Giga+TableFS on PanFS: Scaling Metadata Performance on Cluster File Systems

Kartik Kulkarni, Kai Ren, Swapnil Patil, Garth Gibson

CMU-PDL-13-101
January 2013

Parallel Data Laboratory
Carnegie Mellon University
Pittsburgh, PA 15213-3890

Abstract

Modern File Systems provide scalable performance for large file data management. However, in case of metadata management the usual approach is to have single or few points of metadata service (MDS). In the current world, file systems are challenged by unique needs such as managing exponentially growing files, using filesystem as a key-value store, checkpointing that are highly metadata intensive and are usually bottlenecked by the centralized MDS schemes.

To overcome this metadata bottle-neck, we evaluate a scalable MDS layer for the existing cluster file systems using Giga+ -a high performance distributed index without synchronization and serialization and TableFS -a file system with an embedded No-SQL database using modern key-value pair levelDB. We take layered approach to scale the metadata performance which does not need any hardware infrastructure upgrade in the existing storage clusters. In addition to providing scalable and increased metadata performance by several folds, avoiding metadata hotspots, packing small files, our MDS layer adds no-or-low performance overhead on the data throughput and resource utilizations of the underlying cluster.

Acknowledgements: This research is supported in part by The Gordon and Betty Moore Foundation, NSF under award, SCI-0430781 and CCF-1019104, Qatar National Research Foundation 09-1116-1-172, DOE/Los Alamos National Laboratory, under contract number DE-AC52-06NA25396/161465-1, by Intel as part of the Intel Science and Technology Center for Cloud Computing (ISTC-CC), by gifts from Yahoo!, APC, EMC, Facebook, Fusion-IO, Google, Hewlett-Packard, Hitachi, Huawei, IBM, Intel, Microsoft, NEC, NetApp, Oracle, Panasas, Riverbed, Samsung, Seagate, STEC, Symantec, and VMware. We thank the member companies of the PDL Consortium for their interest, insights, feedback, and support.

Keywords: metadata scaling, metadata bottleneck, layered MDS, checkpointing, PanFS, Giga+, TableFS, No-SQL

1 Introduction

Modern distributed file systems serving "Big Data", and running on some of the largest clusters, including HDFS [7], GoogleFS [6], PanFS [8], and PVFS [5] are experiencing metadata-bottleneck due to centralized metadata server schemes. Most of these file systems bifurcate metadata management from the scalable storage of file data. This design can scale the storage capacity and data access bandwidth. However, since they have a centralized metadata management, the metadata access rate still remains a bottleneck.

This inherent metadata scalability handicap limits metadata intensive workloads such as checkpoint-restart, key-value storage, gene sequencing, image processing[9], phone logs for accounting and billing, and photo storage [12]. The checkpoint-restart workload, where many parallel applications running on, for instance, ORNL's CrayXT5 cluster (with 18,688 nodes of twelve processors each) periodically write application state into a file per process [13, 14]. Applications that do this per-process checkpointing are sensitive to long file creation delays because of the generally slow file creation rate. Other than these specific workloads, in general, studies have shown that about 75% of the file system operations performed in a data center require access to file metadata. [10][11].

To adapt to the needs of metadata intensive applications, the parallel file system makers are now integrating scalable metadata techniques within the file system. Lustre[20] community's Clustered Metadata Server (CMD) project, scaling of GoogleFS to support 50 million+ files in the next version, ColossusFS in order to use BigTable [16] to provide a distributed file system metadata service are some of the major projects in this context.

The Lustre community considers the performance bottleneck that constraints the file system throughput arising from having a single MDS. With a single MDS, Lustre metadata operations can be processed only as quickly as what a single server and its backing filesystem can manage. Lustre community has proposed CMD architecture which allows for multiple metadata servers sharing the metadata workload. Though CMD has complex mechanisms that can impact the throughout like making a distributed metadata transaction atomic via global locking, this is a noteworthy approach towards scaling metadata performance.

However, these adaptations require an increase in the infrastructure to support new and more MDS. "Scalable" in the context of filesystem-built-in MDS usually refers to improving metadata performance near-linearly with the number of dedicated MDS servers added in addition to the existing ones in the cluster. The scalable MDS solutions also often come with an additional performance overhead of maintaining metadata consistency across the distributed MDS, synchronizing the updates, serialization of critical sections etc.

It would be a desirable solution to have a scalable metadata service without infrastructure upgrade, synchronization and serialization overhead. Perhaps every node in a cluster could share the metadata workload in addition to the regular data workload yet not affecting the file system data throughput. In other words, layering a scalable and a distributed metadata service over an existing file system cluster, such that every node contributes to the metadata workload, yet maintaining the data throughput and consistency model would be an ideal choice.

We evaluate such a layered, scalable and distributed metadata management scheme using Giga+[1] -a high performance distributed index without synchronization and serialization. TabeFS[2] -a file system with an embedded No-SQL database using modern key-value pair levelDB[3]. Together, Giga+ and TableFS offer a fast, scalable and distributed metadata service layer for the existing cluster file systems, while still letting the underlying file system manage the file data operations at its highest throughput, and also manage features such as replication, striping etc.

2 Overview

In this effort, we scale the metadata performance of a PanFS storage cluster by layering Giga+TableFS above it. Layering of the Giga+Tables means that each node in a compute cluster (not the PanFS storage cluster), that mounts PanFS in the backend, runs one instance of the Giga+ server and mounts atleast one Giga+ client. Giga+ client mount point is the gateway to access the underlying PanFS storage cluster. All applications in the nodes accessing the under lying PanFS storage do so via their individual Giga+ client mount points. Essentially, we layer Giga+ across our compute cluster without any modification to the backend storage cluster. The Fig 2.1 illustrates Giga+TableFS layering over a 3 node compute cluster. As we can see there is no modification needed to the underlying storage cluster to support this layering.

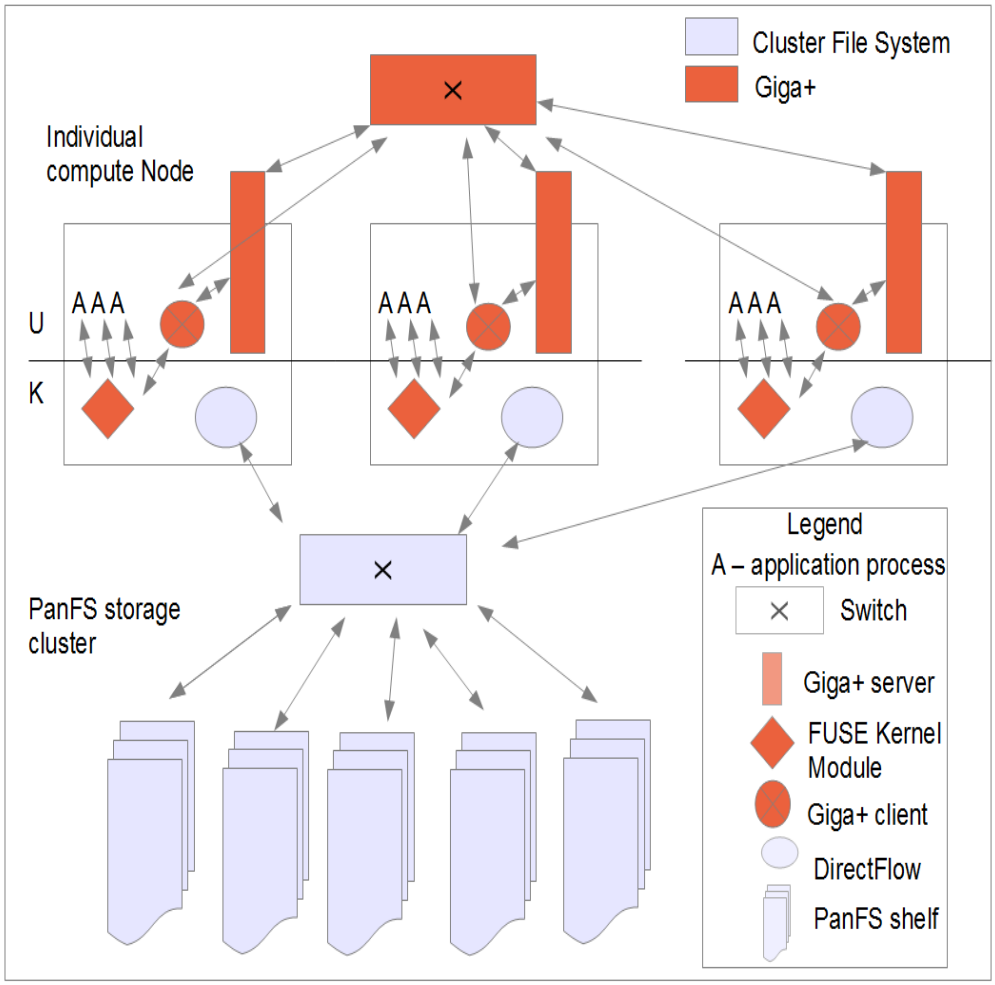


Fig 2.1: Layering Giga+TableFS metadata service over a cluster backed by PanFS storage

2.1 Centralized metadata management of PanFS

In a PanFS storage cluster [17], Director Blades are the metadata managers that centralize metadata service to a set of Storage Blades – object storage devices. A typical configuration involves one Director Blade per tens of Storage Blades. PanFS stores file metadata in object attributes on two of the N objects used to store the file's data. Thus, every file operation involves accessing a Director Blade and may additionally involve two more accesses to the Storage Blades. Since a single Director Blade manages metadata on its set of Storage Blades, the metadata management is centralized.

A PanFS storage cluster can have multiple Director Blades to achieve greater metadata performance. However, a particular volume is assigned to a particular director blade. This means that all the metadata accesses to any file in a volume will have to be served by a single Director Blade, which serves that volume.

2.2 Layering Giga+ TableFS for scalable metadata performance

While layering Giga+ TableFS metadata service layer over cluster file systems (in this particular effort, PanFS), we consider evaluation of the following concerns:

1) What is the metadata performance gain?
2) What is the impact of layering on the underlying file system's data throughput?

An ideal layering would result in scalable metadata performance, which we obtain from the stand-alone Giga+ TableFS directory service; as well as the highest data throughput that is possible from the storage cluster without layering. The layering should also solve problems like metadata hotspots by load balancing across all the nodes in the storage cluster. Finally, additional overhead of running Giga+ servers and clients to accomplish the layering should be minimal.

We evaluated Giga+ TableFS FUSE [15] prototype over an 8 node compute cluster mounting a PanFS storage gear in its backend. Our evaluation includes both qualitative and quantitative reasoning of the above concerns. During the course of our evaluation, we propose and implement 4 techniques to better integrate the metadata service layer with the underlying file system for performance.

3 What is the metadata performance gain?

Giga+ TableFS in a stand-alone deployment [4] - without layering above PanFS, provides scalable metadata performance in terms of number of creates per second and stat lookups. The graph form the Giga+TableFS performance evaluation exercise shows that the absolute creates/sec rate in a single directory on a typical 8 node cluster should be around 25,000. Ideally, this rate should be sustained after layering.

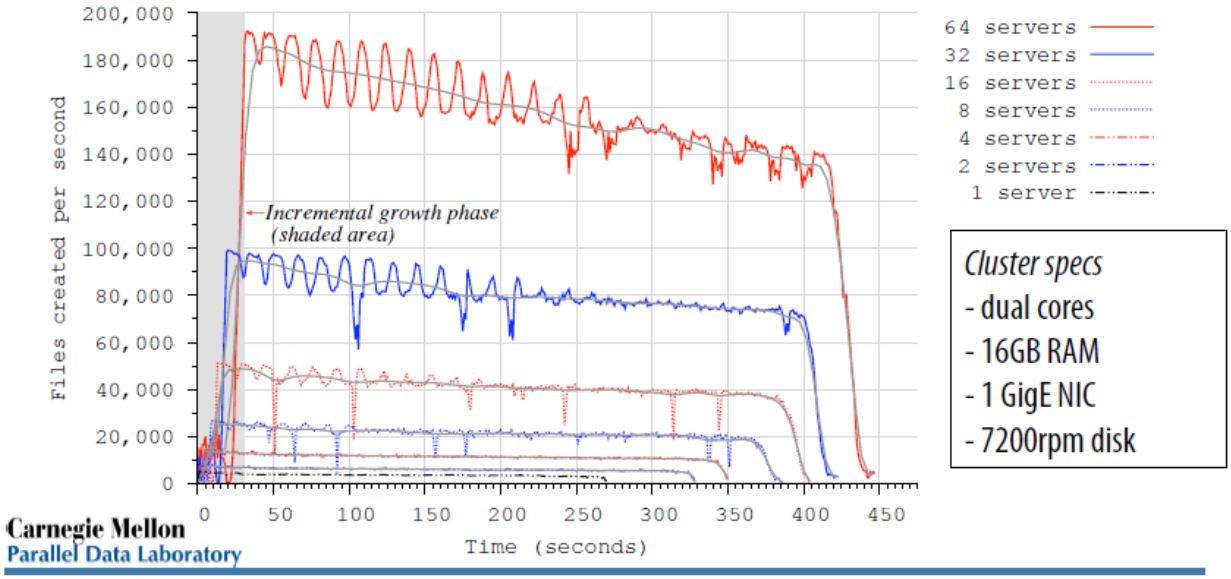


Fig 3.1: Scalable performance of Giga+TableFS: creates/sec rates [4]

3.1 Limitations

Sustaining the high metadata performance of Giga+TableFS over a PanFS cluster has some limitations arising due to properties of PanFS.

1) The file create latency in PanFS is much more than that of Giga+. Creating an object on PanFS takes an update to one MDS and at least two OSDs; while in the best case, a "create" in Giga+ may be on just one node. This fact impacts creates/sec rate, if for every "create" by an application; we create an object in the underlying PanFS storage cluster.

2) Centralization of metadata accesses to every directory. PanFS centralizes updates to a directory, which means that there cannot be concurrency to boost creates/sec rate within a single directory. Whereas in Giga+, since all metadata is stored as key value pairs in SSTables there can be any number of concurrent(but not conflicting) updates to a directory.

3) Coarse grained allocation of volumes to MDS. In PanFS, a volume is mapped to a particular and single Director Blade that manages all the metadata of those volumes. Hence parallel updates to a set of volumes mapping to the same MDS are not scalable.

3.2 Overcoming concurrency limitations

We propose and evaluate 4 techniques for better layering the Giga+TableFS metadata service over PanFS. The techniques work around the centralization limitations of PanFS and can implemented within the Giga+ TableFS layer. We further demonstrate that these are critical in achieving scalable performance.

3.2.1 Distributing same-directory metadata operations across volumes

In order to scale the performance of parallel metadata operations, it is important to provide scope for parallelism by avoiding centralization. It would be an ideal case if every metadata operation by the applications gets routed to a

different MDS. Since Giga+TableFS adds a layer of indirection between the applications' operations and the actual operations at the PanFS cluster, there is a potential scope to provide for such parallelism.

Giga+ incrementally splits a directory into partitions and offloads one of them to a new server for load balancing and parallelism. The idea is to make scope for more than just one server to process concurrent updates to the same directory. A configurable split threshold decides when to split a directory and the splitting process takes place without any serialization or synchronization between the Giga+ servers. The same idea of directory-partitions can be extended and applied when creating objects on PanFS as well. Splitting a large directory into multiple partitions and having different volumes in PanFS that are managed by different MDS serve those partitions, brings up the scope for same-directory metadata operations.

3.2.2 Distributed file creation scheme for PanFS

We propose to represent the Giga+ directories on PanFS as a group of directories that correspond to the Giga+ partitions. All files created in a particular Giga+ directory partition go to their respective-corresponding subdirectories on PanFS. Further, during every split in Giga+ which causes creation of a new partition, a new corresponding sub directory is created in a volume of PanFS that is mapped to a different MDS.

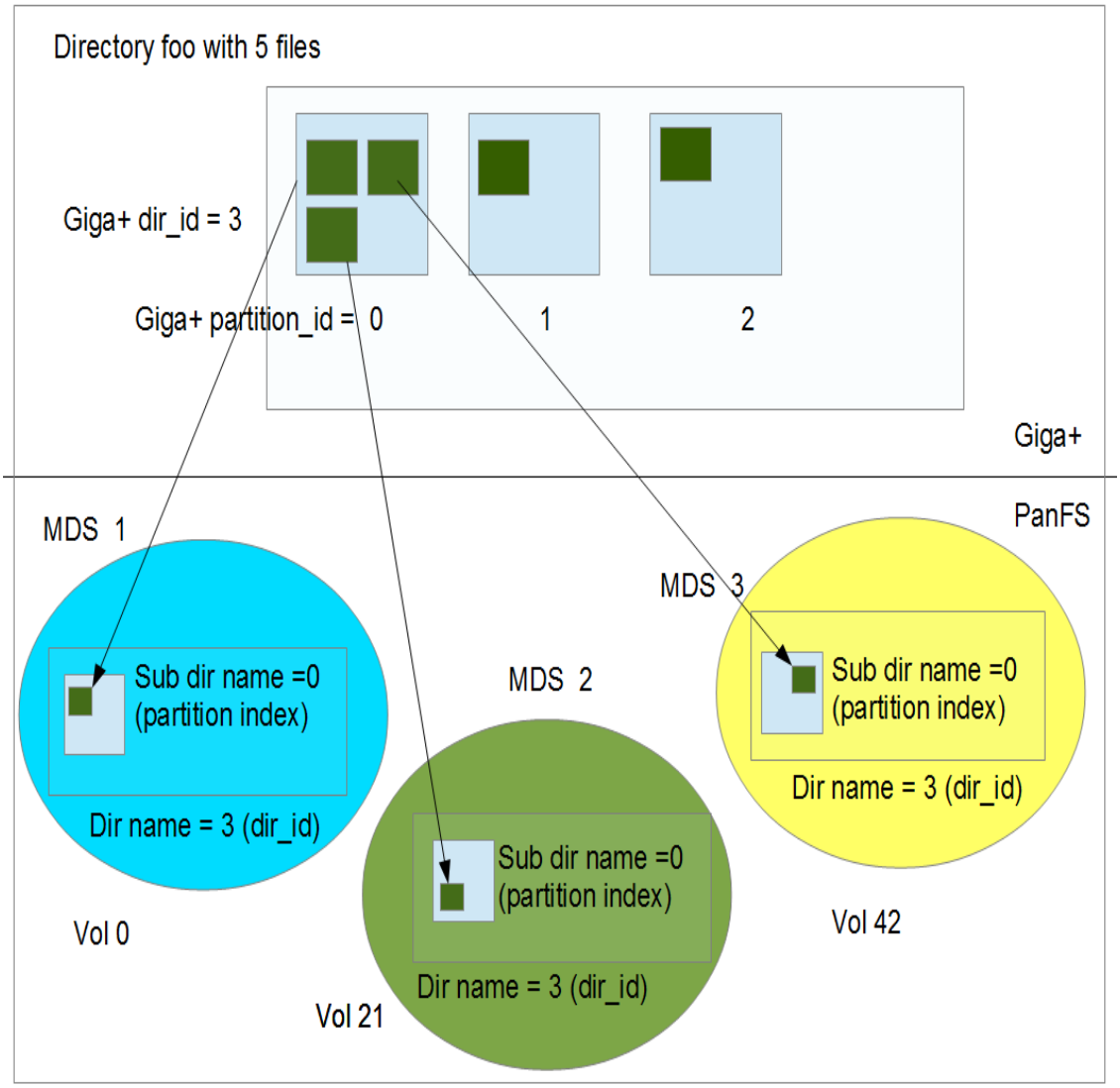


Fig 3.2.2.1: Representing a Giga+ directory on PanFS

As illustrated in the Fig 3.2.2.1, a directory "foo" with 3 partitions by Giga+, is represented in PanFS by creating 3 subdirectories in 3 different volumes (managed by different MDS). Files in the partitions are created in their respective-corresponding PanFS volumes. Thus in this example, creates on different partitions are can be created in parallel in PanFS. This is the same level of concurrency provided by Giga+.

3.2.3 Dispersing Metadata-Hotspots

A large number of accesses to the same set of files within a single directory or a large number of mutations to a single directory can cause metadata hotspots. By extending the inherent mechanism of Giga+ to split a directory to a new server for load balancing, we create the files in different volumes in a distributed manner, hence alleviating the metadata hot spot problem.

Creating a large number of files in a single directory can make it a hot-spot. Looking up metadata from such a large directory can cause performance degradation. By layering Giga+TeableFS, such hotspots can be avoided at the create step by creating a file in PanFS at path = "/dir_id/partition_id/file_name." In Giga+, a directory is partitioned if the number of dentries in it exceeds a configurable "SPLIT_THRESHOLD" (default=8000). This feature is extended and used for evenly distributing the files across the underlying cluster file system. Including "partition_id" in the path name ensures that no directory of the underlying cluster file system has more than "SPLIT_THRESHOLD" files.

3.2.4 Lazy File Creates and Small file packing

We propose to delay the actual file creates on PanFS and instead just create the file entry on Giga+ TableFS. Later when the file is actually written or written beyond a particular threshold size, they file can be actually created on PanFS. This scheme hides the file creation latency into one of the write operations. This scheme has 2 different ramifications that can affect big performance gains for small files.

1) Files below a particular threshold may not be created on PanFS at all. They could be stored as key-value pair within the Giga+TableFS. Since a create operation on PanFS is saved and the small file contents are stored efficiently in the leveldb SSTables (providing sequential disk read/write performance), small file throughput will be maximized.

2) Beyond a particular threshold, the file can actually be created on PanFS and its contents in TableFS can be transferred to the newly created file on PanFS. This background processing between the initial create operation issued by the application and the actual creation on PanFS, can be utilized to share the simultaneous creates among server processes in the cluster.

If 2 processes simultaneously create a file, the "creates" are absorbed by Giga+TableFS. Later, if they have different thresholds, their actual creates on PanFS happens at different times and thus avoiding the original simultaneous create competition. Having different/random thresholds for different processes can hence lead to a better timing-harmony of creates.

3.3 Experimental evaluation of Metadata performance

3.3.1 Experimental setup

Our evaluation was performed on an OpenCirrus [18] compute cluster with a Tashi [19] Virtual Machine Cluster management system infrastructure. Each of our workload generating nodes was a Tashi Virtual Machine running on a dedicated physical machine. Each node mounted a PanFS client (Direct Flow) on a single mount point. The PanFS storage cluster itself had 5 shelves.

Each of our nodes ran 1 instance of Giga+ Server. The Giga+ servers used TableFS internally to store the entire cluster's metadata. The workload generating applications on our nodes used variable number of Giga+ client mount points, depending on the test case.

The below tables provide complete configuration information of various entities – nodes, physical machines, Giga+ and PanFS storage cluster.

Table 3.3.1.1: Compute Nodes

Node type	Tashi Virtual Machine
Total no of nodes (Tashi virtual machine)	8
Total no of physical machines	8
Node(s)/Physical machine	1 (dedicated)

Table 3.3.1.2: System resources per Virtual Node

Resource	Configuration
# CPUs	8
CPU Type and Speed	Intel 2.6 GHz
Memory	14 GBytes
NIC	1 Gig E
Network MTU	1500

Table 3.3.1.3: Giga+ configuration on each Node

# of Giga+ Servers per node	1
Giga+ Server split threshold	8000
# of Giga+ Clients per node	Variable (1 through 4)
FUSE threading configuration	Single Threaded (per Giga+ client)

Table 3.3.1.4: PanFS storage cluster configuration

Resource	Configuration
# of PanFS client (Direct Flow) mount points per node	1
# of PanFS volumes	100
# of volumes per MDS	20
# of Shelves	5
# of MDS per shelf	1
# of OSD per shelf	10
Director Blades [MDS]	5 – A200e-12GB: Quad Core Xeon 1.73GHz, (1) 160GB drive, LAGG Network Interface 1
Storage Blades [OSDs]	50 – A4000-8GB: Xeon 1.73GHz, (2) 2000GB drives, LAGG Network Interface 1

3.3.2 Some commonly used terms explained

node	Tashi virtual machine. Each virtual machine runs on a dedicated physical machine. Every node mounts a PanFS client (Direct Flow) to access the PanFS storage cluster in the backend.
application	Workload generating process. For ex: program creating 100 files and writing 4 GB to each. In case of multiple applications, they run in parallel.
client	Giga+ client. Every client is represented by an individual FUSE mount point. Each node can have multiple clients depending on the test case. Applications read/write to the mount points. We use multiple FUSE mount points on a single node to ensure that applications run in parallel irrespective of the multithreading configuration of FUSE.
server	Giga+ server. Each node runs 1 Giga+ server

3.3.3 Experimental approach and results

The workload, against which the metadata performance is tested, includes:

a) Creating thousands to millions of files (make_node test) in a single directory by parallel applications across multiple nodes.
b) Stating the created files (stat_test).

The parameters evaluated are:
1) Creates/Sec
2) Total time for stat_test on the created files.

The tests were performed for various target schemes:

1) **Directly on PanFS, applications across multiple nodes**– In this scheme of testing, applications across the nodes create files on PanFS via their node's Direct Flow mount point.
2) **No Create-distribution** – Giga+TableFS is layered above PanFS storage cluster. The applications create files via their individual clients. However, there is no file create-distribution as described in 3.2.2.
3) **With create-distribution, applications on same node** – Giga+TableFS are layered above PanFS storage cluster. The applications create files via their individual clients. File create-distribution, as described in 3.2.2, is employed. However, all the clients and the applications are on a single node.
4) **With create-distribution, applications across multiple nodes** – Giga+TableFS are layered above PanFS storage cluster. The applications create files via their individual clients. File create-distribution, as described in 3.2.2, is employed. Giga+ clients and the applications are across multiple nodes.
5) **Stand-alone Giga+TableFS/Lazy creates**
    a. **Stand-alone Giga+TableFS** - On file creation by an application, they are created only in the Giga+TableFS layer and not in the underlying PanFS storage cluster.
    b. **Lazy creates** - On file creation by an application, they are initially only created in the Giga+TableFS layer. Later on writing to the files, they are actually created in the underlying PanFS cluster. This technique is described in 3.2.4

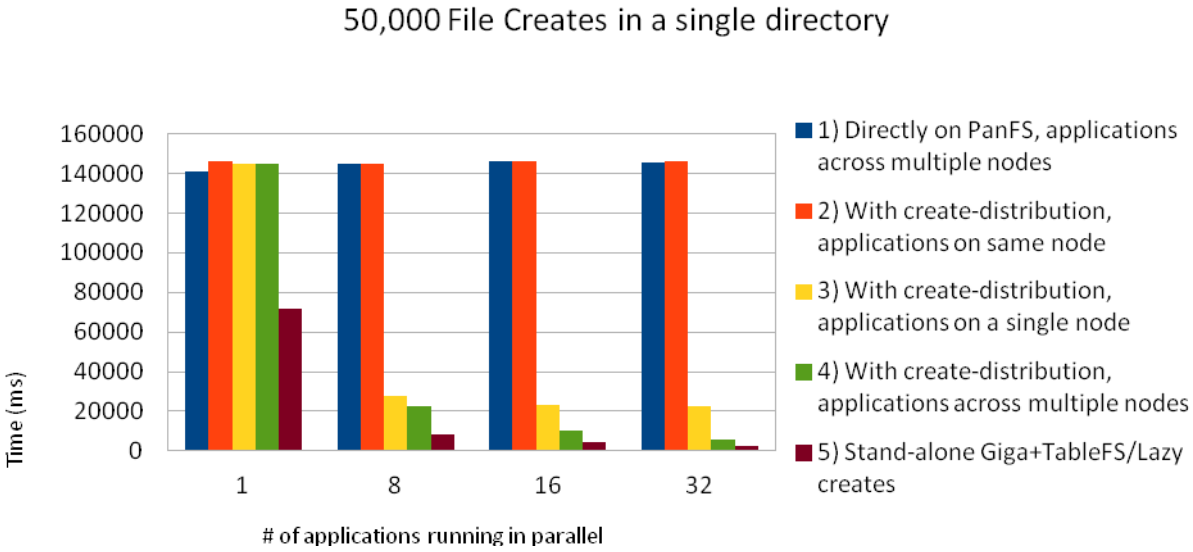


Fig 3.3.2.1: mknod_test results for various target schemes

We can see that the create-distribution (Schemes 3 and 4) to overcome PanFS's centralization of metadata accesses is critical to provide the scalable metadata performance of Giga+TableFS. Also since the metadata path in Giga+TableFS layered over PanFS is quite long (involves more than 1 network links in the worst case), it helps to have parallel applications across different nodes (Scheme 4) to create enough parallelism to create a "pipelining effect" which increases the overall metadata performance. Finally, Lazy-Creates provide the utmost creates/sec rate,

which is close to that of the stand-alone Giga+TableFS. High metadata performance of the stand-alone Giga+TableFS is provided by the efficient on-disk key-value storage of leveldb.

Fig 3.3.2.2 illustrates the results of stat_test – running stat on a directory containing thousands of files from any one client on a node. We can see an improvement in the stat performance of PanFS after layering Giga+TableFS due to efficient key value storage by lelveldb.

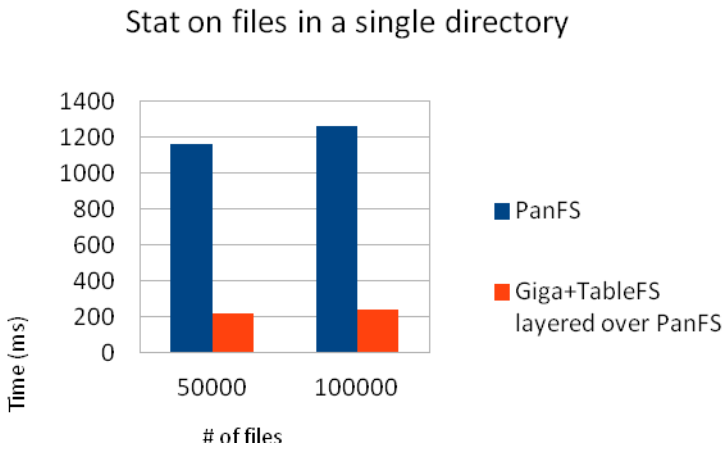
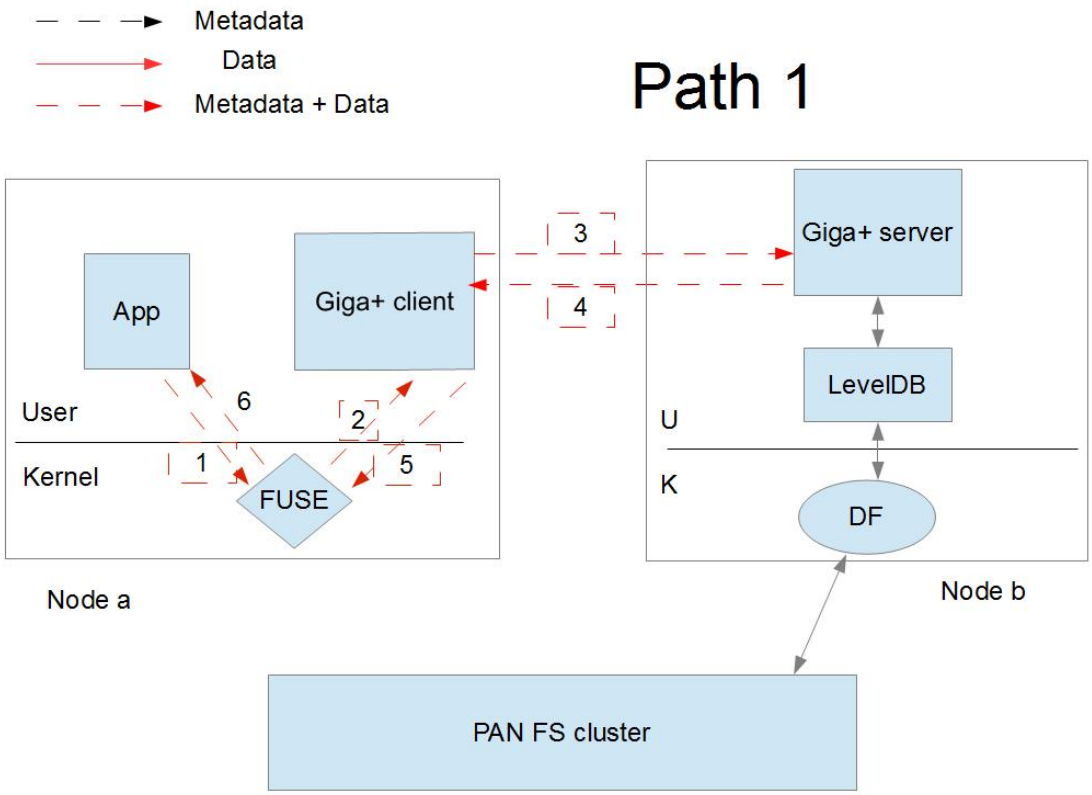


Fig 3.3.2.2: Stat test results for PanFS and Giga+TableFS layered over PanFS

4 What is the impact of layering on the underlying file system's data throughput?

While layering Giga+TableFS helps overcoming the metadata-bottleneck, it is also very important to ensure that the impact of the large file data performance is trivial. It would be an ideal case to have complete isolation of the data and metadata paths so that improving performance on one does not affect the other at all. However, since data modifications to files almost every time results in a metadata modification, complete isolations of the data and metadata paths becomes a non trivial system level problem.

After the Giga+PanFS Layer processes a metadata operation, subsequent data operations need to be routed to reach the actual files. Similarly after completion of the data operations, metadata updates become impending. We have several stages in Giga+TableFS at which the metadata path could be split apart from data path. With respect to metadata updates after a data operation, having the Giga+ Servers proxy data to the applications (Path 1) along with the metadata could be a good choice.



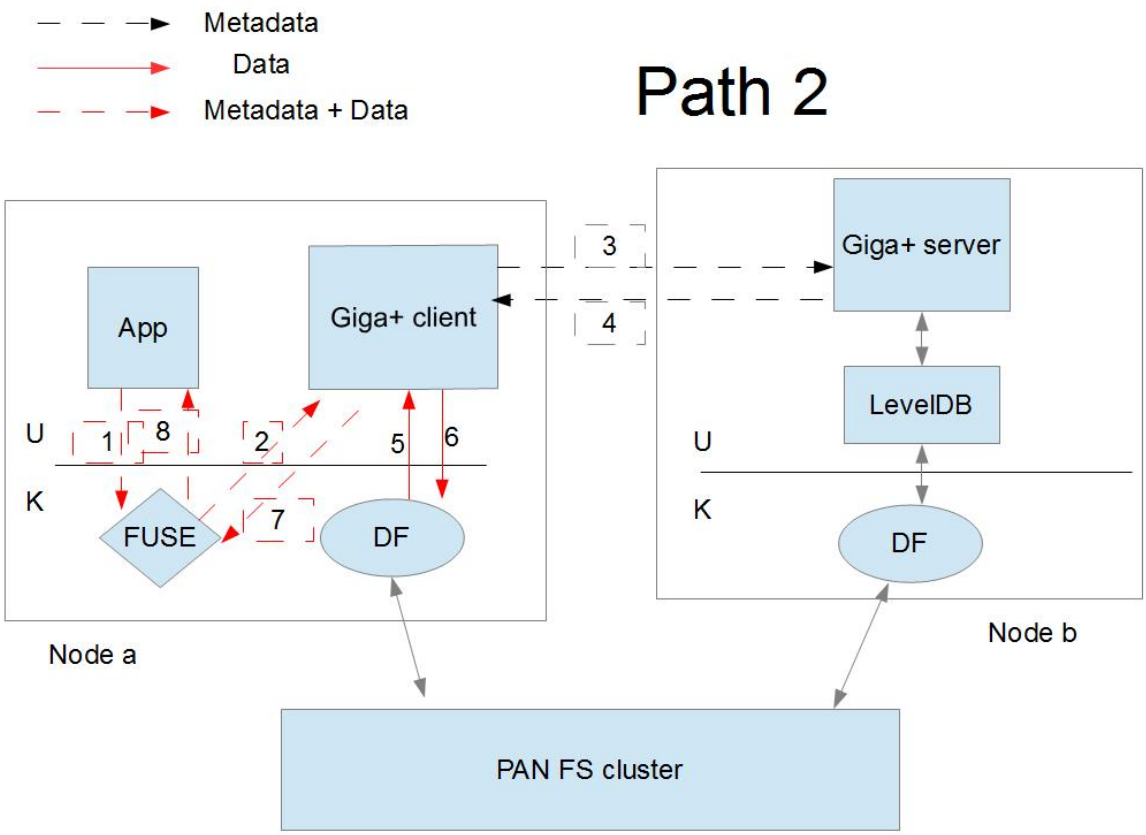
Step 1: Applications query for metadata and data via syscalls
Step 2: FUSE kernel module bounces off the syscalls to userspace Giga+ client
Step 3: Giga+ client queries the Giga+ Server for both metadata and data
Step 4: Giga+ Server relays metadata from lebelDB, and data from PanFS (via Direct Flow mount point)
Step 5: Giga+ client returns the metadata and data to FUSE kernel module
Step 6: FUSE kernel module returns the metadata and data to the applications

Fig 4.1: Path 1, indirect data-access to the files via Giga+ servers.

In this model – Path 1, Giga+ server is responsible for serving both data and metadata. The sequence of steps from 1 through 6 is taken for all metadata and data operations.

Path 1 adds an extra link segment (Application to Giga+ server) to the datapath that otherwise would have been directly reaching the storage cluster otherwise. Since the applications query Giga+ clients (FUSE mounted) for data, there are 2 additional copies from the user space to kernel space and vice versa. These overheads associated with Path 1 are expected to impact the data throughput and hence it may be desirable to have a more direct data path.

In an attempt to avoid the additional network link between the application and its files, we could let the Giga+ client at the application's node proxy data from the underlying storage cluster (Path 2). The advantage of this approach is that the Giga+TableFS layer is still in the loop to complete updating the file metadata upon completion of the data operations, while still avoiding the extra network link latency. However, data copy overhead still exists in this model.

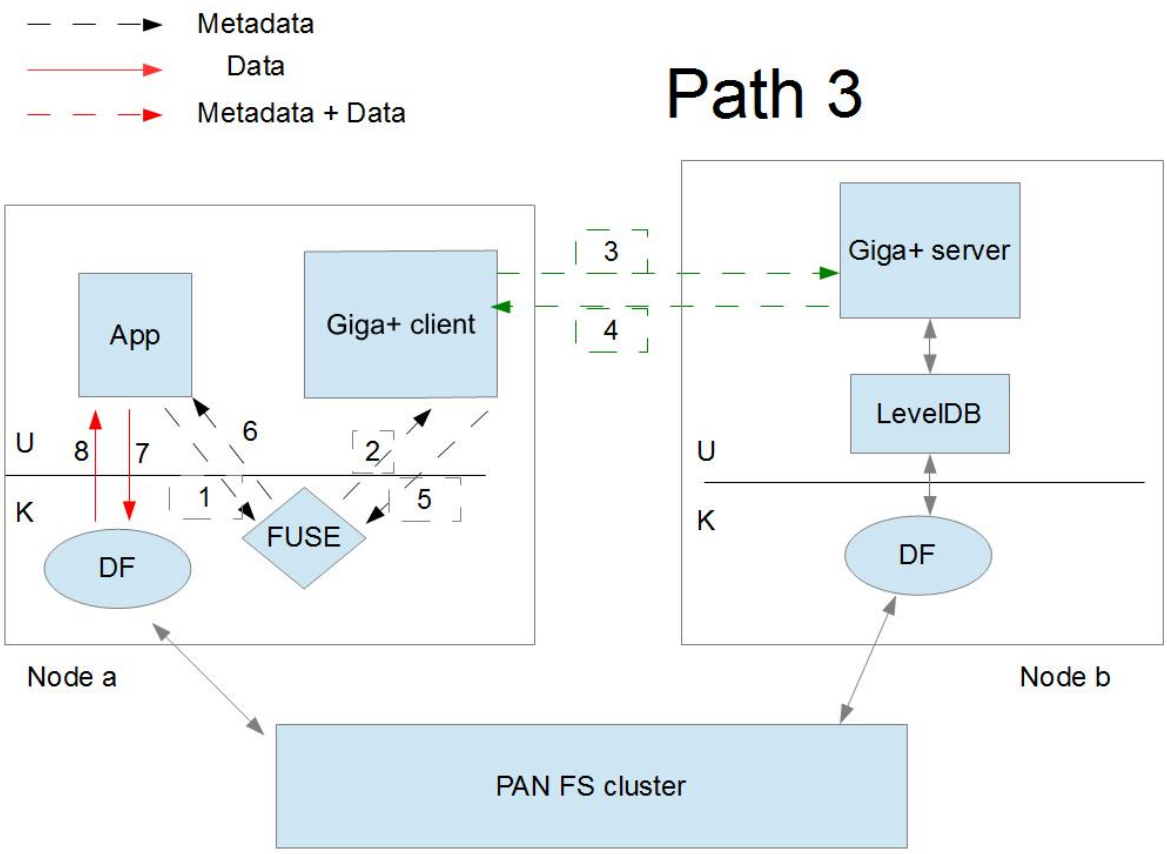


Step 1: Applications query for metadata and data via syscalls
Step 2: FUSE kernel module bounces off the syscalls to userspace Giga+ client
Step 3: Giga+ client queries the Giga+ Server for only metadata
Step 4: Giga+ Server relays metadata from lebelDB
Step 5: Giga+ client directly accesses data from PanFS Direct Flow mount point
Step 6: Direct Flow returns data to the Giga+ client
Step 7: Giga+ client passes metadata and data to the FUSE kernel module
Step 8: FUSE kernel module returns the metadata and data to the application

Fig 4.2: Elimination of the Giga+ server from the data path

In an ideal case, once the metadata lookups are done, it would be desirable to provide the most direct access to the file bypassing the metadata service layer. For instance if the Giga+ client just gave back the direct link of the file on the PanFS cluster, the application could directly reach the data and there would not be a data performance overhead. This is shown in path 3, Fig 4.3

However, on completion of the data operations, updating the metadata changes caused by the data operation (such as change in size etc) in leveldb requires some sort of feedback path to the metadata service layer, telling TableFs that file has stopped changing ,which is missing for this model.



Step 1: Applications query FUSE kernel module only for metadata
Step 2: FUSE kernel module bounces off the syscalls to userspace Giga+ client
Step 3: Giga+ client queries the Giga+ server for only metadata
Step 4: Giga+ Server relays metadata from lebelDB
Step 5; Giga+ client passes metadata and data to the FUSE kernel module
Step 6: FUSE kernel module returns the metadata to the application
Step 7: Application directly accesses data from PanFS Direct Flow mount point
Step 8: Direct Flow returns data to the application

Fig 4.3: Applications directly accessing the files in the storage cluster

4.1 Experimental evaluation of data performance

The data performance test includes creating parallel processes across all the 8 nodes that continuously and simultaneously create N files and write M bytes per file. Each process writes to its individual Giga+(FUSE) mount point (Thus ensuring it executes in parallel irrespective of the multithreaded configuration of FUSE). The average of the cumulative data rate recorded by each process at each instant (file) is plotted. The average of cumulative data rate across all processes and all files is also reported.

We compare the average cumulative data rates, for different number of clients (parallel workload generating processes) for 3 possible schemes path 1, path 2 and path3 – as explained above.

Testing primitives:

- **write_blocks:** This utility writes the specified number of chunks of 4096 Bytes to an open file. It then calculates the total time t taken to write x bytes and prints the data rate (x/t) in MB/s. The time to create the file, open and close is not included in the data rate calculation.

- **create_write:** creates a given number of files in a given directory and for each file, uses the write_blocks utility to write a given number of bytes. For each file, the data rate is logged.The time to create the file, open and close is not included in the data rate calculation.
- **data_test:** runs create_write (with given files and blocks per file) on a given list of mount points. A new instance/process of create_write is created for every mount point.
- **data_test_controller:** runs remotely on OpenCirrus to control the data_tests on the host of VMs (8). It creates a given number of mount points on each VM and calls data_test on each of them.
- **gather_results:** runs remotely on OpenCirrus. It copies to one folder, the data rate logs by each create_write process running on every VM. For every file, sequentially in the logs, it adds the data rate by all the processes and creates an aggregate log.

4.1.1 Experimental setup

Same as described in 3.3.2.

Since our setup has 8 physical machines, each having 1 GE links, the best bandwidth would be 8 Gbps = 1000 MBps.

4.1.2 Experimental approach and results

4.1.2.1 Throughput without Giga+ TableFS layering / Path 3 model

To evaluate the impact of layering Giga+ TableFS metadata service layer, it is important to set the benchmark for data performance without layering – direct data accesses to the PanFS storage cluster. The below graphs show the data rate for above described and compares it with the number of clients.

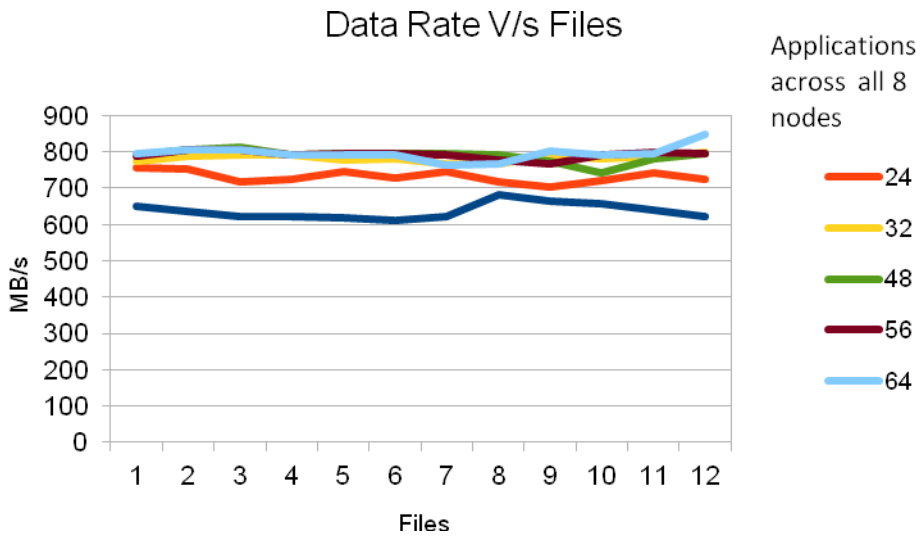


Fig 4.1.2.1: Data rate per file for large write workload (400MB per file) comparison with # of parallel applications

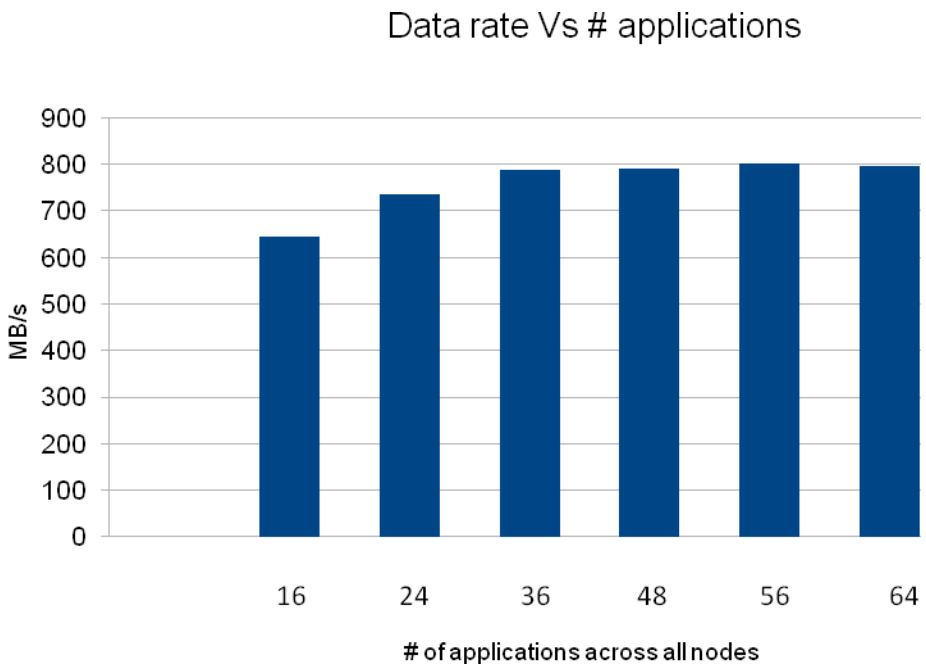


Fig 4.1.2.2: Average data rate comparison with the # of applications generating large write workload

We can see that the data throughput maximizes at 56 applications to about 810 MB/s, when there is sufficient parallelism to keep all the resources utilized. We see similar performance numbers for path 3 scheme as well since path 3 provides same direct path to the files.

4.1.2.2 CPU utilization without Giga+ TableFS layering

CPU utilization is the average of various idle % values reported periodically by vmstat. As seen in Fig 4.1.2.3, in most of the cases the CPU utilization is about 10 %– 15%.

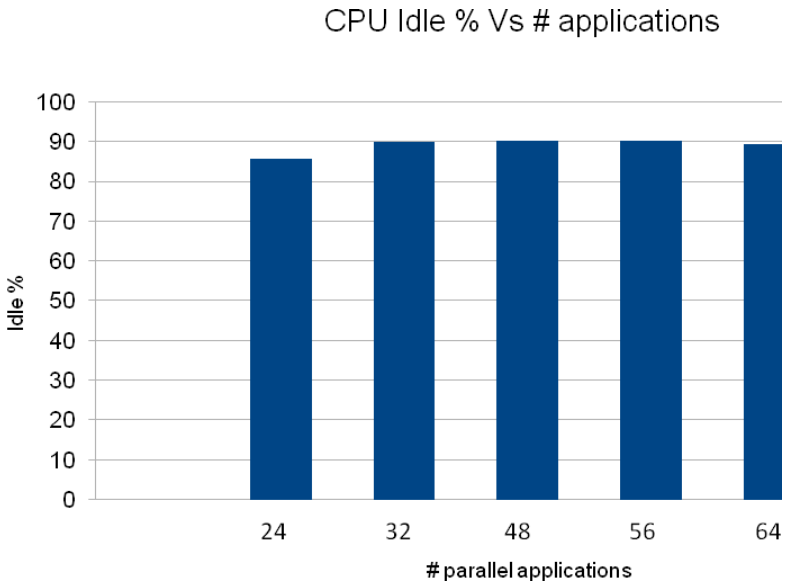


Fig 4.1.2.3: Typical CPU utilization from one of the representative nodes.

4.1.2.3 Memory utilization without Giga+ TableFS layering

Memory utilization is the average of various free memory % values reported periodically by vmstat.

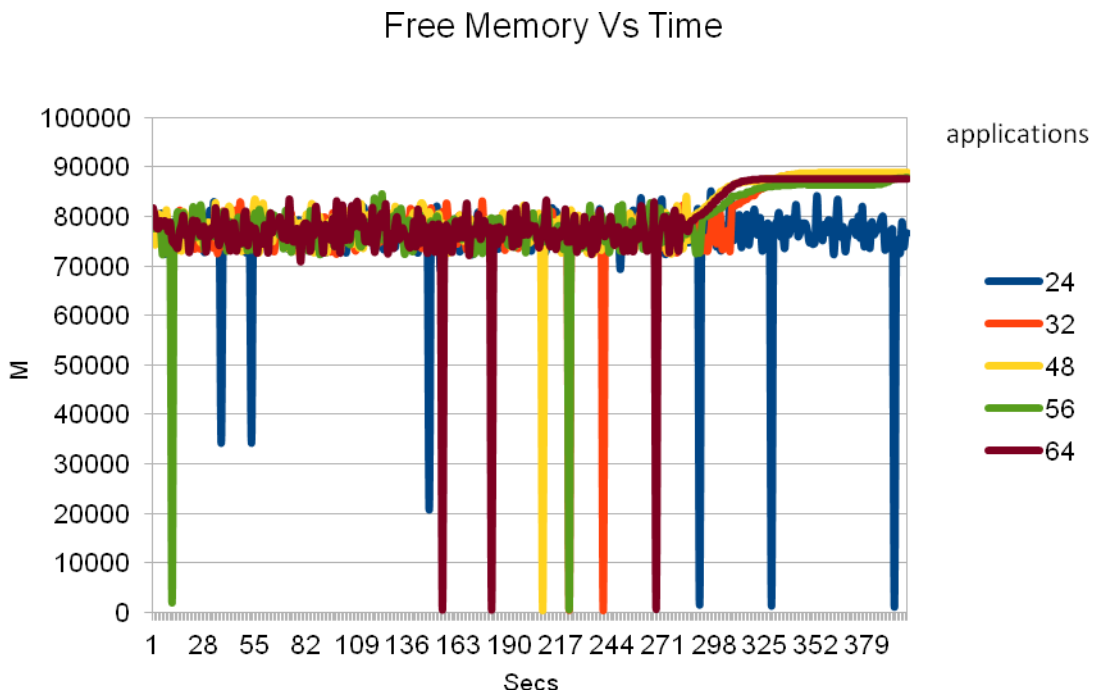


Fig 4.1.2.3: Typical Memory utilization from one of the representative nodes.

4.1.2.4 Throughput after Giga+ TableFS layering / path 2 model

Path 2 model should ideally provide near direct PanFS performance (800 MB/s) or path 3 performance, since it does not have the additional link overhead.

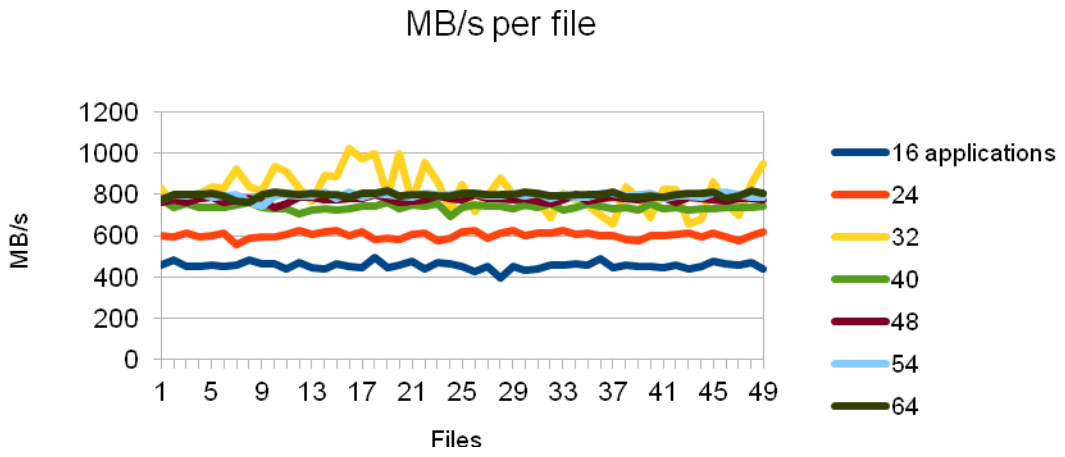


Fig 4.1.2.4: Data rate per file for large write workload (400MB per file) for Path 2

With an increase in the # of applications upto 32, we see an increase in data throughput reaching close to the hardware limits, despite of the 2+ user to kernel layer copy overheads.

4.1.2.5 CPU utilization with Giga+ TableFS layering

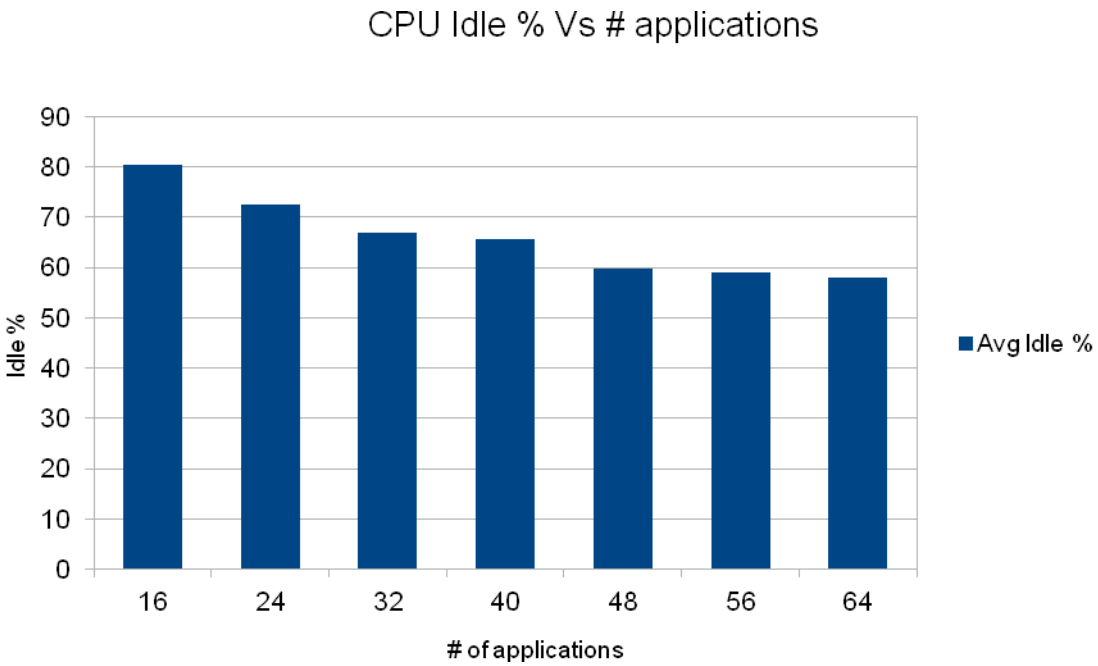


Fig 4.1.2.5: Typical CPU utilization from one of the representative nodes

CPU utilization is the average of various idle % values reported periodically by vmstat. As seen, in most of the cases the CPU utilization is about 10 % – 15% greater than the without-layering case.

4.1.2.6 Memory utilization with Giga+ TableFS layering

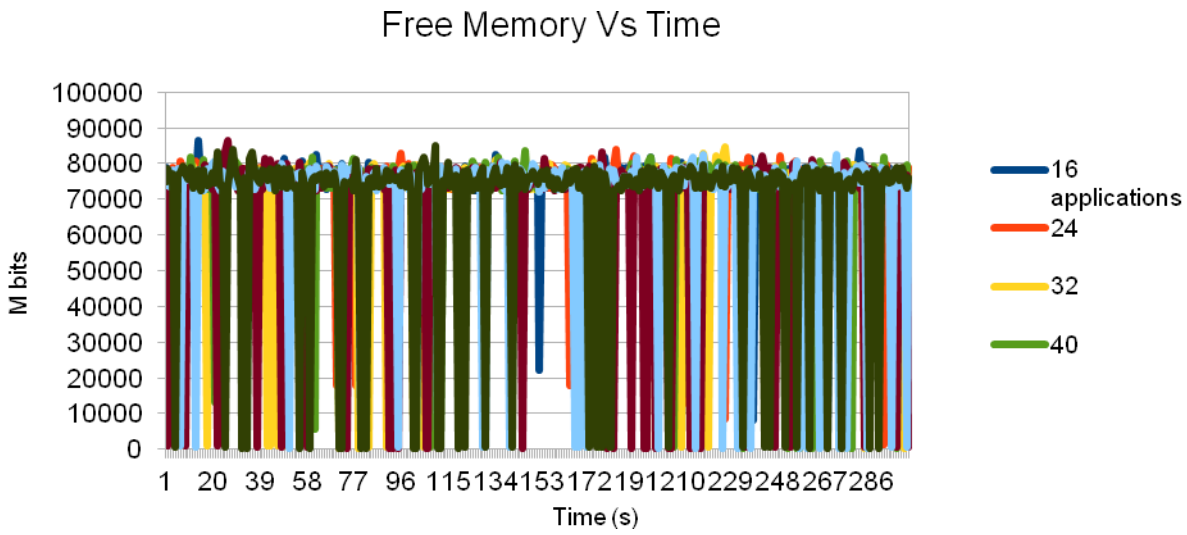


Fig 4.1.2.6: Typical memory utilization from one of the representative nodes from each set of clients

4.1.2.7 Throughput comparison with and without Giga+TableFS layering

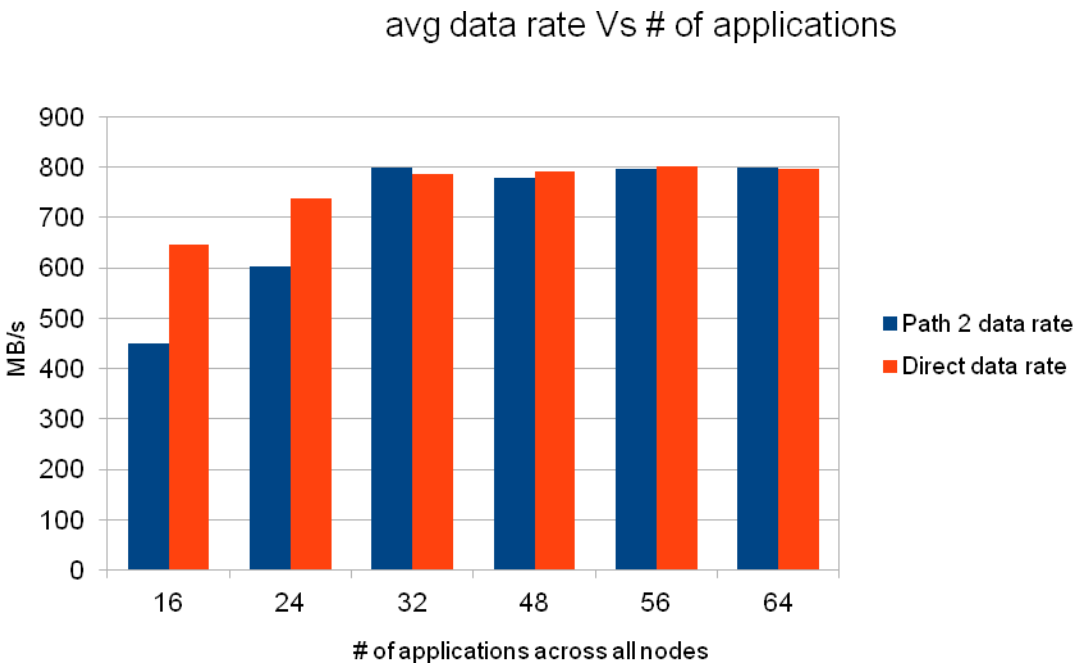


Fig 4.1.2.7: Comparison of data rate with and without Giga+TableFS

Path 2 data rate, due to its inherent overhead of memory copies and kernel context switching, falls behind the data rates of direct-PanFS and path 3. However, by allowing sufficient amount of parallelism by allowing higher number of parallel processes, we can see that Path 2 performs equally well. Path 2 performs equally well because in this exercise, we have enough CPU to saturate a 1 GE link even with copy overhead. However, in case of higher capacity links, this will not be held true.

The maximum average data throughput by both with and without layering Giga+ TableFS is around 800 M Bytes / sec for an 8 node cluster with each machine having a 1 GE Network Interface Card.

4.1.2.8 Reflection from PanFS visualization tool on the data rate of each of the nodes

Fig 4.1.2.8 shows the network-data utilization by DirectFlow clients on each of the nodes during the course of the data performance evaluation. We can see that all the nodes are almost fully utilizing their network IO capability and hence accounting for the maximum data throughput achievable.

Client		Ops/sec	MB/sec	Response Time (msec)
172.19.149.50	*p9532.pdl.cmu.local*	240	111.19	114.68
172.19.149.129	*p9581.pdl.cmu.local*	238	110.22	118.39
172.19.149.179	*p95b3.pdl.cmu.local*	235	109.51	118.72
172.19.149.163	*p95a3.pdl.cmu.local*	231	107.63	114.08
172.19.149.67	*p9543.pdl.cmu.local*	231	107.20	116.88
172.19.149.87	*p9557.pdl.cmu.local*	230	108.92	117.57
172.19.149.252	*p95fc.pdl.cmu.local*	228	108.60	116.29
172.19.149.127	*p957f.pdl.cmu.local*	163	87.89	155.49

Page generated: 05:45:04 December 15, 2012

Terms & Conditions : Site Map : Home

panasas

Fig 4.1.2.8: Panasas DirectFlow statistics on the 8 nodes

5 Results and Conclusions

1. The Giga+ TableFS – distributed metadata service can be layered over existing cluster filesystems without sacrificing much of the direct data bandwidth provided nodes have sufficient CPU power, at the same time exploiting the benefits – scalable and high performing metadata service, avoiding metadata hotspots, small file packing using leveldb etc.
    1. We see path2 and path3 providing matching data performance compared to direct access.
    2. We also see a clear advantage in metadata performance in the figures 3.3.2.1 and 3.3.2.2.

2. The underlying cluster file system gear's configuration impacts the layering of Giga+ metadata service layer. The number of MDS, volumes and their mapping, and metadata centralization scheme need to be taken into account while designing the file placement and distribution logic of the Giga+ layer. In case of PanFS, 3 such details are required to be considered:
    1. Metadata accesses to a particular directory are centralized: It is important to keep the number of dentries in a single directory small. It helps to split a directory as the number of dentries or dentry-access-heat increases.
    2. Metadata accesses to a particular volume are served by a single MDS: In case of high parallel metadata workloads, it helps to distribute the files across various volumes.
    3. A group of Volumes may be managed by a single MDS: In case of high parallel metadata workloads, it helps to distribute the files across various MDS-managed-volumes.

3. The way applications interact with the filesystems (via FUSE) impacts the degree of decoupling we can bring about between the metadata and data paths.
    1. Since “getattr” is used as a tool to decide what FS operations to choose for the intended operation, it gets hard to completely isolate the metadata management from the underlying cluster file system to Giga+ layer leaving it only the data management.
    2. If “getattr” response from the Giga+ client abstracts a file as a symlink, applications chase it for read/write via the most direct path reaching it in the underlying cluster file system. This is Path 3 with the highest data rate. However the metadata operations in this case also get routed to the cluster file system and Giga+TableFS is not serving the metadata at all.
    3. On the other hand, if “getattr” reflects the file attributes as-is then along with the metadata operations, data operations are also routed through Giga+. In this case path 2 will be the best decoupling possible.

4. Although FUSE layer adds an absolute performance overhead of 15 – 20 % on the data operations, this overhead can be hidden by increasing the number of parallel operations.
    1. By increasing the number of Giga+ clients per VM, we could see that the aggregate data rate obtained matches the highest value possible for our setup.
    2. This increase in the parallel workload is under the premise that, by doing so we do not encounter any other resource crunch such as CPU, Memory etc.

5. Admin imposed infrastructure limitations on the storage cluster can be overcome by Giga+.
    1. We could see that path 3 and path 2 (for sufficiently high number of parallel workload generating processes) get a higher aggregate throughput than direct PanFS path
    2. The entire workload was writing to a few volumes of PanFS only, and hence (could have) was facing a configuration limitation on the subset of the storage servers/disks a volume can access.

Since Giga+ indirection layer, distributes the files across all the volumes, such configuration limitations can be overcome.

6 References

[1] Scale and Concurrency in GIGA+: File System Directories with Millions of Files. Swapnil Patil, Garth Gibson, USENIX Conference on File and Storage Technologies (FAST) 2011

[2] TableFS: Embedding a NoSQL Database Inside the Local File System. Kai Ren, Garth Gibson Carnegie Mellon University Parallel Data Lab Technical Report. CMU-PDL-12-103 May 2012.

[3] LevelDB: A fast and lightweight key/value database library. http://code.google.com/p/leveldb/.

[4] A case for scaling HPC metadata performance through de-specialization. Swapnil Patil, Kai Ren, and Garth Gibson. http://www.pdl.cmu.edu/ftp/HECStorage/patil-pdsw12.pdf

[5] 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. MIT Press, 2000.

[6] S. Ghemawat, H. Gobioff, and S.-T. Leung, "The Google file system," in Proceedings of the nineteenth ACM symposium on Operating systems principles, ser. SOSP '03.

[7] K. Shvachko, H. Huang, S. Radia, and R. Chansler. The Hadoop Distributed File System. In Proceedings of the 26th IEEE Transactions on Computing Symposium on Mass Storage Systems and Technologies (MSST '10), Lake Tahoe NV, May 2010.

[8] B. Welch, M. Unangst, Z. Abbasi, G. Gibson, B. Mueller, J. Small, J. Zelenka, and B. Zhou. Scalable Performance of the Panasas Parallel File System. In Proceedings of the 6th USENIX Conference on File and Storage Technologies (FAST '08), San Jose CA, February 2008.

[9] D. Tweed. One usage of up to a million files/directory. Email thread at http://leaf.dragonflybsd.org/mailarchive/kernel/2008-11/msg00070.html, November 2008.

[10] D. Roselli, J. R. Lorch, and T. E. Anderson, "A comparison of file system workloads," in Proceedings of the annual conference on USENIX Annual Technical Conference, ser. ATEC '00. Berkeley, CA, USA: USENIX Association, 2000.

[11] S. A. Weil, K. T. Pollack, S. A. Brandt, and E. L. Miller, "Dynamic metadata management for petabyte-scale file systems," in Proceedings of the 2004 ACM/IEEE conference on Supercomputing, ser. SC '04. Washington, DC, USA: IEEE Computer Society, 2004.

[12] D. Beaver, S. Kumar, H. C. Li, J. Sobel, and P. Vajgel. Finding a Needle in Haystack: Facebook's Photo Storage. In Proceedings of the 9th USENIX Symposium on Operating Systems Design and Implementation (OSDI '10), Vancouver, Canada, October 2010.

[13] 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 Proceedings of the ACM/IEEE Transactions on Computing Conference on High Performance Networking and Computing (SC '09), Portland OR, November 2009.

[14] S. Dayal. Characterizing HEC Storage Systems at Rest. Technical Report CMU-PDL-08-109, Carnegie Mellon University, July 2008.

[15] FUSE: Filesystem in Userspace. FUSE.sourceforge.net

[16] F. Chang, J. Dean, S. Ghemawat, W. C. Hsieh, D. A. Wallach, M. Burrows, T. Chandra, A. Fikes, and R. Gruber. Bigtable: A Distributed Storage System for Structured Data. In Proceedings of the 7th USENIX Symposium on Operating Systems Design and Implementation (OSDI '06), Seattle WA, November 2006.

[17] Panasas Hardware Architecture. http://www.panasas.com/products/hardware-architecture

[18] Open Cirrus, an open cloud-computing research testbed. https://opencirrus.org/

[19] Apache Tashi, a software infrastructure for cloud computing on massive internet-scale datasets. http://incubator.apache.org/tashi/
[20] Oracle Lustre File System. http://wiki.lustre.org/index.php/MainPage.