Diagnosing Performance Issues in Cray ClusterStor Systems

Comprehensive Performance Analysis with Cray® View for ClusterStor™

Patricia Langer
Cray, Inc.
Bloomington, MN USA
planger@cray.com

I. INTRODUCTION

The size, scale and number of subsystems used in current HPC deployments make it difficult for administrators to monitor application performance and determine root cause when system performance issues occur. Many factors can contribute to poor application performance and system problems, including component failures, resource oversubscription or poorly-written applications. Isolating the root cause of a performance issue can be challenging as data needed to analyze the problem must be mined from multiple sources across different system components and may require privileged user access to retrieve. Additionally, system data must be obtained in a timely manner or critical information may be lost.

Due to the complexity of most storage system deployments, it can take weeks or months to isolate a performance problem, create a reproducible test case and identify the root cause. Performance analysis requires dedicated resources from both the customer site and Cray to gather information and conduct evaluations. Time spent isolating and debugging complex problems results in reduced human and system productivity, and increased TCO, with a direct, negative impact on corporate budgets.

To address significant challenges in storage performance analysis, Cray has introduced View for ClusterStor[1], an innovative tool that provides detailed visibility into resource utilization by system applications, enabling more efficient debugging, targeted optimizations and faster time to resolution. View for ClusterStor enables system administrators to:

- *Identify poorly performing jobs by using real-time and historical data*
- *Isolate performance problems by correlating performance-impacting systemic events*
- *Observe system trends by analyzing data in graphical and tabular formats*

View for ClusterStor's integration with storage components enables the collection of performance and jobstats metrics, system logs, and system events from the ClusterStor system, metrics from the high-speed InfiniBand interconnect, and job event details from the workload manager. This information is aggregated and made available to administrators via the View for ClusterStor GUI.

This paper demonstrates the innovations in View for ClusterStor, as compared to traditional problem debugging and root cause isolation methods, applied to a use case in the form of a customer-reported problem. The use case first considers how the customer and Cray used traditional tools and debug methods to identify the root cause of a MetaData Server (MDS) performance degradation. The use case then examines how the same problem could be identified, much more efficiently, using View for ClusterStor. Contrasting traditional debug methods with the performance analytics presented by View for ClusterStor makes a compelling case that customer resources are maximized, and significant time saved with the use of View for ClusterStor.

II. DESCRIPTION OF CUSTOMER USE CASE

During a standard application run, the customer identifies significant performance degradation of their ClusterStor MDS as a result of 100% utilization. Although system applications are progressing, overall system performance is significantly impacted, which negatively affects other filesystem users.

The customer determines that performance degradation occurs when a specific application is run on the attached Cray Cluster system (CCS). No degradation is observed when the same application is run on the attached Cray XC system. OpenSHMEM[2] is used on the Cray Cluster system, while Cray SHMEM[3] is used on the Cray XC. The customer presumes that the MDS degradation is related to SHMEM and reports it to Cray, indicating that the application in question is using a simple file open pattern via SHMEM.

III. WALKTHROUGH OF TRADITIONAL ROOT CAUSE ANALYSIS

A. *Identification of the Problem (October 2017)*

When the customer reports the system performance issue, Cray Support is engaged, and a case is opened. Given the asynchronous nature of the problem, the customer and Cray Support coordinate the collection of logs and general support information for review and to start root cause analysis. Information is shared through the case. Several people are involved with the case—Cray Support identifies what information is needed, and the customer, using root access, collects the requested information and provides it to Cray. Information sharing, however, is limited to what can be securely allowed offsite, given the customer's security policies.

While information and log collection are underway, the customer successfully creates a reproducible test case and shares it with Cray. This development proves to be a critical step in the debug effort.

The customer and the on-site Cray team run the reproducer, redirecting IO in different ways to help determine behavioral differences based on how the application is invoked.

- `srun --output=/proj/users/$USER/shmem-%t.out –error=all -n 1600 .slow-snx-shmem`

  This command invokes the reproducer piping output to a non-Lustre filesystem (/proj) using SHMEM

- `srun –output=/c/$USER/shmem-%t.out –error=all -n 1600 .slow-snx-shmem`

  This command invokes the reproducer piping the output to the Lustre filesystem (/c) using SHMEM

- `srun --output=/c/$USER/mpi-%t.out –error=all -n 1600 .slow-snx-mpi`

  This command invokes the reproducer piping the output to the Lustre filesystem (/c) using MPI

During the reproducer runs, the customer monitors the MDS on /c (Lustre filesystem mount point) and notices that the % CPU use increases when the "srun-snx-shmem" test case is run with output redirected to /c. Based on this limited testing, the customer and Cray team determine that redirecting IO to a non-Lustre filesystem (/proj) appears to have no adverse effects.

B. *Engaging Cray Engineering (November 2017)*

At this point, a Cray Lustre engineer is engaged to analyze the collected information and logs, results of the reproducer runs, and additional data known about the case.

The Cray Lustre engineer inquiries about test runs in which IO was redirected to different locations. Specifically, for each test run, the engineer asks how the application was invoked and the location to which the application output was written. These inquiries and answers are entered into the case.

C. *Holiday and Vacation Period (November – December 2017)*

No updates occurred in the case from mid-November through December due to the holiday break and vacations.

D. *Debug, Analysis and Identification of Root Cause (January – February 2018)*

Responses to the Cray engineer's outstanding questions are entered into the case. The responses confirm the following:

- /c is the Lustre filesystem mount point
- Writing to something other than the Lustre filesystem does not exhibit adverse behavior

The Cray engineer requests that the customer run the following commands to gather additional MDS resource utilization data. These commands must be run when the reproducer is operating and the MDS is at 100% utilization.

- `ps auxww | awk '/\[(mdt|ll_ost)/ {gsub(/ll_ost/,"ost"); print $8" "$11}' | sort | uniq -c -w6`
- `lctl get_param mds.MDS.mdt.stats | grep "req_"`
- `lctl get_param -n ldlm.services.ldlm_cbd.stats | grep "ldlm_bl”`

The ps auxww command lists the MDT threads and what those threads are doing. The results will help determine if the threads are running, if they are waiting on disk or if the threads are sleeping, waiting on locks. Based on the results, one can determine if the MDS is busy with CPU activity, disk activity or waiting on protocols within the MDS system.

The lctl[4] (low level Lustre filesystem configuration utility) command, using the get_param function with the mds.MDS.mdt.stats parameter, gathers statistics from the MDS. These results will help determine if MDS requests are waiting and for how long. The results will show if the MDS has a backlog of requests to process and if the MDS is keeping up with the backlog. This information is one way to determine if the MDS is overloaded with requests that it is unable to process.

The lctl command, using the get_param function with the ldlm.services.ldlm_cbd.stats parameter, collects MDS lock information, including specific data that indicates lock contention, and if any requests are blocked on locks.

The results of the command runs are available within a week. The Cray Lustre engineer analyzes the information and concludes the following:

- ps command results indicate there is a lot of CPU activity and not much disk activity
- lctl mds command results confirm there is a lot of CPU activity
- lctl ldlm command results indicate there are no lock contentions

These results indicate to the Cray Lustre engineer that the root cause is neither MDS hardware nor software, and more information is needed to determine what is consuming the MDS resources and causing the degradation.

The Cray Lustre engineer reviews the original problem report. As the test cases use different stdout pathways to the Lustre filesystem, the Cray Lustre engineer focuses on the differences between OpenSHMEM[2] and Cray SHMEM[3].

The Cray Lustre engineer summarizes the stdout redirection data, shown in Table I.

TABLE I. SUMMARY OF STDOUT REDIRECTION TESTS

Command	Performance
MPI output directed to Lustre (/c)	Fast
SHMEM to Lustre (/c)	**Slow**
MPI to non-Lustre (/proj)	Fast
SHMEM to non-Lustre (/proj)	Fast
SHMEM to Lustre (/c), but with logging hidden (MXM_LOG_FILE)	Fast

With the information gathered to date, the Cray engineer confirms there is a reasonable workaround to improve system performance, but the root cause has not yet been determined. The workaround is to write the application output to a local /tmp filesystem and then move the output to the Lustre filesystem.

- srun --output=/tmp/shmem-%t.out --error=all -n 1600 ./slow-snx shmem; mv /tmp/shmem-%t.out /c/$USER/shmem-%t.out

The Cray Lustre engineer surmises, based on the command results and other evidence, that SHMEM is using a poor write pattern when the stdout log file is written to the Lustre filesystem. To test this theory, the Cray Lustre engineer asks the customer to re-run the lctl commands and add one additional command (md_stats).

- ps auxww | awk '/\[(mdt|ll_ost)/ {gsub(/ll_ost/,"ost"); print $8" "$11}' | sort | uniq -c -w6
- lctl get_param mds.MDS.mdt.stats | grep "req_"
- lctl get_param -n ldlm.services.ldlm_cbd.stats | grep "ldlm_bl”
- **lctl get_param mdt.*.md_stats**

The lctl command, using the get_param function with the mdt.*.md_stats parameter, collects metadata operation statistics from all MDTs. This supplemental information provides the confirmatory clue to the Cray engineer regarding the root cause of the performance degradation. The md_stats command results, correlated and summarized in Table II, show that although several metadata operations could cause a problem, in this case the clear offender is sync.

TABLE II. MD STATS COMMAND RESULTS

Operation	MXM Workaround	No Workaround	Change
open	4135	9883	239%
close	4078	9575	235%
unlink	961	6024	627%
mkdir	4	2000	50000%
rmdir	3	2000	66667%
getaddr	31116	131598	423%
statfs	201	2009	1000%
sync	**0**	**830725**	**infinite**

E. *Root Cause (February 2018)*

The root cause is identified as the method by which the application is writing to the stdout log file, specifically, the high number of syncs to disk when writing stdout to the Lustre filesystem. For every write to stdout, a sync to disk occurs, resulting in 830,000 syncs over a 3-minute period or 4600 syncs per second.

Additional analysis is required to determine how to resolve the problem. Based on the collected data, the Cray engineer suspects that the problem is in SLURM’s output handling of srun. Until a resolution is implemented, the customer can use the identified workaround. The issue is handed back to Cray Support for analysis and to determine next steps.

F. *Summary of Use Case Timeline*

As demonstrated by this use case, complex problems take time to analyze and, typically, require engagement of a filesystem expert. This situation required an experienced Lustre engineer to help debug and diagnose the root cause. In cases involving multiple parties, an added complexity, the timeline from first identification of a problem to root cause determination can take several months. In this use case, the customer experienced performance degradation for more than four months before the root problem was isolated and a workaround identified. Table III summarizes the timeline of the debugging activity.

TABLE III. DEBUG TIMELINE

Timeframe	Activity
October 2017	Customer identifies a performance issue with the MDS
	Offending application is identified
	Problem reported to be related to SHMEM
November 2017	Cray Lustre engineer engaged
	Metrics requested from the MDS and OSTs
December 2017	Holiday and vacation period – no activity
January 2018	Requested command output returned to Cray for analysis
February 2018	Workaround is identified
	Data is collected which isolates the problem
	Root cause identified

IV. WALKTHROUGH OF VIEW FOR CLUSTERSTOR ROOT CAUSE ANALYSIS

View for ClusterStor gathers and persists available metrics, logs and events from the ClusterStor storage system, metrics from the high-speed InfiniBand network, and job event details. The collected Lustre performance data includes performance information for OSTs, MDTs and jobstats. Raw Lustre performance metrics are derived into usable (delta) metrics, which are also persisted. These metrics can then be graphed in tools such as Grafana. Lustre jobstats metrics are correlated with job event information, which includes the jobid, apid, start time, stop time, apname and userid. This information is also persisted and correlated with other collected data.

As View for ClusterStor collects, derives and persists Lustre performance data, additional summary metrics are calculated and persisted on a per-job basis. These summary metrics provide the average IO size per job and the total metadata operations per job. Data is gathered every 5 to 30 seconds and persisted for up to 4 weeks or more.

System logs and events are persisted into Elasticsearch and served up through a UI for searching and refined filtering. Data is retained using a configurable retention policy, allowing administrators to view near-real time and historical information.

View for ClusterStor's GUI provides administrators with a visual representation of the collected data. The GUI home page provides a high-level overview of ClusterStor system performance. Administrators can selectively view details about running jobs, via the job summary table and can drill down to see details of OST and MDT usage per a selected job. These views can be customized to show data from the last 15 minutes, last hour, n days, weeks or months. View for ClusterStor utilizes features within Grafana to display metrics in pre-defined Grafana panels. In addition, administrators may create and save their own Grafana panels to view customized arrays of data rather than the default panels.

Calculated job summary information is displayed in the job summary table (shown Figure 3), providing a visual method to identify jobs which are performing poorly, based on small IO patterns or abnormally high metadata operations.

View for ClusterStor provides an alert and notification capability, delivering pre-defined alarm definitions. These alarms are site-customizable and extensible. When an alarm hits a defined threshold, an email notification is sent to the configured email alias, providing early notification of possible problems.

Note that the data presented in this section is based on a reproduction of the customer test case. Actual customer data is not used or presented.

A. *Identification of the Problem (Day 1)*

Upon notification that there is an MDS performance issue, the administrator logs into the View for ClusterStor GUI and starts analysis. The home page shows a high-level performance summary for each ClusterStor system that is monitored by View for ClusterStor (Figure 1).

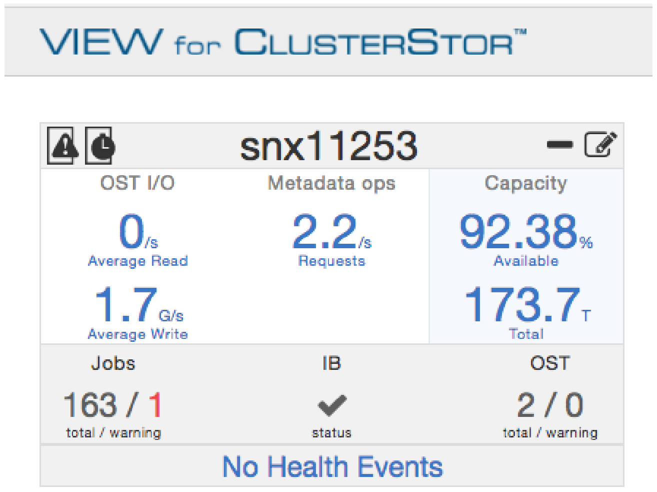


Figure 1. Home Page

B. *Debug, Analysis and Identification of Root Cause (Day 1)*

Based on the performance summary, the administrator determines that 1 of 163 running jobs has been flagged as (possibly) performing poorly.

The Administrator navigates to the system overview page for further details on the overall IO and metadata performance of the system (Figure 2) and observes job write performance degradation starting around 14:45.

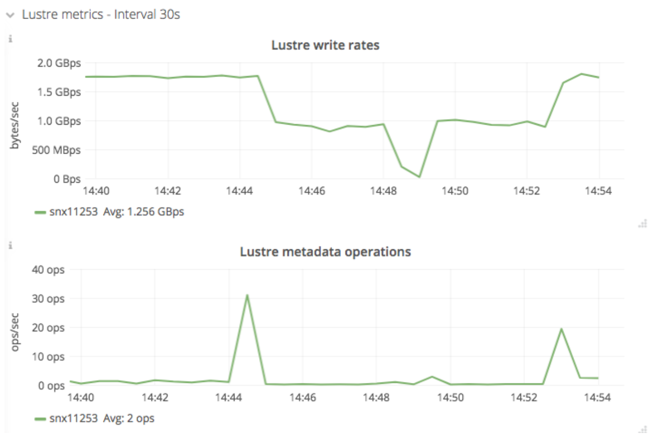
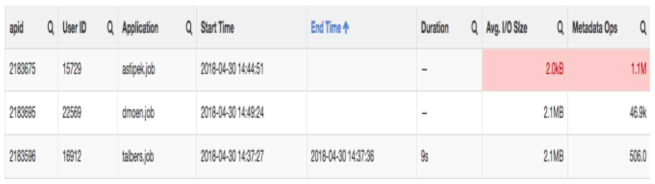


Figure 2. System Overview

The administrator returns to the home page and selects the Jobs link for the ClusterStor system, navigating to the table which summarizes all jobs running on the selected system (Figure 3).



apid	User ID	Application	Start Time	End Time ↑	Duration	Avg. I/O Size	Metadata Ops
2183675	15729	astipek.job	2018-04-30 14:44:51		–	2.0kB	1.1M
2183695	22569	dmoen.job	2018-04-30 14:49:24		–	2.1MB	46.9k
2183596	16912	talbers.job	2018-04-30 14:37:27	2018-04-30 14:37:36	9s	2.1MB	506.0

Figure 3. Job Summary Table

The table shows all jobs active for the last 15 minutes. One job is flagged as a possible poor performer (apid 2183675). The administrator notes the userid of the user who launched the job, when the job started and ended, the application name and the IO and metadata summary metrics. Based on this data, the administrator determines that metadata operations in the flagged job are high as compared to other running jobs. The administrator clicks on the apid link for job 2183675 to view additional job metadata details (Figure 4).

Error! Reference source not found.

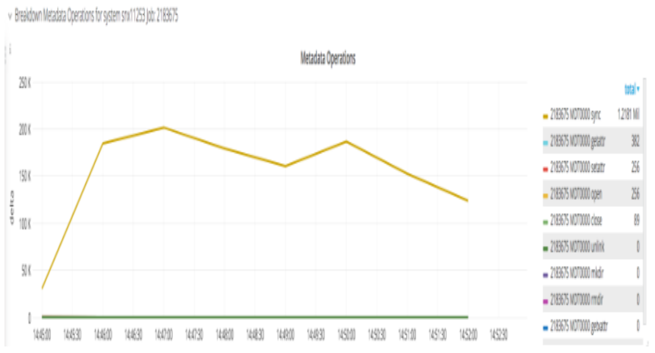


Figure 4. Job Details

From this screen, the administrator confirms that job 2183547 is writing small IO and metadata operations are high during the job run. The administrator observes that for this job, sync operations number in the millions; sync is identified as the root cause.

C. *Root Cause Summary*

Within 5 to 10 minutes of first analyzing data in View for ClusterStor, the administrator identified a poorly performing job on the ClusterStor system, located the specific job that appeared to be the culprit, and viewed IO and metadata operation details for the job that confirmed the problem was caused by unusually high sync metadata operations being executed by the application. No reproducer was required for the analysis and the administrator isolated the root cause without engaging a filesystem expert. With root cause determined, the administrator can contact the user and inform them that their application is causing performance degradation of the ClusterStor system. The user will need to review their application to determine the source of the large number of syncs and how to resolve the problem.

V. CUSTOMIZING VIEW FOR CLUSTERSTOR GRAFANA PANELS

The administrator may create a customized Grafana panel to show details of multiple jobs. As shown in Figure 5, the customized panels can display data for both a job which is performing well and a job which is performing poorly (due, in this case, to a high number of syncs). As observed in these graphs, a job which is performing poorly can, because of resource contention, adversely impact a job which is performing optimally.

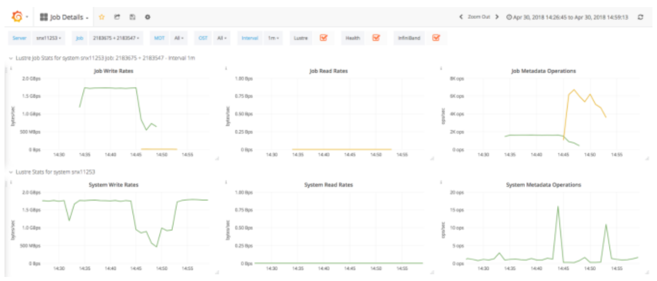


Figure 5. Customized Grafana Panels Showing Details for Multiple Jobs

VI. SUMMARY/CONCLUSION

Providing the tools to monitor and collect relevant metrics and information in a timely manner is critical to problem resolution. Putting this information into the hands of onsite administrators is very powerful way to reduce lost productivity and limit the resources, measured in time and expense, needed to determine the root cause of a performance problem.

In the case of a poorly-written application, traditional methods to debug and isolate root cause may require the use of several resources, working over several months, to bring a case to resolution. This extended timeframe reflects days, and possibly weeks, needed to gather information and collect logs, transfer the data and wait for analysis, conduct next steps and, finally, perform confirmatory testing.

View for ClusterStor collects the necessary metrics and data in a timely manner and correlates it with other information to provide a comprehensive picture of system health. As demonstrated by the use case presented in this paper, View for ClusterStor reports the state of all jobs running on the system (optimal IO and non-optimal IO) and enables an administrator to quickly identify poorly performing jobs. Because a broad set of system data is readily available and presented in easy-to-use graphs and tables, administrators have the critical information and metrics necessary to effectively isolate possible causes, greatly reducing the need to create a reproducible use case and saving significant time and effort to successfully debug performance issues.

VII. DEFINITIONS

[1] https://www.cray.com/products/storage/clusterstor/view

[2] **OpenSHMEM** is an effort to create a specification for a standardized API for parallel programming in the Partitioned Global Address Space. Along with the specification, the project is also creating a reference implementation of the API.

[3] **SHMEM** (from **Cray**'s “shared memory” library) is a family of parallel programming libraries, providing one-sided, RDMA, parallel-processing interfaces for low-latency distributed-memory supercomputers.

[4] **lctl** is the Low Level Lustre filesystem configuration utility. lctl is used to directly control Lustre via an ioctl interface, allowing various configuration, maintenance and debugging features to be accessed. lctl can be invoked in interactive mode by issuing the lctl command.