hardware-based database operations that may be available in hardware OSDs.

C. Distributed OSD-based MDS

One of the advantages of OSDs is the ability to store user-defined attributes along with data. This means that it is possible to store metadata as attributes of the data. In the previous design, attributes were used to store the metadata. However, they were attached to a zero-length object on a dedicated OSD MDS. Another alternative is to attach the metadata to one or more of the data objects. The bottom-right of Figure 2 shows the Distributed OSD MDS case.

In this design, file metadata is stored as attributes on the first stripe of data. The advantage of this technique is that

since the first stripe of data for different files is stored on different OSDs, we end up with more OSD metadata servers. Essentially all data storage devices are now metadata storage devices as well. This removes the need for separate dedicated disks or servers for metadata, and ensures that metadata operations are not bottlenecked by a single server. This also provides a natural load-balancing for the file system in terms of metadata operations, assuming that the way files are distributed mimics the way that metadata is accessed.

A limitation with this design is that under heavy load, it is possible that metadata and I/O operations can interfere with each other, adversely affecting both. The next section shows experiments that attempt to quantify this.

In this design we store the metadata only on the object that contains the *first* stripe of data. An alternate approach would be to replicate the metadata across multiple, or all, objects that contain file data. This would provide multiple servers to retrieve metadata information, but comes at the expense of changing many metadata update operations to $O(N)$ in the number of servers instead of $O(1)$.

D. Directory Operations

As shown in Figure 2, our current design still employs a separate directory server. The reason is that in order to provide consistency, directory operations such as entry creation and removal have to be atomic. This is because these operations are implemented by a series of smaller steps at the device, including searching for an existing entry before creating a new one. Most server-fronted file systems implement atomicity by providing some form of a locking sub-system. OSDs lack atomic operations, and do not provide any support for locking or transaction semantics. Further, an OSD cannot initiate communication, which complicates consistency enforcement. We are proposing extensions to the OSD specification [11] to include atomic operations. In the meantime we have used a regular PVFS MDS for all directory operations in our experiments.

Even though we use a dedicated PVFS MDS for directory operations, it has no influence on the results presented in this paper. The main argument that we make in this paper is that dedicated I/O and metadata servers are an artifact in parallel file system design and coupling data and metadata results in a modular storage system. This work studies what happens if we move non-directory metadata operations to OSDs. Related work [15] addresses the separate issue of handling directory manipulations on OSDs.

V. Microbenchmarks

This section describes the experiments used to evaluate our metadata offloading techniques. We present results from two microbenchmarks first, then present a model to explain the results and forecast trends. A later section presents metadata-intensive application performance results. In the experiments we evaluate the four configurations shown in Figure 2.

Our experimental testbed consists of 30 nodes in a Linux cluster running kernel version 2.6.22. Each node has dual AMD Opteron 250 processors, 2 GB RAM, and an 80 GB SATA disk. The nodes are connected by onboard Tigon 3 Gigabit Ethernet and an SMC 48-port Gigabit switch.

A. Stat Latency

Figure 3 shows the latency of a stat (left) and a create (right) operation as a function of the number of I/O elements. A PVFS metadata server is used for directory operations in this and all tests, having no effect on the differences among the results. The stat operation communicates with the metadata servers to find information such as file modification time, and also with all the storage devices to find the file size. The create operation creates multiple zero-length files on the file system.

As shown in Figure 3 (left), the stat latency for all the curves increases with the number of I/O elements. This is as expected due to the fact that the stat operation communicates with all I/O servers to find the file size. The Dedicated OSD MDS and the Distributed OSD MDS have almost identical latency curves because they both use OSDs for I/O and metadata. Stock PVFS has the lowest latency, 0.5 ms for 25 I/O servers. This is the result of using a simpler database (DB4) for storing metadata. Our OSD target uses SQLite for database operations. The overhead associated with SQLite along with the protocol processing time adversely affects the stat latency for all of the OSD variants.

The create latency curves for stock PVFS, OSD IOS and Dedicated OSD MDS have a similar slope because they are constrained by a single metadata server. The Distributed OSD MDS however has a lower create latency, around 1.6–1.7 ms, because it does not create an explicit metafile object during the create operation.

B. Create Throughput

Figure 4 shows the aggregate "create" throughput as a function of the number of clients. This microbenchmark creates multiple zero-length files on a file system with four I/O elements and a PVFS metadata server for directory operations. The idea is to study the effect of a metadata intensive operation on the metadata servers. On the left, we see the create throughput results for disk-based storage. The common factor among all the curves is that they plateau almost immediately. Only a handful of clients are required to saturate the create capacity of the servers.

The curves for stock PVFS and OSD IOS are similar because they both use a PVFS server for a MDS. The throughput in the case of a Dedicated OSD MDS is better than stock PVFS or OSD IOS because it uses two metadata servers. One is an OSD-based MDS for file operations and the other is a PVFS MDS for directory operations. The unique design of the Distributed OSD MDS (metadata is stored on all the I/O OSDs) avoids the bottleneck resulting from a single metadata server. As such, the mean aggregate create throughput for twenty clients is almost two and half times more than stock PVFS.

Figure 4 (right) shows the aggregate create throughput results for RAM-based storage. Mounting the file system in

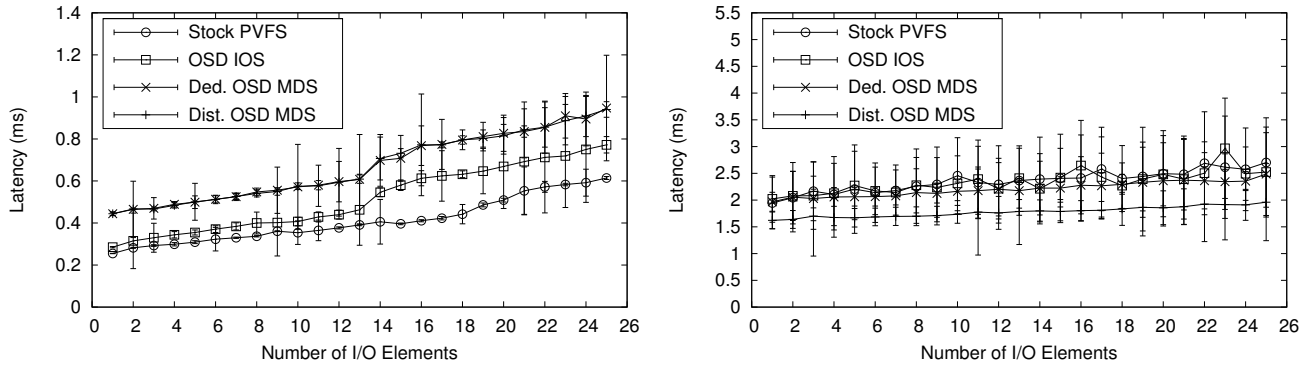

Fig. 3: Latency as a function of the number of I/O elements; left: PVFS stat; right: PVFS create.

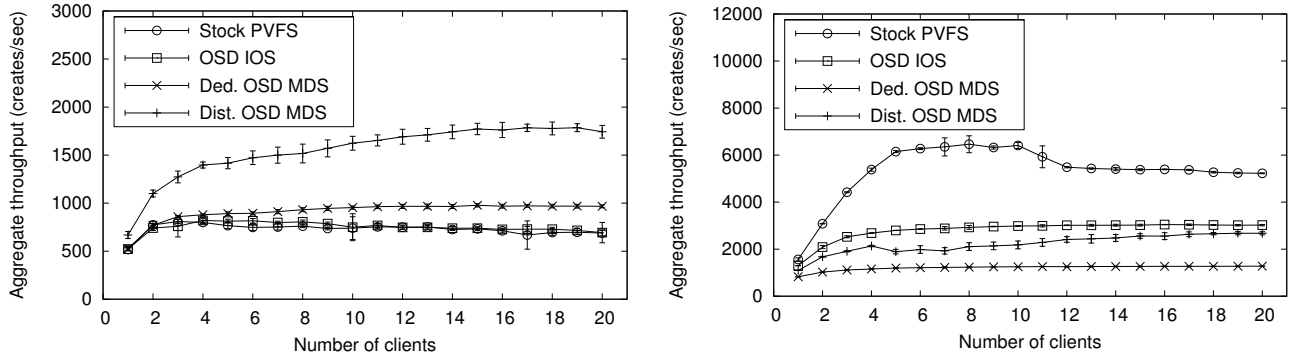

Fig. 4: Aggregate create throughput; left: disk-based storage; right: RAM-based storage.

memory allows us to mitigate the effects of disk seek time and also to analyze the OSD protocol processing costs. Stock PVFS outperforms the other three OSD variants in this test primarily due to multithreading and also because it uses a simpler DB4 database. While SQLite offers some distinct advantages over DB4, it also incurs a significant overhead. The Distributed OSD MDS performs better than the Dedicated one, again, because its design allows it to use multiple metadata servers.

VI. Scalability Model

In order to make projections for larger systems and to make performance predictions based on future technological trends, an analytical model is essential. Here we present a simple first order model based on our experiments for the create metadata operation using OSD IOS, Dedicated OSD MDS, and Distributed OSD MDS.

A PVFS client takes the following steps to create a file. First it selects a random metadata server and requests to create a metafile, receiving the corresponding metafile handle. Then it selects a subset of the I/O servers and creates datafiles, this time receiving the corresponding datafile handles. Next the client sets the attributes of the metafile handle, including the list of datafile handles. Finally it creates a directory entry for the newly created file on the metadata server hosting the parent directory of the file.

Term	Definition	Value
N_{md}	Number of metadata servers	1
N_{mo}	Number of metadata OSDs	1
N_{io}	Number of I/O elements	4
N_c	Number of clients	variable
T_{mfp}	Metafile create time on PVFS	242 μs
T_{mfo}	Metafile create time on OSD	318 μs
T_{dfo}	Datafile create time on OSD	318 μs
T_{sap}	Setattr time on PVFS	446 μs
T_{sao}	Setattr time on OSD	457 μs
T_{de}	Dirent create time on PVFS	419 μs
T_{net}	Network roundtrip	101 μs
C	Client overheads	3 μs

TABLE II: Definition and the values of the model variables.

With this process in mind we present the following equations to model the file create time for the three OSD metadata schemes:

$$T_{ios} = T_{mfp}+T_{dfo}+T_{sap}+T_{de}+4\times T_{net}+C$$
$$T_{ded} = T_{mfo}+T_{dfo}+T_{sao}+T_{de}+4\times T_{net}+C$$
$$T_{dis} = \quad T_{dfo}+T_{sap}+T_{de}+3\times T_{net}+C$$

Table II gives the meanings and experimentally derived values of the variables in the model. For example, in the case of OSD IOS, ignoring the constant term, the time to

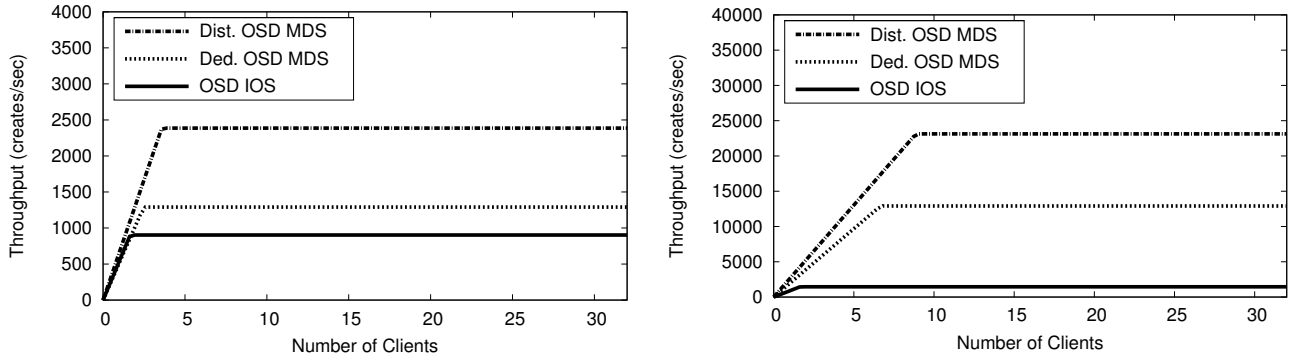

Fig. 5: Create throughput as a function of number of clients; left: performance using software OSDs; right: expected performance on hardware OSDs.

create a file (T_{ios}) is the sum of the times required to create a metahandle on a PVFS server (T_{mfo}), create datafiles on OSD I/O elements (T_{dfo}), set attributes on a PVFS server (T_{sap}), create the directory entry (T_{de}); and, the four network round trip times (T_{net}) required by these steps.

Given the models for the create operation, we can now model the create throughput for the three cases. For OSD IOS, the create throughput Th_{ios} is a function of the number of clients (N_c), metadata (N_{md}) and I/O servers (N_{io}). It is given as

$$Th_{ios} = \min\left(\frac{N_c}{T_{ios}}, \frac{1}{\max\left(T_{dfo}, \frac{T_{mfp}+T_{sap}+T_{de}}{N_{md}}\right)}\right).$$

The *min* expression has two terms. One is the upper bound on client side throughput, and the other is the upper bound on server side throughput. In the case of OSD IOS, the server bound is determined by the two time components: the time to create the datafiles in parallel on the I/O servers (T_{dfo}), and the times related to metadata operations ($T_{mfp}+T_{sap}+T_{de}$). These metadata operations can be distributed among multiple metadata servers, hence we factor by N_{md}. Of the time components, throughput is limited by the slowest of the two, and since throughput is the inverse of the time to create, the server bound for the create throughput is determined by the second term in the *min* expression. Analogously, from the clients' perspective, if we keep adding clients, a first order model for the throughput is given by the first term in the *min* expression. However, the overall throughput is determined by the minimum of the two components, giving the above expression.

The structure of the throughput expression in the case of the Dedicated OSD MDS (Th_{ded}) is similar to the OSD IOS:

$$Th_{ded} = \min\left(\frac{N_c}{T_{ded}}, \frac{1}{\max\left(T_{dfo}, \frac{T_{de}}{N_{md}}, \frac{T_{mfo}+T_{sao}}{N_{mo}}\right)}\right).$$

The main difference is in the expression for the upper bound on the server side. In this case, the server side limit is determined by the slowest of the following operations: the time to create datafiles on the OSDs (T_{dfo}), the time to create the metafile and set attributes on OSDs ($T_{mfo}+T_{sao}$) which can be distributed on multiple Dedicated OSDs and the time to create the directory entry on PVFS servers (T_{de}) which can be distributed across N_{md} metadata servers. The explanation for the rest of the expression follows the same logic as OSD IOS.

The throughput expression for the Distributed OSD MDS (Th_{dis}) has a similar structure:

$$Th_{dis} = \min\left(\frac{N_c}{T_{dis}}, \frac{1}{\max\left(T_{dfo}+\frac{T_{sao}}{N_{io}}, \frac{T_{de}}{N_{md}}\right)}\right).$$

The server bound in this case is determined by three terms: the time to create the directory entry which can be distributed across N_{md} metadata servers, the time to create the datafiles (T_{dfo}), and the time to set attributes (T_{sao}). As explained in Section IV-B, in the case of the Distributed OSD MDS, the datafiles are created on each OSD and attributes are set on one of them, hence the factor of N_{io}.

Figure 5 (left) shows the throughput curves for the three schemes based on the models above, using the values in Table II. Though the models predict the general shape of curves shown in Figure 4 (left), the actual values are not obtained. The curves for Dedicated OSD MDS and OSD IOS asymptote quickly like the experimental data since very few clients are needed to saturate the metadata servers. The curve for Distributed OSD MDS saturates quickly compared to the experimental data, because the model hits the upper bound imposed by directory entry costs. The reason for this mismatch is our simple first order model. Since we are interested in general trends, we ignore factors such as server-side caching effects, data size increases, and pipelining. We also assume that each time variable does not vary with respect to N_c, which is not accurate.

A. Expected Performance on Hardware OSDs

The models are parametrized based on values obtained using the software OSD implementation. In the case of real OSD disks, we expect the overheads associated with the emulation to be reduced due to multiple factors, such as the use of custom ASICs and content-addressable RAM for attribute operations,

a significant cache of FLASH memory, pipelining and fine-grained hardware parallelism. In that case we expect the times associated with the OSDs (T_{dfo}, T_{sao}, T_{mfo}) to be reduced to roughly 10% of the values listed in Table II. If we take this into account, we observe that all three curves are limited by PVFS server costs (T_{mfp}, T_{sap}, T_{de}). For example, in the Distributed OSD MDS case, reducing OSD times alone would have no effect as it is already limited by directory entry costs. Figure 5 (right) shows the expected throughput using times reduced to 10% for OSD operations, and eliminating directory operation times, which are identical in all three cases. This shows the expected improvements by distributing metadata and by placing it directly on storage devices instead of going through a server. However, critical to obtaining this performance is the need to address the issue of directory entry management. This will be addressed in future work.

VII. Application Performance

We now present performance data gathered on applications. First we show a checkpoint program, which consists of the kernel of a checkpoint operation commonly found in many large scale parallel applications. We follow the checkpoint analysis with the Scalable Synthetic Compact Application (SSCA) benchmark.

A. Checkpoint

Most long-running parallel codes write their system states periodically to checkpoint files. This enables the user to restart the application from the previous checkpoint in the event of a system or application crash. The checkpoint operation is metadata intensive. Each process running the benchmark writes to an individual checkpoint file.

The total number of checkpoint files created in a single run is the product of the number of clients and the total number of iterations. The size of the data stored by each client at each iteration is another free parameter.

Figure 6 shows the checkpoint throughput results as a function of the size of the parallel job for different checkpoint file sizes (32 kB, 256 kB). We used four I/O servers and a single PVFS metadata server for directory operations.

With the exception of stock PVFS, the file throughput scales with the number of clients for each of the file systems. Stock PVFS tends to perform poorly for small message sizes. This is due to a default tuning that favors very large messages. Thus, the PVFS performance plateaus or degrades as we increase the number of clients. Distributed OSD MDS performs better than the Dedicated OSD MDS because it can use I/O servers for metadata operations. Even though OSD IOS uses a single MDS, its performance is comparable to that of Distributed OSD MDS. This is because by using a regular PVFS MDS, OSD IOS can avoid the overheads of a software-based OSD, such as the slower database, and increased protocol processing time.

In the next set of experiments, we varied the number of I/O elements for a fixed number of clients (8). Figure 7 shows the checkpoint throughput results as a function of the number of I/O elements. The curves of the OSD-based file system plateau quickly and then degrade as we increase the number of servers. In the 256 kB case, Distributed OSD MDS performs better than the other schemes for reasons outlined above. The 32 kB results are similar with the exception that the checkpoint performance degrades slightly with the number of servers. This is due to the fact that increasing the number of I/O elements requires more time to create the datafiles, as the operation is not perfectly parallel.

B. Scalable Synthetic Compact Application

The Scalable Synthetic Compact Application (SSCA) [16] Benchmarks are a set of pseudo-real applications that closely mimic high-performance computing workloads. We have used the file I/O component of SSCA-3 (Sensor Processing, Knowledge Formation and File I/O) to evaluate our metadata offloading techniques.

The SSCA-3 application creates a large number of files and is I/O and metadata intensive. We have modified the application to issue MPI-IO calls instead of using the POSIX interface (fopen, fread, fwrite). This change makes it easier to evaluate PVFS performance without the overhead of the kernel interface, and should have minimal impact on metadata operations. We also removed an odd behavior in SSCA-3 where it would break all read and write operations into loops of 4-byte reads and writes; that is, even though the application kernel produces reasonably sized operations, a subroutine breaks them into a series of 4-byte reads or writes. We used four I/O servers and a single client for all the test runs.

Figure 8 (left) shows the execution time of the SSCA-3 application for the *test1* run, as described in the software package. This configuration creates about 5000 files and writes almost 1 GB of data. Since the *test1* run is not metadata intensive, stock PVFS outperforms the other OSD-based metadata solutions which are limited by overheads in the software OSD.

Figure 8 (right) shows the results for the SSCA-3 *test7GB* run. This more interesting configuration creates about 95000 files and writes almost 7 GB of data. With the exception of OSD IOS, the average execution time of stock PVFS and the other OSD variants is almost the same (about 77 minutes). While stock PVFS had outperformed the others during the *test1* run, its performance is comparable to the other metadata offloading techniques during the *test7GB* run. This result validates our hypothesis that current approaches to distributed metadata do not scale sufficiently well for some real workloads.

We were expecting the Distributed OSD MDS to have a lower execution time for this metadata-intensive workload since it uses multiple metadata servers. However, *test7GB* is also data-intensive, so the benefits of using multiple OSDs as metadata servers is lost because of the overheads (slower SQLite DB and protocol processing time) associated with performing I/O on OSDs implemented in software. In using a hybrid deployment (PVFS MDS for metadata operations, OSDs for I/O), OSD IOS strikes a happy medium. It avoids

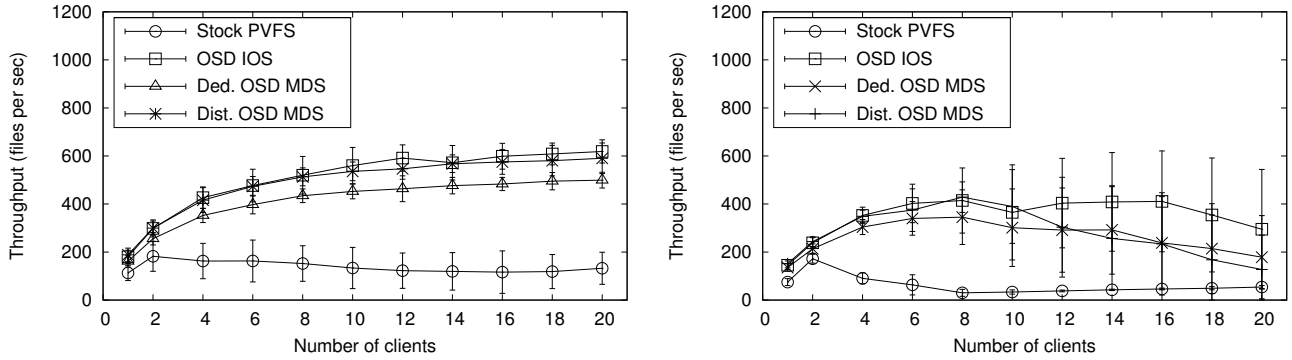

Fig. 6: Checkpoint throughput as a function of number of clients; left: checkpoint size = 32 kB; right: checkpoint size = 256 kB.

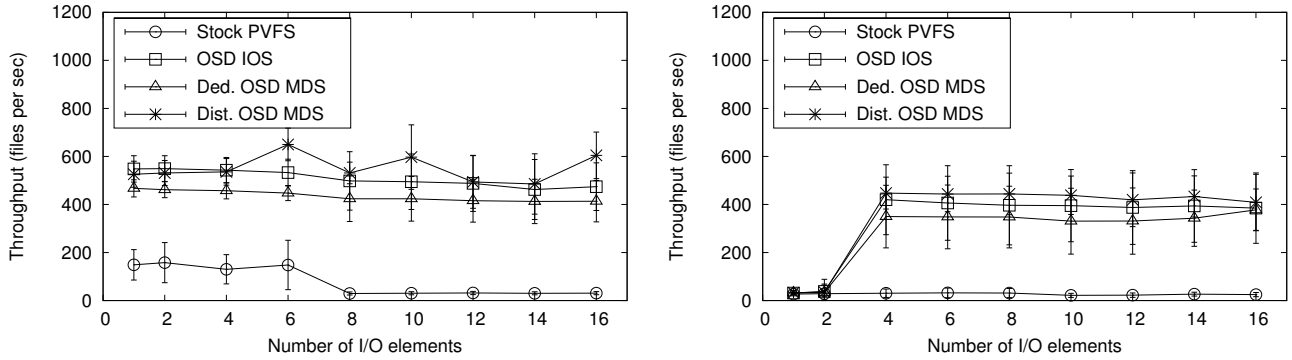

Fig. 7: Checkpoint throughput as a function of number of I/O elements; left: checkpoint size = 32 kB; right: checkpoint size = 256 kB.

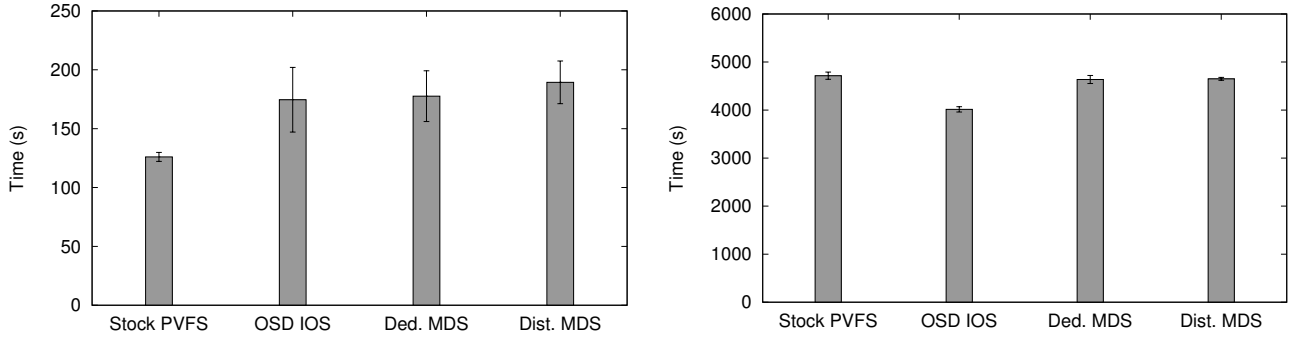

Fig. 8: SSCA-3 execution time; left: SSCA-3 test1; right: SSCA-3 test7GB.

the overhead of OSD-based metadata management while exploiting the high-throughput offered by OSD I/O disks.

One potential drawback of coupling data and metadata on a single device is that data and metadata operations can fall on the same critical I/O path and adversely affect the application's performance. Since SSCA-3 is both data and metadata intensive, it affords us an opportunity to verify if this is indeed the case. As we can see from Figure 8 (right), there is no significant increase in the SSCA-3 execution time for the Distributed OSD MDS file system. If the metadata operations were interfering with the critical I/O path, we would have seen an impact on the application execution time.

VIII. Related Work

File system metadata management has long been an active area of research [17]. With the advent of commodity clusters and parallel file systems [4], [7], [5], managing metadata efficiently and in a scalable manner offers significant challenges.

Distributed file systems often dedicate a subset of the servers for metadata management. Mapping the semantics of data and metadata across different, non-overlapping servers allows file systems to scale in terms of I/O performance and storage ca-

pacity. File systems such as NFS [18], Lustre [5] and GFS [19] use a single metadata server to manage a globally shared file system namespace. While simple, this design does not scale, resulting in the metadata server becoming a bottleneck and a single point of failure.

To mitigate the problems associated with a central metadata server, AFS [17] and NFS [18] employ static directory subtree partitioning [20], [21] to partition the data namespace across multiple metadata servers. Each server is delegated the responsibility of managing the metadata associated with a subtree. Hashing [21] is another technique used to partition the file system namespace. It uses a hash of the file name to assign metadata to the corresponding MDS.

A recent trend among distributed file systems is to use the concept of objects to store data and metadata. CRUSH [22] is a data distribution algorithm that maps object replicas across a heterogeneous storage system. Lustre [5], PanFS [7] and Ceph [6] use various non-standard object interfaces requiring the use of dedicated I/O and metadata servers. Instead, our work is an attempt to break away from the dedicated server paradigm and redesign parallel file systems to use standards-compliant OSDs [11], [23] for data and metadata storage.

IX. Conclusions and Future work

Dedicated metadata servers limit the performance and scalability of parallel file systems. In this paper we make the case for recoupling file data and metadata using OSDs. We have successfully offloaded the metadata management capabilities of a parallel file system to OSDs. We presented two MDS offloading techniques that exploit the semantic opportunities offered by OSDs to manage metadata. We also presented results from microbenchmarks and applications. Despite the overhead associated with using software-based OSDs, the performance of our OSD-based metadata management techniques is comparable to, and at times better than, that of dedicated metadata servers.

A related publication addresses issues of using OSDs for directory operations. While the tests in this paper were carefully designed to study the non-directory metadata aspects, that study isolates just the directory aspects.

Acknowledgment

This material is based upon work supported by the National Science Foundation under Grant No. 0621484.

References

[1] CERN, "The large hadron collider," http://lhc.web.cern.ch/lhc/.
[2] SDSS, "Sloan digital sky survey," http://www.sdss.org/.
[3] S. A. Weil, K. T. Pollack, S. A. Brandt, and E. L. Miller, "Dynamic metadata management for petabyte-scale file systems," in *SC '04: Proceedings of the 2004 ACM/IEEE conference on Supercomputing*. Washington, DC, USA: IEEE Computer Society, 2004, p. 4.
[4] P. H. Carns, W. B. Ligon III, R. B. Ross, and R. Thakur, "PVFS: A parallel file system for Linux clusters," in *Proceedings of the 4th Annual Linux Showcase and Conference*, 2000, pp. 317–327.
[5] Cluster File Systems, Inc., "Lustre: a scalable high-performance file system," Cluster File Systems, Tech. Rep., Nov. 2002, http://www.lustre.org/docs/whitepaper.pdf.
[6] S. A. Weil, S. A. Brandt, E. L. Miller, D. D. E. Long, and C. Maltzahn, "Ceph: A scalable, high-performance distributed file system," in *Proceedings of OSDI'06*, Seattle, WA, Nov. 2006, pp. 307–320.
[7] D. Nagle, D. Serenyi, and A. Matthews, "The Panasas ActiveScale storage cluster—delivering scalable high bandwidth storage," in *Proceedings of the ACM/IEEE SC2004 Conference (SC'04)*, Pittsburgh, PA, Nov. 2004.
[8] D. Roselli, J. Lorch, and T. Anderson, "A comparison of file system workloads," in *Proceedings of the 2000 USENIX Annual Technical Conference*, Jun. 2000, pp. 41–54.
[9] F. Wang, S. A. Brandt, E. L. Miller, and D. D. E. Long, "OBFS: A file system for object-based storage devices," in *21st IEEE / 12th NASA Goddard Conference on Mass Storage Systems and Technologies (MSST'04)*, College Park, MD, Apr. 2004, pp. 283–300.
[10] F. Wang, Q. Xin, B. Hong, S. A. Brandt, E. L. Miller, D. D. E. Long, and T. T. McLarty, "File system workload analysis for large scale scientific computing applications," in *Proceedings of the Twentieth IEEE/Eleventh NASA Goddard Conference on Mass Storage Systems and Technologies*, College Park, MD, Apr. 2004.
[11] R. O. Weber, "Information technology—SCSI object-based storage device commands -2 (OSD-2), revision 1," INCITS Technical Committee T10/1729-D, Tech. Rep., Jan. 2007.
[12] A. Devulapalli, D. Dalessandro, P. Wyckoff, N. Ali, and P. Sadayappan, "Integrating parallel file systems with object-based storage devices," in *Proceedings of SC'07*, Reno, NV, Nov. 2007.
[13] D. R. Hipp *et al.*, "SQLite," http://www.sqlite.org/, 2007.
[14] A. Devulapalli, D. Dalessandro, P. Wyckoff, and N. Ali, "Attribute storage design for object-based storage devices," in 24^{th} *IEEE Conference on Mass Storage Systems and Technologies (MSST 2007)*, San Diego, CA, Sep. 2007.
[15] N. Ali, A. Devulapalli, D. Dalessandro, P. Wyckoff, and P. Sadayappan, "An OSD-based approach to managing directory operations in parallel file systems," in *IEEE International Conference on Cluster Computing*, Tsukuba, Japan, Sep. 2008.
[16] HPCS, "Scalable Synthetic Compact Application," http://www.highproductivity.org/SSCABmks.htm.
[17] J. H. Morris, M. Satyanarayanan, M. H. Conner, J. H. Howard, D. S. Rosenthal, and F. D. Smith, "Andrew: a distributed personal computing environment," *Communications of the ACM*, vol. 29, no. 3, pp. 184–201, 1986.
[18] B. Pawlowski, C. Juszczak, P. Staubach, C. Smith, D. Lebel, and D. Hitz, "NFS version 3: Design and implementation," in *USENIX Summer Technical Conference*, 1994, pp. 137–152.
[19] S. Ghemawat, H. Gobioff, and S.-T. Leung, "The Google file system," in *SOSP '03: Proceedings of the nineteenth ACM symposium on Operating systems principles*. New York, NY, USA: ACM Press, 2003, pp. 29–43.
[20] E. Levy and A. Silberschatz, "Distributed file systems: concepts and examples," *ACM Computing Surveys*, vol. 22, no. 4, pp. 321–374, 1990.
[21] S. Brandt, E. Miller, D. Long, and L. Xue, "Efficient metadata management in large distributed file systems," in *20th IEEE/Eleventh NASA Goddard Conference on Mass Storage Systems and Technologies*, Apr. 2003.
[22] S. A. Weil, S. A. Brandt, E. L. Miller, and C. Maltzahn, "Crush: controlled, scalable, decentralized placement of replicated data," in *SC '06: Proceedings of the 2006 ACM/IEEE conference on Supercomputing*. New York, NY, USA: ACM Press, 2006, p. 122.
[23] Seagate Research, "The advantages of object-based storage," http://www.seagate.com/docs/pdf/whitepaper/tp_536.pdf, Seagate, Inc., Tech. Rep. TP-536, 2005.