Using Comprehensive Analysis for Performance Debugging in Distributed Storage Systems

Andrew W. 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. 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 visualize the storage system by displaying nodes and intuitively animating their metrics and behavior allowing easy analysis of complex problems.

1. Introduction

The complex nature of distributed storage increases debugging complexity. Distributed storage performance problems 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, 9, 11]. Understanding inter-node problems has become an interesting challenge explored more recently [3, 6, 8, 11, 12, 16]. The key problem facing current approaches is an inability to achieve comprehensive understanding of the *entire* storage system. More specifically, a complete view of the storage system includes 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. We assert that in order to fully understand and debug distributed storage, both node-specific and inter-node events must be analyzed.

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 system size 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 and analyzing both micro and macro behavior, and offering 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 represent system activity and behavior in real time.

We have implemented our profiler in the Ceph petascale, distributed file system [17] 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 characterize metadata operations via pie charts. By viewing multiple nodes at once, users can easily understand complex inter-node relationships. Evaluations of our profiler indicate a very limited overhead, with scalability in storage systems over 1,000 nodes. Using our visualization we uncovered several important performance issues in Ceph.

2. Related Work

Profiling and benchmarking storage systems presents many unexplored challenges. While significant inroads have been made in profiling local file systems which reside on a single machine [2, 9, 11], more current work is exploring general distributed systems. These areas include profiling black-box systems [1, 5, 11], fine-grained profiling [11], end-to-end request tracing [3, 16], and model checking [6, 7, 13, 14]. Each of these tools is effective for locating and determining bugs and performance problems in distributed systems. We argue for a simpler approach to identifying performance problems in distributed storage. We believe visualizing storage system nodes, events, and operations allows users to easily identify problems. This is particularly true for inter-node problems, which are more difficult to identify with previous solutions. We believe a simple debugging strategy, such as system visualization, will become even more important as systems become larger and more complicated.

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. 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. A number of systems use call-graphs to build the path taken by requests through the system [1, 3, 14, 16]. This approach is useful for identifying inefficient software components and slow nodes in a network. Unfortunately, it only provides information for a single path in the system and understanding the interaction of many correlated paths can be difficult. Previous systems have also relied on visualization to aide in debugging distributed systems [4, 10, 12, 15]. A key difference between these solutions and ours is these solutions are aimed at more general distributed systems. Our work is aimed at analyzing both low- and high-level metrics across a large number of storage nodes.

3. 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 the 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.

3.1. 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.

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 occurred during the previous poll interval. This aggregation is then time 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.

3.2. 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. In order to visualize and understand the many cause-and-effect relationships of events in a distributed storage system the visualization servers try to organize and forward system metrics as a cohesive, time-ordered stream.

However, ordering a continual stream of events by time stamps poses an inherent trade-off between timeliness and global event ordering. Events are batched 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 window are passed to the visualization client in the correct order. Long 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.

With a clustered visualization server 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` and `1600` to a specific server where all events for that period can be correctly ordered. Authority for a time interval may be calculated via a simple hash, mapping intervals to servers.

An important scalability issue is the amount of data that is forwarded to the visualization server. If each client collects and forwards a large amount of data, 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 to values of interest which correspond to specifications provided by the visualization application. For example, when the visualization application is only analyzing storage device performance, the amount of data collected from storage system clients can be reduced.

We have implemented the visualization server as a Java RMI server. 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.

3.3. Visualization Application

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. Users can view any subset of nodes or metrics in order to improve comprehension of large-scale systems.

4. Implementation Details

We have implemented our performance debugging system in the Ceph petascale, distributed file system [17]. 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. Prototype status indicates 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.

In our visualization application users are able to toggle the metrics being displayed and which nodes are in focus. Figure 1 shows a screenshot of the visualization application with numeric labels and descriptions added. Clients are represented on top, with MDSs following, and OSDs on the bottom. 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 1 allows users to toggle the nodes and metrics in view.

5. Visualizing the Ceph File System

We conducted a study on Ceph to evaluate the ability of our profiling and visualization techniques to aid in debug-

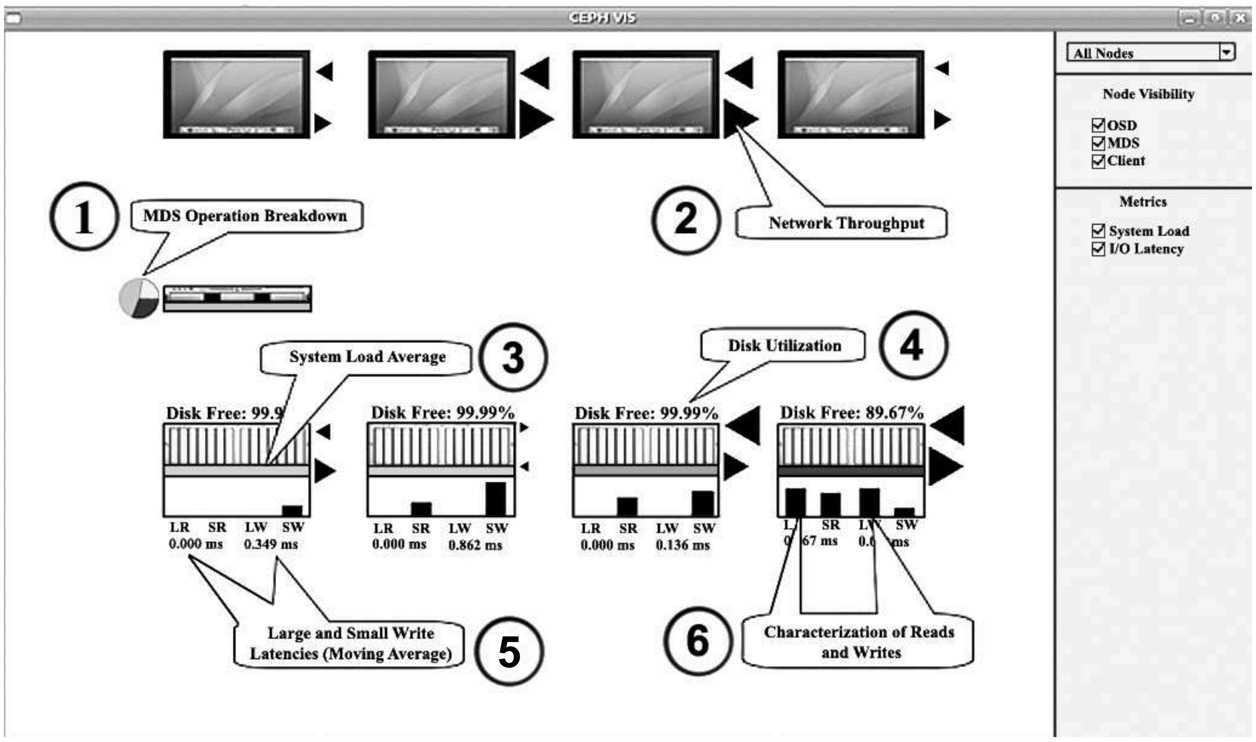
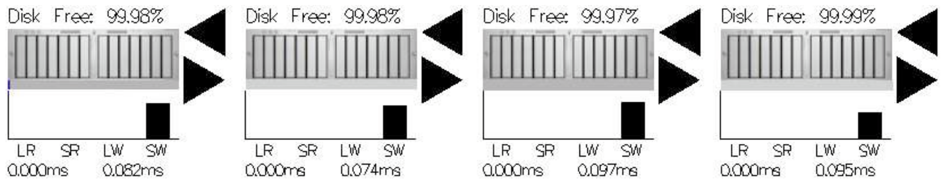


Figure 1. A labeled screenshot of the visualization application.

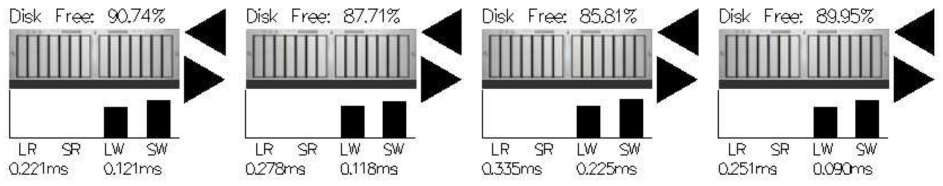
ging performance. Our experiments focus on the visualization application's ability to reveal inter-node performance problems. For each performance issue revealed, we validate our observation through additional experiments. We used a 25 node cluster with 4 OSDs and a varying number of MDSs and clients. A single visualization server was run on a separate 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.

Our 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 2(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 2(b) shows a much higher load on each OSD, and latency for small write operations (the journal flushes) significantly increased. 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 3. 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 surprising 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.

Ceph employs an advanced metadata load balancing scheme called Dynamic Subtree Partitioning [18]. 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 implementation 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 4 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 Fig-



(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 2. Increased load and latency on OSDs show I/O and metadata workloads interfering.

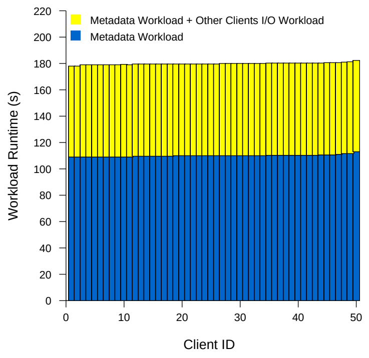


Figure 3. The time to run a metadata workload with and without other clients performing an I/O workload.

Figure 4. Uneven load across three MDSs during a flash crowd.

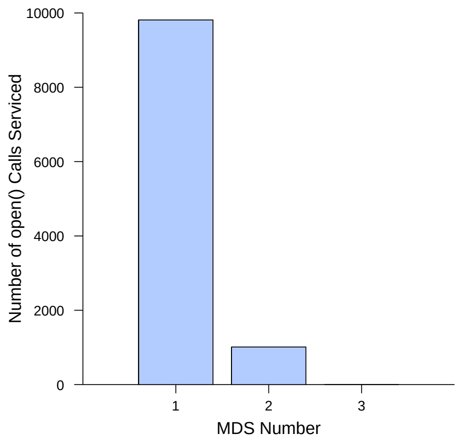


Figure 5. The number of opens handled by each MDS during a flash crowd.

ure 5. 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 either an infrequent exchanges of load metrics or slow migration of the namespace.

6. Performance Evaluation

We evaluated the overhead and scalability of our profiler in Ceph. Experiments were conducted using the same 25 node cluster, though 12 OSDs were used instead of 4. 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 6 shows the results and standard deviations. The 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. To explore the performance overhead added by our profiler, we ran three workloads, each with and without our profiling infrastructure and measured the total run time for each workload. We used a heavy-metadata only workload, a light-I/O and light-metadata workload, and a heavy-I/O only workload. Table 1 shows our results. We see that the visualization client profiling each node adds a near negligible overhead to each workload.

7. Future Work and Conclusions

The opportunity exists in a number of areas for future work. One possible area is the granularity of data that

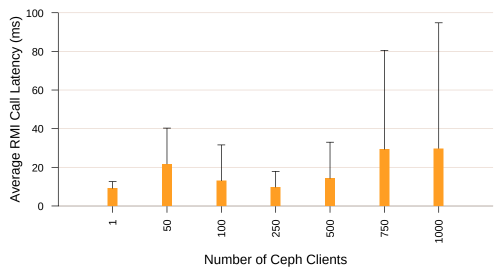


Figure 6. Average latency for RMI calls as clients increase.

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

Table 1. Time (in seconds) to run three workloads with and without profiling the system.

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.

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 performance issues in the Ceph petascale file system. 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. In conclusion, we believe our work has motivated and demonstrated a need to achieve a simple comprehensive view of the storage system if complex performance debugging is to be achieved. 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. *19th SOSP*, Oct. 2003.

[2] A. Aranya, C. P. Wright, and E. Zadok. Tracefs: A File System to Trace Them All. *3rd FAST 2004*, Mar. 2004.

[3] P. T. Barham, A. Donnelly, R. Isaacs, and R. Mortier. Using magpie for request extraction and workload modelling. *6th OSDI*, Dec. 2004.

[4] R. Bosch, C. Stolte, D. Tang, J. Gerth, M. Rosenblum, and P. Hanrahan. Rivet: a flexible environment for computer systems visualization. *27th SIGGRAPH*, July 2000.

[5] M. Chen, A. Accardi, E. Kcman, J. Lloyd, D. Patterson, A.Fox, and E. Brewer. Path-based failure and evolution management. *1st NSDI*, Mar. 2004.

[6] 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. *6th OSDI*, Dec. 2004.

[7] I. Cohen, S. Zhang, M. Goldszmidt, J. Symons, T. Kelly, and A. Fox. Capturing, indexing, clustering, and retrieving system history. *20th SOSP*, Oct. 2005.

[8] R. Fonseca, G. Porter, R. H. Katz, S. Shenker, and I. Stoica. X-trace: A pervasive network tracing framework. *4th NSDI*, Apr. 2007.

[9] S. Graham, P. Kessler, and M. McKusick. Gprof: A call graph execution profiler. *SIGPLAN*, June 1982.

[10] N. Joukov, A. Traeger, R. Iyer, C. P. Wright, and E. Zadok. Operating system profiling via latency analysis. *7th OSDI*, Nov. 2006.

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

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

[13] A. V. Mirgorodskiy, N. Maruyama, and B. P. Miller. Problem diagnosis in large-scale computing environments. *SC '06*, Nov. 2006.

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

[15] 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*, June 2006.

[16] S. A. Weil, S. A. Brandt, E. L. Miller, D. D. E. Long, and C. Maltzahn. Ceph: A scalable, high-performance distributed file system. *7th OSDI*, Nov. 2006.

[17] S. A. Weil, S. A. Brandt, E. L. Miller, and C. Maltzahn. CRUSH: Controlled, scalable, decentralized placement of replicated data. *SC '06*, Nov. 2006.

[18] S. A. Weil, K. T. Pollack, S. A. Brandt, and E. L. Miller. Dynamic metadata management for petabyte-scale file systems. *SC '04*, Nov. 2004.