Using Comprehensive Analysis for Performance Debugging in Distributed Storage Systems

Technical Report UCSC-SSRC-07-05
May 2007

Andrew Leung aleung@cs.ucsc.edu
Eric Lalonde elalonde@cs.ucsc.edu
Jacob Telleen jtelleen@cs.ucsc.edu
James Davis davis@cs.ucsc.edu
Carlos Maltzahn carlosm@cs.ucsc.edu

Storage Systems Research Center
Baskin School of Engineering
University of California, Santa Cruz
Santa Cruz, CA 95064
http://www.ssrc.ucsc.edu/

Using Comprehensive Analysis for Performance Debugging in Distributed Storage Systems

Andrew Leung Eric Lalonde Jacob Telleen James Davis Carlos Maltzahn
University of California, Santa Cruz
{aleung,elalonde,jtelleen,davis,carlosm}@cs.ucsc.edu

Abstract

Achieving performance, reliability, and scalability presents a unique set of challenges for large distributed storage. Debugging issues can be daunting given the scale of these systems. Recent work has focused on fine-grained performance analysis; this insufficient for building a complete understanding of the system. To identify problem areas, there must be a way for developers to have a comprehensive view of the entire storage system. That is, users must be able to understand both node specific behavior and complex relationships between nodes.

We present a distributed file system profiling method that supports such analysis. Our approach is based on combining node-specific metrics into a single cohesive system image. This affords users two views of the storage system: a micro, per-node view, as well as, a macro, multi-node view, allowing both node-specific and complex inter-nodal problems to be debugged. We have implemented a prototype profiler in the Ceph distributed file system with a focus on efficiency, portability, and scalability. We visualize the storage system by displaying nodes and intuitively animating their metrics and behavior allowing easy analysis on complex problems. We evaluate the overhead and scalability of our prototype profiler and find it contributes little overhead and easily scales to storage systems of over 1,000 nodes. Our visualization has allowed us to uncover several important performance issues within Ceph, ranging from poor load management to supringly strong correlation between, supposedly decoupled, metadata and data operations.

1 Introduction

The complex nature of distributed storage increases debugging complexity. Moreover, performing root-cause analysis is complicated in large-scale storage systems where data and load are highly decentralized and the number of possible causes is large. A single I/O request can traverse user and kernel-level code on many different machines, and its performance is dependent on many factors such as load and cache state. This makes understanding and debugging file system behavior extremely difficult.

Distributed storage performance have two classifications: node-specific issues (problems either occurring at, or relevant to, a single node) and inter-node issues (problems caused by relationships with other nodes). Debugging node-specific problems has been researched for many years with much success [2, 5, 13, 16, 28]. Understanding inter-node problems has become an interesting challenge explored more recently [3, 8, 10–12, 16, 17, 22, 24]. The key problem facing current approaches is an inability to achieve comprehensive analysis of the *entire* storage system. In particular, a complete view of the storage system includes all node-specific events, as well as, complex inter-node events.

Understanding inter-node problems has proven difficult because these problems are: (1) distributed, the source of a problem may be far removed from where its effect is observed, (2) opaque, the number of nodes obfuscates the problem source, and (3) sporadic, the problem may only occur on a few nodes or only under specific workloads. For example, storage device *A*'s high I/O latencies may be due to local performance issues or may be caused by other nodes. *A*'s performance problems may be the result of another device mirroring data onto *A*, *A* replicating data onto a slower device, or *A* having to perform recovery caused by another device failing. We assert that in order to fully understand and debug distributed storage, both node-specific and inter-node events must be analyzed.

Profiling is a common technique for identifying performance issues, understanding behavior, and debugging problems in any system. Current profiling techniques for distributed systems take a fine-grained approach via activity tracking and resource accounting. These have proven useful in revealing single-path bottlenecks and building workload models. However, such a fine-grained approach faces difficulties when attempting to correlate inter-node

relationships with performance. In this case both micro (node-specific) and macro (inter-node) analysis of the system is necessary to achieve a complete understanding of performance.

The current standard for visualizing system performance is to log and graph relevant performance counters. This is appropriate when seeking knowledge of individual metrics, but as the size of a system grows, the usefulness of these techniques diminishes. For example, users can easily view the throughput of any single node as a graph and be satisfied, but the log and graph approach fails when the goal is to convey more complex concepts such as how an individual node failure impacts overall resource availability.

We take a comprehensive approach to profiling, analyzing both micro and macro behavior, and offer a more robust view of the system than standard log and graph techniques. We profile the system by running a *visualization client* on each node. The client is responsible for collecting node-specific instrumentation data and local machine statistics. Data is forwarded to a cluster of *visualization servers* which use timestamp information to serialize data from all nodes into a single, cohesive stream of system events. This serialization enables cause and effect analysis of distributed performance problems. The ordered stream is then fed into a *visualization application* which uses computer animation to intuitively animate represent system activity and behavior in real time.

We have implemented our profiler in the Ceph petascale, distributed file system [25] along with a prototype visualization application. Our prototype visualizes storage behavior by animating all nodes in the system and their various system metrics and activities. For example, we animate CPU utilization as a changing color scale and charaterize metadata operations via changing pie charts. Our animation allows users to see all nodes in the system and easily understand how those nodes are operating. By viewing multiple nodes at once, users can easily understand complex inter-node relationships. For example, the effects of mirroring data become obvious because changes in performance can easily be identified. Evaluations of our profiler indicate a very limited overhead and scalability in storage systems over 1,000 nodes. Using our visualization we uncovered several important performance issues in Ceph. These issues range from poor load balancing of metadata operations to a suprisingly strong correlation between, supposedly decoupled, metadata and data operations.

The remainder of the paper is organized as follows. Section 2 discusses related profiling and visualization work. Section 3 presents the design goals behind our profiling and visualization. We discuss the design and implementation of our system in Sections 4 and 5. We present our experiences profiling the Ceph petascale, distributed file system in Section 6 with a performance analysis in Section 7. We discuss future work in Section 8 and conclude in Section 9.

2 Related Work

Profiling and benchmarking storage systems is a heavily researched topic, though distributed storage present many unexplored challenges. Significant inroads have been made in tracing and profiling local file systems which reside on a single machine [2, 5, 13, 16, 28]. Latency analysis and system instrumentation are common methods of aggregating data, while performance counters are used to express node performance. This is insufficient for a distributed environment, where understanding the inter-node performance relationships is critical to problem resolution.

Previous research in profiling general distributed systems addresses the scenario where applications are black-box entities whose source code cannot be viewed or instrumented (often for proprietary reasons) [1, 6, 16]. When high-level graph construction is sufficient for profiling, this approach is quite useful. However, in a distributed storage system, where capturing internal state is critical to establishing an accurate view, the necessary instrumentation makes a black-box approach inadequate.

Other existing techniques focus on fine-grained profiling and end-to-end request tracing is used in order to identify bottlenecks [11, 12, 16, 22], model workloads [18, 24], and identify anomolies [20] in a distributed environment. Systems like Magpie [3] and Pinpoint [7] use statistical modeling to infer performance outliers. They rely heavily on trace files, which limits their ability to identify bugs that are common to all paths in a system. This approach also assumes that outliers are necessarily the result of bugs. Other approaches like Pip [21] require that the users specify their expectation for how a workload should perform. This can be a complex task, and puts the burden on the user to have a deep knowledge of the workload.

Another profiling approach uses statistical correlation to understand which performance thresholds are related to violations in Service Level Objectives (SLOs) [8, 9]. This work has similar objectives to our own, in that the goal is to identify states which relate to performance degradation. However, this work relies on having a precise definition of performance violations, which a SLO affords. In contexts such as scientific computing environments, the lack of such a definition puts the administrator in the po-

sition of arbitrarily defining service objectives. Instead it is more important to know whether a performance degradation is the result some specific inefficiency, or whether the system as a whole is simply experiencing high utilization.

We rely on animation to visualize system data because it can take raw data and manipulate it so that recognizable patterns begin to emerge, which makes management of resources and trend analysis easier [15]. This allows the user to see subtle interactions that can easily be overlooked by other methods of data exploration. Visualization can also be used for prediction and intuitive troubleshooting.

There have been numerous systems which rely on visualization techniques to analyze systems data. Rivet [4] is an architecture for flexible and extensible visualization of generalized distributed systems. Because of their focus on general distributed systems, Rivet requires additional data transformation stages to generalize data, which adds overhead. Visualization is done through showing pipelines of system events, statistics, and graphs. While useful for general distributed systems, it is inadequate for storage systems which require visualization of both time-coordinated high level system state as well as low-level information. NetLogger [14] is a methodology for generating precise performance event logs and visualizing the aggregate data for performance analysis. It can profile any arbitrary metric chosen by developers, and coordinates log files to a central repository. NetLogger's log-and-graph approach to visualization makes root-cause analysis difficult in a distributed file system, where the problem source is often removed from where its effect is felt. Further, the delay associated with creating NetLogger event logs is prohibitively expensive for a distributed file system, where I/O latencies usually measure in microseconds.

Several systems use call-graphs to build the path that requests take through the system, allowing bottlenecks along the path to be revealed [1, 3, 19, 20, 23, 24]. This approach has proved useful for identifying inefficient software components and slow nodes in a network. Unfortunately, is a call-graph only provides information about a single path in the system and is unable to build a comprehensive view of how many nodes are interacting at once.

Ganglia [17] is a distributed monitoring project that has a similar approach to visualization as our own. Ganglia is able to monitor systems using scalable techniques that introduce low processing overhead to each system. We take a similar approach in how we represent individual system nodes. One major difference between our work and Ganglia is that Ganglia does not afford the user with a method of correlating metrics on separate machines. If Ganglia reports that two machines are experiencing high resource utilization, one must still log into those machines and analyze each independently to figure out why. Answering questions regarding inter-node relationships is a primary goal of our work, and to that end our design goals supports this type of analysis directly.

3 Design Goals

The design of our profiler and visualization was motivated by several goals:

Low Overhead: In order to achieve accurate performance metrics, profiling must introduce minimal overhead. Therefore, introducing major computations along critical paths must be avoided.

Scalability: As distributed storage systems grow larger, scalability becomes a major design focus. With large systems comes an increases the number of points being profiled and the amount of data being collected. This requires that the profiling infrastructure be able to grow with the size of the storage system and that visualization techniques continue to present intuitive results even as the amount of data becomes large.

Portability: The utility of any distributed profiling system is bounded by the range of nodes on which it can be applied. For this reason, portability is a key design criteria. While points of instrumentation will differ between nodes and file systems, the method for gathering the information should be completely abstracted from the file system itself. As a result, simple instrumentation should be the only modification requirement.

Comprehensive Analysis: Distributed storage systems have complex relationships with many nodes which affects performance. This complexity is exacerbated as system size increases. To be effective, it must be feasible to quickly and easily understand large amounts of data and the effect of node interactions. At the same time, understanding the performance of each individual node remains integral. Therefore, any visualization must make both of these macro and micro metrics intuitive to the user.

4 Methodology

To achieve our design goals, we take a distributed approach to profiling. Each node in the file system runs a local *visualization client*, which is responsible for profiling local storage system and machine information. Periodically, each visualization client updates a *visualization server* with recent changes to the local node. The server is

responsible for chronologically ordering the data to produce a single, serialized stream of system events from all nodes and sending this serialized sequence to a *visualization application*. The visualization application visually represents nodes in the system and displays or animates their metrics and behavior. This provides an intuitive interface where both node-specific and complex inter-node behavior are easily understood. We discuss this process in detail throughout this section.

4.1 Visualization Client

The visualization client is implemented as a user-space process which collects instrumentation data and machine statistics. This design provides two key benefits. First, the client does not add overhead to critical paths because it relies only on instrumentation data and can reside outside of the storage system. This improves development time and ensures that the client does not interfere with system performance. Second, by only requiring instrumentation data, the client can profile any instrumented part of the storage system. This greatly improves portability and allows profiling of components in user and kernel space. For example, profiling a kernel-level file system only requires that the file system log data to a shared file which the visualization client can read. This is illustrated in Figure 1, where we see that the visualization client can profile user level file systems without interrupting calls to VFS. Additionally, the client can profile a kernel-level file system, such as NFS, by simply requiring that NFS log instrumentation data to a source accessible to the client. The benefits we have discussed indicate that how and where the system is instrumented is critical to the robustness of the profile. Therefore, regardless of where the storage system resides, building a robust profile is dependent on accurate system instrumentation.

We have implemented the visualization client as a Java RMI client. The client profiles the local node by using two methods of data collection. The first method is concerned with metrics which are common to all nodes in the storage system, such as system load and network utilization. This node-agnostic method is performed by a thread which periodically polls local OS resources like `/proc/loadavg`. The second method is node-specific, and depends on the role that the node plays in the storage system (e.g., client or server). These events are captured from instrumented code in the storage system itself. The visualization client stores collected metrics in a local database. Periodically, a separate communication thread polls this database to aggregate events that occured during the previous poll interval. This aggregation is then time

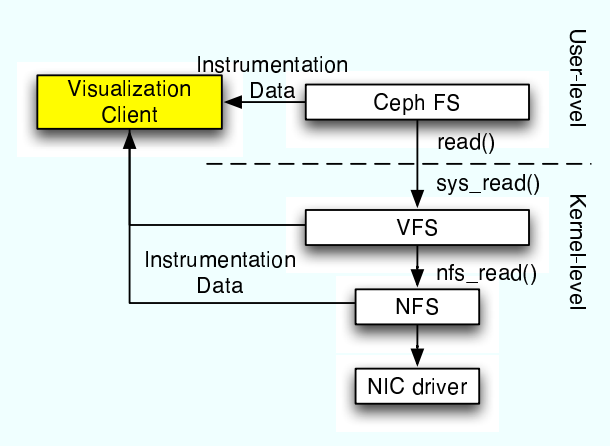


Figure 1: The visualization client allows profiling user and kernel-level file systems.

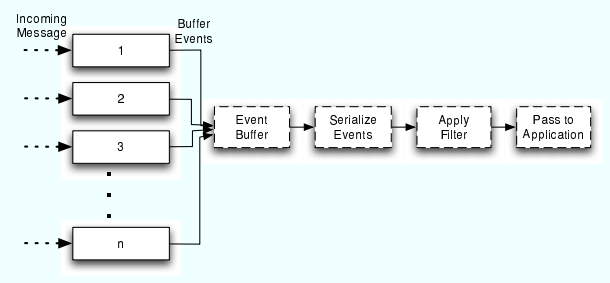


Figure 2: A high-level pipeline of visualization server operations.

stamped and sent to the visualization server over RMI. The communication period is kept short to ensure that information sent to the visualization server is fresh.

4.2 Visualization Server

In order to achieve a complete view of the system, metrics from all nodes must be aggregated to a common location. A cluster of visualization servers is responsible for receiving and organizing data from all nodes in the system and for passing a serialized ordering of system events to the visualization application. Event ordering is achieved by comparing the time stamps of events received from all visualization clients. We assume that each node in the system is roughly synchronized.

The key benefit of centralizing metrics at the visualization servers is that it allows visualization clients to act independently of each other, requiring the client's responsibility to simply be collecting and forwarding metrics to the server. Requiring the visualization clients to coordinate in order to serialize events would introduce major complexities and add overhead from to the extra communication.

System metrics are organized into a cohesive, time-ordered series by the visualization servers based on timestamps. The reason for this is two-fold. First, events in a distributed storage system have many cause-and-effect relationships. For example, a write at a storage device can be the result of numerous metadata operations causing a metadata server to flush its journal. As such, understanding these events is directly dependent on understanding their ordering. Second, serializing data at the visualization server allows the design of the visualization application to be simplified. For example, the visualization application is designed using an event-driven model where it simply displays events are they are received.

A consequence of ordering events at the visualization server is that events can be received out-of-order. If this issue were not addressed, events would be passed to the visualization application in the wrong order. We address this by batching events at the server before sending them to the visualization application. By batching for a short period (less than a second) the server can receive and order a number of events, ensuring that all events received within the buffer window are passed to the visualization client in the correct order. Long buffer windows cause a large number of events to be correctly ordered, but also imply that the visualization application will receive updates less frequently. The merits of this tradeoff varies between systems, depending on how important event ordering is.

The main difficulty with clustering is that each server cannot create a complete ordering of events for a buffer window because visualization clients may communicate with any server. To address this, servers communicate all information for a specific time interval to a specific server who is the authority for that interval. For example, all servers may forward data that is timestamped between logical time `1500.0` and `1600.0` to a specific server where all events for that period can be correctly ordered. Authority for a time interval may be calculated via simple hash, mapping intervals to servers.

Another important scalability issue is the amount of data that is forwarded to the visualization server. If clients collect a large amount of data from each node, a system with a large number of nodes will overwhelm even a reasonably sized visualization server cluster. To alleviate this, the visualization server limits the amount of data sent by each client. The server only requires that clients forward data values of interest, which correspond to specifications from the visualization application. For example, when the visualization application is only analyzing storage device performance, the amount of data collected about storage system clients can be reduced. We are exploring further extensions to this which is discussed in Section 8.

We have implemented the visualization server as a Java RMI server. Figure 2 demonstrates a high-level view of the stages in the server pipeline. Before passing events to the visualization application the server applies a filter. The filter serves to limit the amount of data passed to the application. For example, if the user has chosen to focus the visualization application on a subset of system nodes, the server only needs to pass data for those nodes being displayed.

4.3 Visualization Application

The effectiveness of any performance debugging solution depends on its ability to easily and intuitively convey information to the user. There have been many approaches towards visualizing activity in distributed systems and several have been discussed in Section 2. While some of these solutions are not suitable for large-scale file systems, others may prove effective. In any case, the suitability of any visualization is determined by the user's goals. We have chosen to focus on a visualization method that easily presents both micro (node-specific) and macro (inter-node) metrics. As a result, we have chosen to use simple computer animation to visually represent the entire file system and animate system behaviors.

We have implemented an initial visualization prototype in C++ using the OpenGL 1.5 library. While our prototype is rudimentary, it serves as a proof-of-concept reference. The visualization application may or may not reside on the same node as the visualization server. As such, the server may stream metrics to the visualization application via IPC, sockets or a shared file. Nodes are animated by glyphs corresponding to the role of the node in the system (e.g., client or file server). All collected metrics and measurements correspond to animations which are displayed relative to their node glyph. For example, a file server's I/O characterization may be shown as a dynamically changing histogram adjacent to the glyph. Alternatively, a node's system load may be animated by changing the color of the node's glyph. Users can view any subset of nodes or metrics in order to improve comprehension of large-scale systems. We discuss further details of our implementation in Section 5.

Visualizing each node in the system and animating node metrics and behavior provides several features that make it a reasonable approach for distributed storage systems. First, animations are easy to understand. At a glance, users can understand changing colors or shapes far more intuitively than logs or static graphs. Second, users

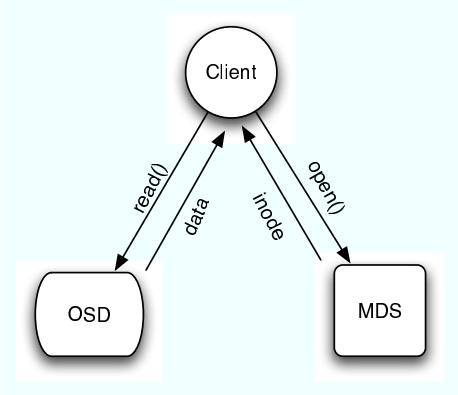


Figure 3: Parallel file system architecture and data flow.

Node Type	Metric	Animation
MDS	System Load	Color Map
	Operation Breakdown	Pie Chart
Client	Network Traffic	Arrows in/out
OSD	System Load	Color Map
	Network Traffic	Arrows in/out
	Disk Utilization	Percentage
	I/O Size Breakdown	Histogram
	Write latencies	Values

Table 1: The metrics collected by the visualization client and their corresponding representation in the visualization application. Our analysis has emphasis on OSD and I/O performance.

can validate inter-node relationships. For example, if file I/O causes a node to become heavily loaded, one can identify that file's mirror because it will also be heavily loaded. Third, users can narrow their view to focus on the performance of a single node or set of nodes. This supports arbitrarily tailoring analysis as the user sees fit. Fourth, viewing multiple nodes allows users to validate observations. For example, analysis of high I/O latency can determine whether the issue is occurring on more than one node, whether the latency is comparable on all nodes involved, whether load is comparable on all nodes, and whether the high latency persists over time. Fifth, visualization can be done in real-time or offline. By profiling and viewing the system as a workload runs in real-time, users can monitor performance and identify problems early. By recording log data and replaying it, users can diagnosis previously observed performance issues. Our analysis and results in Section 6 support these beliefs.

5 Implementation Details

We have implemented our performance debugging system in the Ceph petascale, distributed file system [25]. We chose Ceph because it is large-scale (designed for petabytes of data and tens of thousands of nodes), supports high performance computing workloads, has several unique design features, and is currently in prototype status. This means problems are likely abundant and analysis is helpful to current designers.

Ceph is designed as an object-based, parallel file system. These systems generally consist of three main components: the client, a metadata server cluster (MDS), and a cluster of object storage devices (OSD). These systems achieve scalability and performance by separating the control and data paths. Clients communicate all namespace operations, such as `open()` and `stat()`, to the MDS and all file I/O operations, such as `read()` and `write()`, to the storage devices. Large-scale systems may contain tens of thousands of clients and storage devices and hundreds of metadata servers. Figure 3 shows the parallel file system architecture and data flow.

Our system is particularly well suited for profiling parallel file systems because they present several unique features. First, file data is highly distributed, commonly striped across many storage devices. This means profiling any single device is insufficient. Second, the control path is decoupled from the data path. As a result, MDS profiling techniques are inherently different from storage device profiling techniques. Third, location of data and load-balancing play major roles in how a system performs and scales. This implies the importance of replication policies, failure handling, and other reliability techniques. These factors motivate the need for a macro view of the entire file system and the need for cause and effect relationships to be correlated in time.

Ceph is designed around several novel concepts which make it an excellent system to evaluate. Ceph utilizes a pseudo-random data placement function, called CRUSH [26], which scales better than traditional allocation tables. The separation between metadata and data is maximized by pushing responsibilities, such as object serialization, to intelligent OSDs. Finally, load-balancing at the MDS is handled by dynamically assigning responsibility for namespace sub-trees [27]. All of these concepts are quite new and therefore their implications have yet to be fully understood, meaning Ceph provides an excellent profiling test subject.

Our visualization client collects different metrics depending on the type of node it is profiling (i.e., client, MDS, OSD). In addition, a visualization client runs on each node in Ceph's Monitor cluster, which is responsible for managing the MDS and OSD clusters and for bootstrapping clients. The Monitor is used to learn of nodes

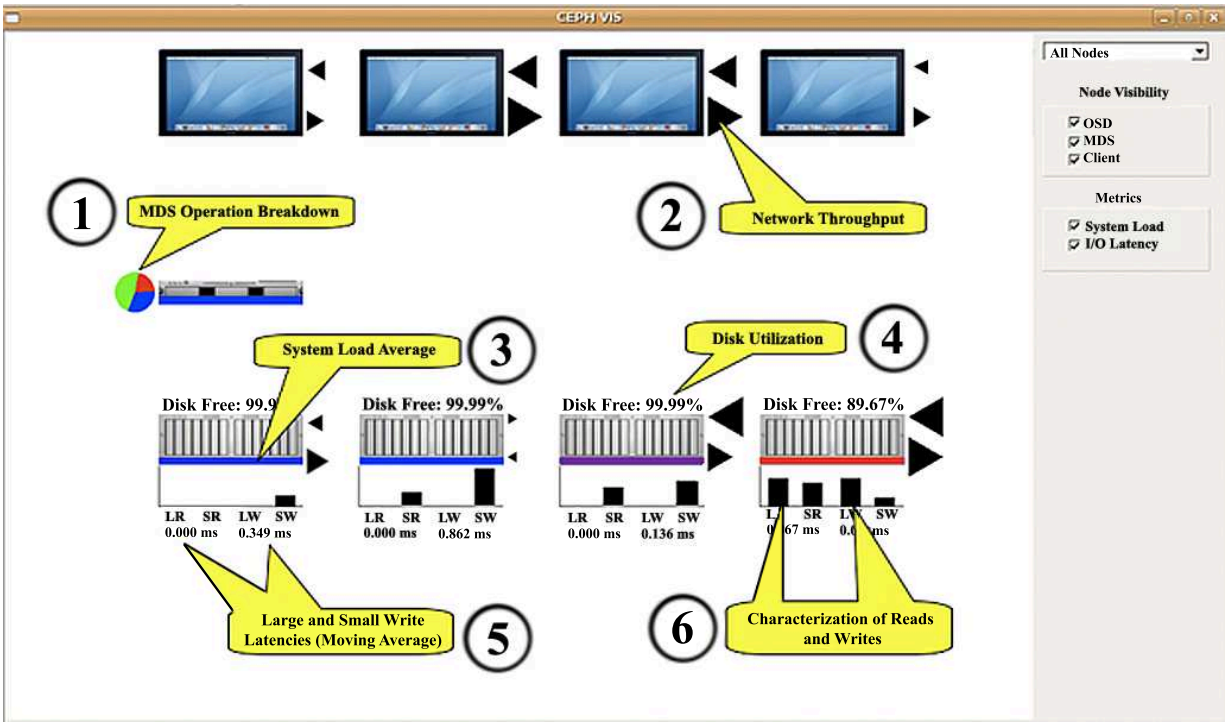


Figure 4: A labeled screenshot of the visualization application.

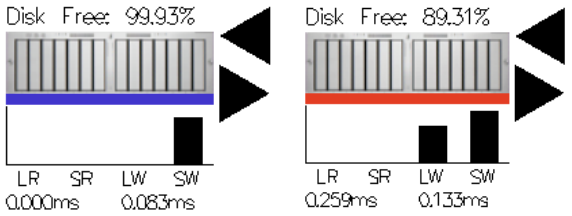
entering or leaving the system or changing state (i.e., not responding but not confirmed failed). These events are sent to the visualization server as architectural updates. The visualization client collects several simple, but useful, metrics from other nodes in the system, with an emphasis on OSD performance. These metrics are enumerated in Table 1.

In our visualization application users are able to toggle the metrics being displayed and which nodes are in focus. Figure 4[1] shows a screenshot of the visualization aaplication with numeric labels and descriptions added. Clients are represented on top, with MDSs following, and OSDs on the bottom. Table 1 also details the animation used to present each metric. Network traffic, labeled 2, is shown as triangles pointing in and out, which grow and shrink as traffic varies. System load average, labeled 3, changes from blue to red, indicating low and high load respectively, as the load changes. Each MDS has a pie chart, labeled 1, showing a breakdown of the number of `open()` (blue), `readdir()` (red), and `stat()` (green) operations received. Each OSD shows a breakdown of I/O by type and size, labeled 6, with a kilobyte acting as the cutoff between large and small I/O. Below the I/O breakdown, labeled 5, is the moving average of large and small write latencies, respectively. Disk utilization, labeled 4, is shown above each OSD as the total percentage of free space used. The menu on the right of Figure 4 allows users to toggle the nodes and metrics in view.

[1] All screenshots have a corresponding gray-scale version that can be included.

6 Profiling and Visualizing the Ceph Petascale, Distributed File System

We conducted a study on Ceph to evaluate the ability of our profiling and visualization techniques to aid in debugging performance. Our experiments focus on the visualization application's ability to reveal inter-node performance problems. For each performance issue revealed, we validate our observation through additional experiments. Each study was conducted on a 25 node cluster where each node was a PC with four 64-bit 2GHz Dual-Core CPUs, 8GB of RAM, 4 SCSI hard disks, connected through a 10 Gigabit Extreme Switch and running RHEL 4 with kernel version 2.6.9. All experiments were conducted with the Ceph client cache disabled as the write back policy resulted in high variability between runs. Each experiment used 4 OSDs while the number of MDSs and clients varied between experiments. A single visualization server was run on a seperate node with the visualization application also residing on that node. To save page space, the figures in our analysis only include the key portions of our visualization.



(a) Write latency for 80 clients writing a shared file using small write sizes.

(b) Write latencies for 80 clients writing a shared file using small write sizes and 32 clients writing non-shared files using large write sizes.

Figure 5: OSD profiles for workloads with varying I/O sizes. Small write latencies significantly increase when another workload is introduced.

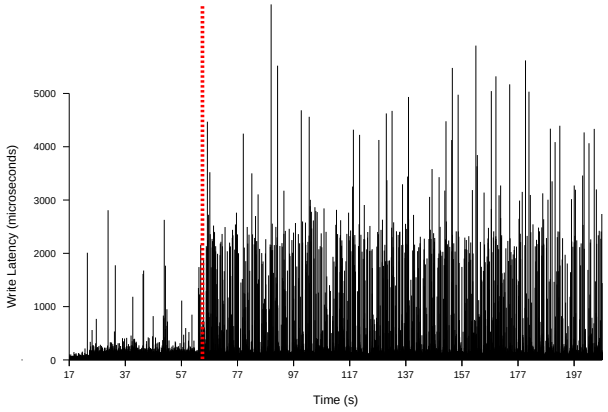


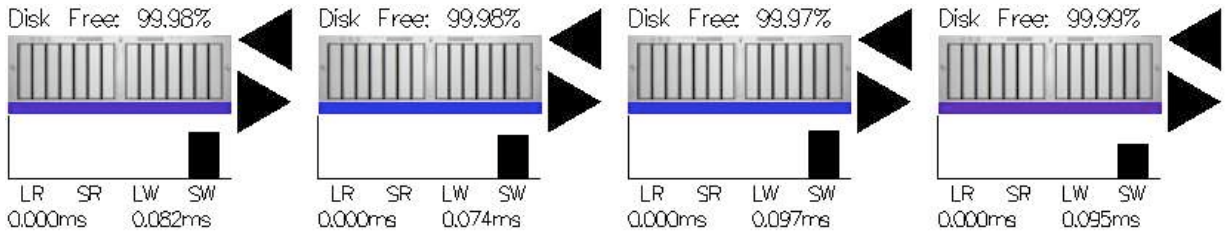
Figure 6: The latency for small writes on an OSD. Latency significantly spikes at 65 seconds when large I/Os are introduced. This is denoted with a dotted line.

6.1 Small Write Latency

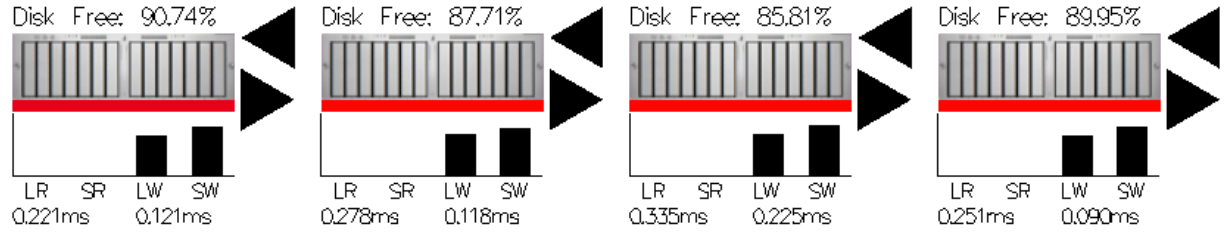
We ran an experiment to analyze the impact writes of varying sizes play in the storage system. We ran an experiment with 80 clients writing a shared file on a single OSD using a small write size just under a kilobyte. The profile of this single OSD is depicted in Figure 5(a) and shows average small write latencies on the order of 80μs. We then introduced another 32 clients, each of whom wrote non-shared files in large 1MB chunks. The profile of the same OSD is shown in Figure 5(b). Introducing the large write workload served to dramatically increase the latency of small writes, far beyond expectations. Small write latency increased on the order of 60% and greatly increased load on the OSD. While introducing an additional 32 clients should add overhead, it is not expected to be so severe, particularly because Ceph's data placement algorithm is designed to distribute I/O evenly. We conducted a separate experiment to validate our observations. We ran the same workloads, introducing the 32 clients issuing large writes 65 seconds into to the experiment, and plotted the latencies for small writes in Figure 6. We see a significant spike in latency for small writes at time 65 when large writes begin. This indicates what is most likely an implementation inefficiency in Ceph's OSD code. This also serves as a simple example of how allows easy problem identification.

6.2 Metadata and I/O Separation

Our second profile explores a more complex example of inter-node performance problems. The profile consisted of examining the effects various workload types have on performance. We began by running a workload consisting only of metadata operations where 50 clients each created a large tree of directories and files, and then walked and read the entire tree. We observed a large number of small writes being written to the OSDs, depicted in Figure 7(a). Upon further investigation we discovered the MDS's metadata journal was being synchronously flushed to the OSDs on every metadata operation to ensure the logs reliability. Then we introduced a second workload where 20 clients ran I/O heavy operations, in which each client wrote a gigabyte to a unique file. The only metadata operations issued were to open and close the files. Our visualization of the OSDs under both workloads in Figure 7(b) shows a much higher load on each OSD, and again the latency for small write operations (the journal flushes) significantly increased, which we witnessed in the previous experiment. To validate our observation and analyze the impact of the high latency journal flushes, we compared the time required to run the metadata only workload with and without the additional 20 clients performing I/O. Our results are shown in Figure 8. The metadata workload is over 60% slower when there are additional clients performing I/O. This overhead is due to the added latency of journal flushes slowing the performance of metadata operations. This is suprising because it conflicts with the general intuition that metadata and data operations are decoupled in parallel file systems. This dependency can be eliminated by bypassing the MDS's need to store the journal on the OSDs, perhaps via reliable NVRam or a separate journal store. This experiment demonstrates how a comprehensive view of the entire system allows problems with seemingly remote causes to be identified.



(a) OSD activity with a metadata-only workload running.



(b) OSD activity running a metadata workload and a I/O workload which issues large writes.

Figure 7: OSD profiles under a metadata only and a metadata and I/O workload. The increased load and latency indicate the I/O workload is interfering with the metadata workload.

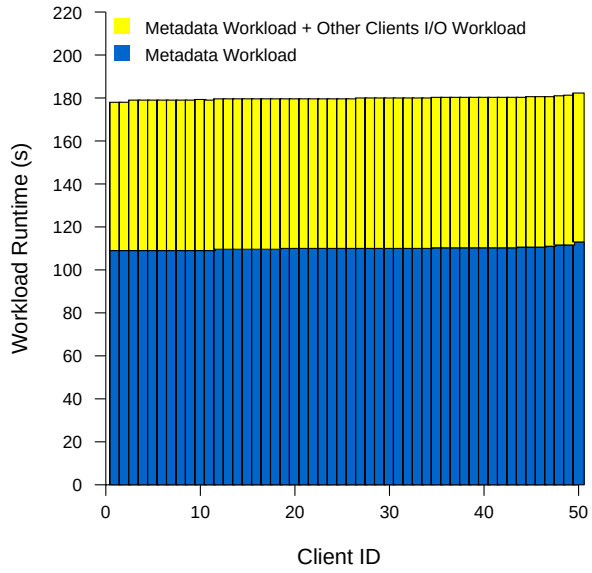


Figure 8: The average time for 50 clients to run a metadata exclusive workload with and without 20 additional clients performing an I/O exclusive workload.

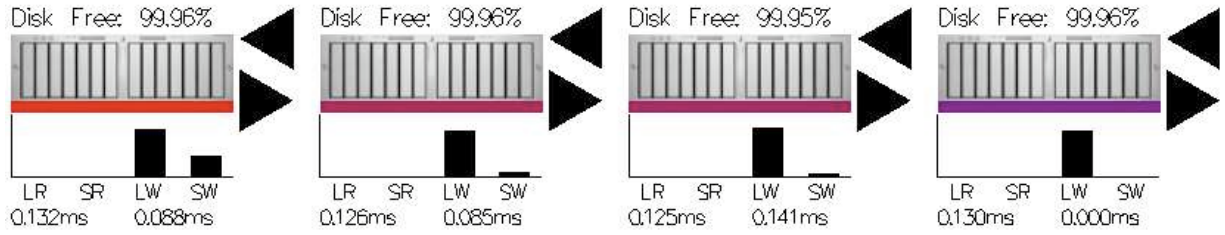
Figure 9: Load distribution across three metadata servers during a flash crowd workload. One MDS signifigantly more load than the others.

6.3 MDS Load Balancing

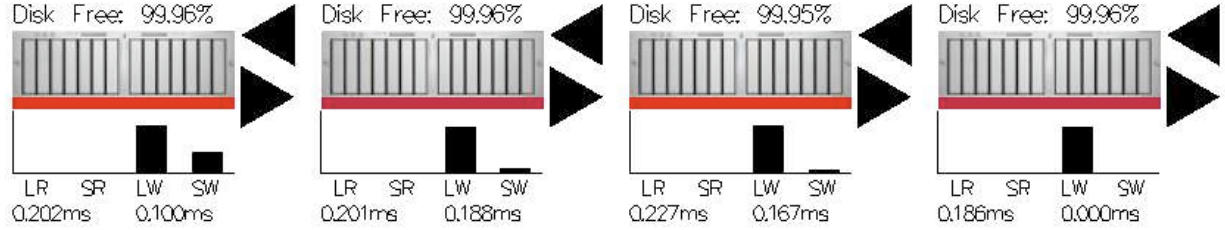
Ceph employs an advanced metadata load balancing scheme called Dynamic Subtree Partitioning [27]. DSP allows a MDS node to dynamically share responsibility for a hot or popular portion of the namespace with another, less loaded, MDS node. We tested Ceph's implemenation of this strategy using a flash crowd and 3 MDS nodes. The flash crowd consisted of over 11,000 total open requests from 2,000 clients. Figure 9 shows the MDS load distribution as depicted by our visualization. The pie chart next to the first two MDSs indicate each node only received open requests, while the third MDS has not received any requests. We immediately see one MDS is far more loaded than the other two. We investigate further by measuring the number of requests received by each OSD, shown in Figure 10. The distribution of load is very uneven, with one MDS handling over 90% of the requests and the third handling none at all. This indicates implementation inefficiencies with the load balancing policy, such as infrequent exchanges of load metrics between MDS nodes or slow migration of the namespace to other MDS nodes.

6.4 Replication Policy Impact

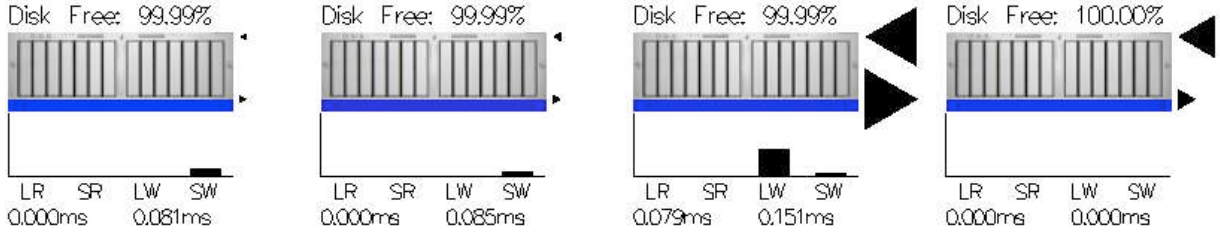
We conducted a final experiment which analyzed the role replication plays in storage performance. We first ran a workload with 50 clients writing non-shared files using a large write size just over a kilobyte. This was run twice, once with a single replica and once with three replicas. Ceph mirrors each replica onto separate OSDs, though our visualization does not directly show replication. This is because replication passes through a seperate interface which was not encapsulated by our intrumentation. The



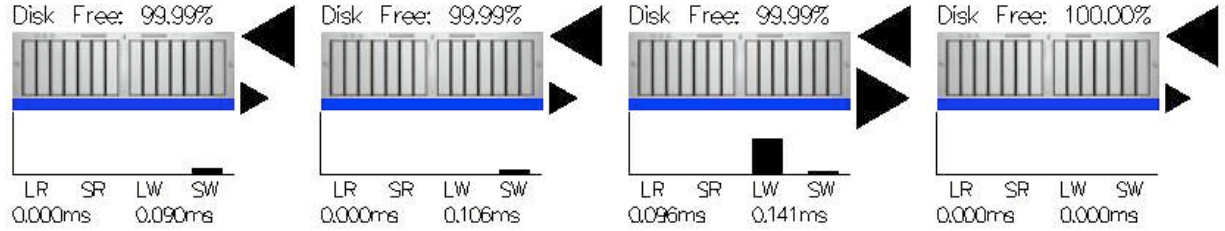
(a) Fifty clients with one replica.



(b) Fifty clients with three replicas.



(c) A single client with one replica.



(d) A single client with three replicas.

Figure 11: OSD profiles with different replication strategies. The added write latency to do three replicas is far larger with 50 clients than with 1.

replication is however noticable via analysis of the network interface. Each OSD receiving lots of data, without high write traffic, is acting as a mirror. The OSD profiles with different replication strategies are shown in Figures 11(a) and 11(b). We see the latency for both write sizes significantly increases, on the order of 60%, with three replicas versus one. While an overhead is expected, we explore further and re-run each experiment with only a single client. Our visualization is shown in Figures 11(c) and 11(d) with OSD 3 being written to in both cases. With a single client latency still increases, though the discrepancy between the two is far smaller. Average latencies are provided in Table 2. With a single client three-way replication adds a 37% overhead, while with 50 client, three-way replication adds a 59% overhead. If replication only added a cost to apply updates remotely we would expect the latency differences to be closer than those observed. Unfortunately this is not the case, as in addition to the cost of applying the replica, each node pays a cost for acting as the mirror for another nodes data. This adds additional overhead which can easily go overlooked with other visualization strategies. Our approach allows the increased latency and network traffic (due to replication) to easily

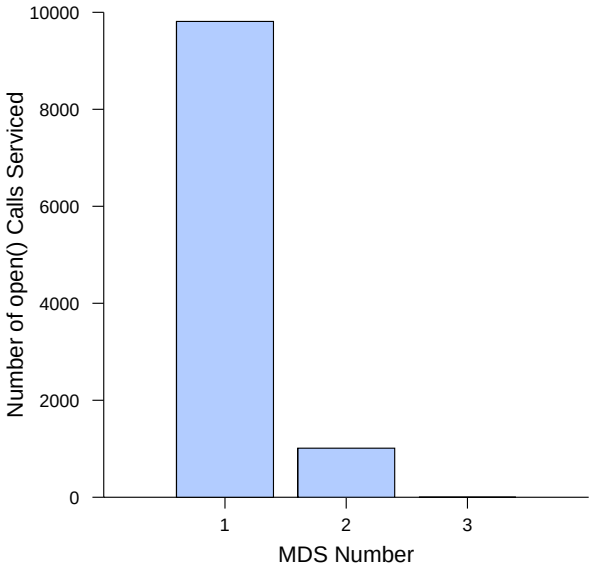


Figure 10: The number of open requests handled by each MDS during a flash crowd workload.

# Clients	# Replicas	`write()` latency
1	1	71.82
1	3	98.52
50	1	134.27
50	3	215.29

Table 2: The average write latency (in microseconds) for different replication policies. Adding more repliacs becomes signifigantly more expensive with more clients.

be seen together, which opens the door for more complex problems to be revealed.

7 Performance Evaluation

We evaluated the overhead and scalability of our profiler in Ceph in several experiments. Experiments were conducted using the same 25 node cluster, though 12 OSD were used instead of 4. Our results indicate that our basic prototype profiler is able to scale to reasonably large systems ($>$ 1000 nodes) even with only a single visualization server. Also, our results show that requiring only instrumentation code incurs a very minimal overhead.

In our first experiment we measured the latency for RMI function calls which push collected metrics from the visualization client to the visualization server. We varied the number of Ceph clients (and thus visualization clients) and Figure 12 shows the results and standard deviation of each run with outliers removed. Even with a system size greater than 1,000 nodes RMI call latency is modest. The

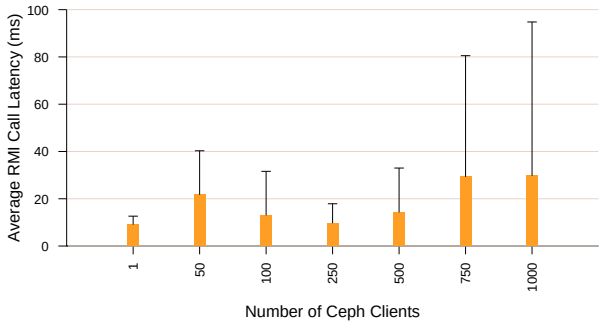


Figure 12: Average latency for RMI function calls to push collected metrics from the visualization client to the server as the number of Ceph clients increases.

average call latency with 1,000 nodes is only three times that of latency with one node. This indicates that even in very large systems the cluster of visualization servers may be kept small.

The second experiment analyzes the number of messages received by the visualization server as the number of nodes in the system increases. Figure 13 shows the total number of messages received by the visualization server once all clients have finished writing 500MBs using 1MB size writes. The number of messages quickly increases as the number of nodes increases to over 1,000. Though the aggregate number is high, there are roughly less than 130 messages received from each node. The major factor that contributes to the increase in total message is the length of the workload, as more messages will be sent for longer running workloads and adding clients increases the length of the workload.

Our final experiment looks at the performance overhead added by our profiler. We ran three workloads, each with and without our profiling infrastructure present and measured the total time for each workload to run. We used three workloads, a heavy-metadata only workload, a light-I/O and light-metadata workload, and a heavy-I/O only workload. Table 3 shows our results. We see that the visualization client profiling each node adds a near negligible overhead to each workload. This comes directly from requiring only instrumentation code to be added to the storage system, eliminating any expensive bottlenecks on critical paths.

8 Future Work

The opportunity exists in a number of areas for future work. As scalability is a necessity when performing online analysis of distributed systems, the volume and granularity of metrics sent to the visualization server must cor-

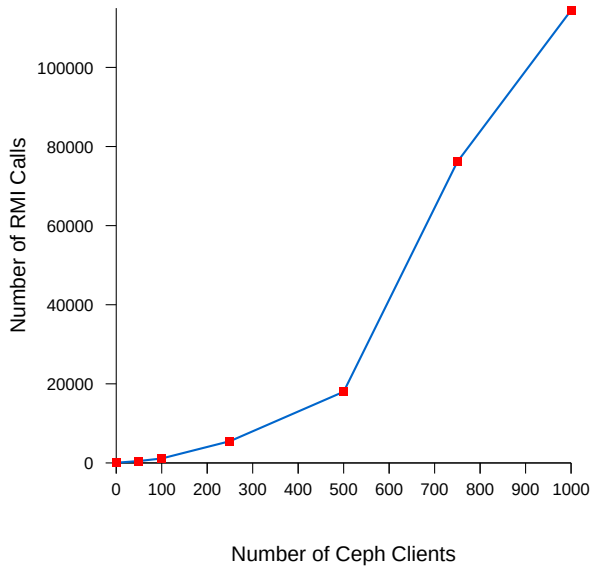


Figure 13: The total number of RMI calls received by the visualization server as the number of Ceph clients increases. The total number of calls accounts for the time required for each client to write a 500MB file.

Workload	W/o Client	W/ Client
Metadata	165	167
I/O w/ metadata	157	158
I/O	146	146

Table 3: The total time (in seconds) required to run three different workloads with and without the visualization client profiling the system. Profiling adds a near negligible overhead since only instrumentation data is added to the storage system.

respond to the level of view abstraction chosen by the user. There are a number of improvements we can make to our feedback-loop approach to aggregating data. One possible area is the granularity of data that the visualization server aggregates. For example, if the user zooms the view into a small region, the granularity of data sent from that region should become more fine-grained. This would complement our current approach of only sending data from nodes which currently appear in the view.

Another area of potential research is the integration of automated performance anomaly detection using statistical analysis. This approach may compliment our comprehensive view of system analysis by notifying the visualization application when performance outliers are detected in the system. Such notifications will make the user aware of issues outside of the current view.

Finally, further advances to the visualization application will support additional metrics relevant to debugging distributed file systems. Our current prototype has more OSD-specific metrics than any other device. Future enhancements are needed MDS performance metrics, specifically in the areas of journal operations and load balancing. Also, metrics describing activity in each Ceph client's local cache is needed to understand how client cache performance effects workload throughput.

9 Conclusions

We presented a new approach to distributed storage system profiling that focuses on offering an intuitive view of system performance in a scalable fashion. We successfully identified a number of performance issues in the Ceph peta-scale file system, including those which result from inter-node relationships. We also identified performance degradation that resulted from seemingly unrelated system activities. The ability of our system to identify these issues shows promise for our prototype visualization application.

The low-overhead associated with our profiling technique supports the idea that both low and high-level performance metrics can be collected without reducing performance. Our performance analysis has also shown that the communication overhead associated with aggregating performance metrics does not increase significantly as the number of clients increases.

In conclusion, we believe our work has motivated and demonstrated a need to achieve a comprehensive view of the storage system if complex performance debugging is to be achieved. We've asserted our position that as storage become larger and more complex, a more extensive understanding of the system as a whole is required. We hope our work serves motivates others toward this goal.

References

[1] M. K. Aguilera, J. C. Mogul, J. L. Wiener, P. Reynolds, and A. Muthitacharoen. Performance debugging for distributed systems of black boxes. In *SOSP '03: Proceedings of the nineteenth ACM symposium on Operating systems principles*, New York, NY, USA, October 2003.

[2] A. Aranya, C. P. Wright, and E. Zadok. Tracefs: A File System to Trace Them All. In *Proceedings of the Third USENIX Conference on File and Storage Technologies (FAST 2004)*, pages 129–143, San Francisco, CA, March/April 2004. USENIX Association.

[3] P. T. Barham, A. Donnelly, R. Isaacs, and R. Mortier. Using magpie for request extraction and workload modelling. In *Proceedings of the 6th Symposium on Operating Systems Design and Implementation (OSDI)*, San Francisco, CA, Dec. 2004.

[4] R. Bosch, C. Stolte, D. Tang, J. Gerth, M. Rosenblum, and P. Hanrahan. Rivet: a flexible environment for computer systems visualization. In *SIGGRAPH Computer Graphics 2000*, volume 34. ACM, 2000.

[5] R. Bryant, R. Forester, and J. Hawkes. Filesystem performance and scalability in linux 2.4.17. In *Proceedings of the FREENIX Track: 2002 USENIX Annual Technical Conference*, Berkeley, CA, USA, June 2002.

[6] M. Chen, A. Accardi, E. Kcman, J. Lloyd, D. Patterson, A.Fox, and E. Brewer. Path-based failure and evolution management. In *Proceedings of the First Symposium on Networked Systems Design and Implementation (NSDI)*, San Francisco, CA, 2004.

[7] M. Chen, E. Kiciman, E. Fratkin, A. Fox, and E. Brewer. Pinpoint: Problem determination in large, dynamic, internet services. In *Proceedings of the International Conference on Dependable Systems and Networks*, 2002.

[8] I. Cohen, J. S. Chase, M. Goldszmidt, T. Kelly, and J. Symons. Correlating instrumentation data to system states: A building block for automated diagnosis and control. In *Proceedings of the 6th Symposium on Operating Systems Design and Implementation (OSDI)*, San Francisco, CA, Dec. 2004.

[9] I. Cohen, S. Zhang, M. Goldszmidt, J. Symons, T. Kelly, and A. Fox. Capturing, indexing, clustering, and retrieving system history. In *SOSP '05: Proceedings of the twentieth ACM symposium on Operating systems principles*, pages 105–118, New York, NY, USA, 2005. ACM Press.

[10] D. Ellard and M. Seltzer. New nfs tracing tools and techniques for system analysis. In *LISA 03: Proceedings of the 17th Annual USENIX Conference on Large Installation Systems Administration*. USENIX Association, 2003.

[11] R. Fonseca, G. Porter, R. H. Katz, S. Shenker, and I. Stoica. X-trace: A pervasive network tracing framework. In *NSDI '07: Proceedings of the 4th USENIX Symposium on Networked Systems Design and Implementation*. USENIX Association, 2007.

[12] D. Geels, G. Altekar, P. Maniatis, T. Roscoe, and I. Stoica. Friday: Global comprehension for distributed replay. In *NSDI '07: Proceedings of the 4th USENIX Symposium on Networked Systems Design and Implementation*. USENIX Association, 2007.

[13] S. Graham, P. Kessler, and M. McKusick. Gprof: A call graph execution profiler. *Proceeedings of the SIGPLAN '82 Symposium on Compiler Construction*, June 1982.

[14] D. Gunter, B. Tierney, B. Crowley, M. Holding, and J. Lee. Netlogger: A toolkit for distributed system performance analysis. In *MASCOTS '00: Proceedings of the 8th International Symposium on Modeling, Analysis and Simulation of Computer and Telecommunication Systems*, page 267, Washington, DC, USA, 2000. IEEE Computer Society.

[15] D. Hughes. Using visualization in system and network administration. In *LISA '96: Proceedings of the 10th USENIX conference on System administration*, pages 59–66, Berkeley, CA, USA, 1996. USENIX Association.

[16] N. Joukov, A. Traeger, R. Iyer, C. P. Wright, and E. Zadok. Operating system profiling via latency analysis. In *Proceedings of the 7th Symposium on Operating Systems Design and Implementation (OSDI)*, Seattle, WA, Nov. 2006.

[17] M. L. Massie, B. N. Chun, and D. E. Culler. The ganglia distributed monitoring system: design, implementation, and experience. *Parallel Computing*, 30(5-6):817–840, 2004.

[18] M. P. Mesnier, M. Wachs, R. R. Sambasivan, A. X. Zheng, and G. R. Ganger. Modeling the relative fitness of storage. *SIGMETRICS Perform. Eval. Rev.*, 2007.

[19] B. P. Miller, M. D. Callaghan, J. M. Cargille, J. K. Hollingsworth, R. B. Irvin, K. L. Karavanic, K. Kunchithapadam, and T. Newhall. The paradyn parallel performance measurement tool. *Computer*, 28(11):37–46, 1995.

[20] A. V. Mirgorodskiy, N. Maruyama, and B. P. Miller. Scalable systems software—problem diagnosis in large-scale computing environments. In *SC '06: Proceedings of the 2006 ACM/IEEE conference on Supercomputing*, page 88, New York, NY, USA, 2006. ACM Press.

[21] P. Reynolds, C. Killian, J. L. Wiener, J. C. Mogul, M. A. Shah, and A. Vahdat. Pip: Detecting the unexpected in distributed systems. In *In Proceedings of the Third Symposium on Networked Systems Design and Implementation (NSDI)*, San Jose, CA, 2006.

[22] K. Shen, M. Zhong, and C. Li. I/o system performance debugging using model-driven anomally detection. In *FAST '05: Proceedings of the 4th USENIX Conference on File and Storage Technologies*. USENIX Association, 2005.

[23] S. S. Shende and A. D. Malony. The tau parallel performance system. *Int. J. High Perform. Comput. Appl.*, 20(2), 2006.

[24] E. Thereska, B. Salmon, J. Strunk, M. Wachs, M. Abd-El-Malek, J. Lopez, and G. R. Ganger. Stardust: tracking activity in a distributed storage system. *SIGMETRICS Perform. Eval. Rev.*, 34(1), 2006.

[25] 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 the 7th Symposium on Operating Systems Design and Implementation (OSDI)*, Seattle, WA, Nov. 2006.

[26] S. A. Weil, S. A. Brandt, E. L. Miller, and C. Maltzahn. CRUSH: Controlled, scalable, decentralized placement of replicated data. In *Proceedings of the 2006 ACM/IEEE Conference on Supercomputing (SC '06)*, Tampa, FL, Nov. 2006. ACM.

[27] 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 (SC '04)*, Pittsburgh, PA, Nov. 2004. ACM.

[28] S. Zhou, H. D. Costa, and A. J. Smith. A file system tracing package for berkeley unix. Technical Report UCB/CSD-85-235, EECS Department, University of California, Berkeley, 1985.