Monitoring Tools for Large Scale Systems

Ross Miller, Jason Hill, David A. Dillow, Raghul Gunasekaran, Galen Shipman, Don Maxwell

Oak Ridge Leadership Computing Facility, Oak Ridge National Laboratory
Oak Ridge, TN 37831, USA,
{rgmiller,hilljj,dillowda,gunasekaranr,gshipman,maxwellde}@ornl.gov

Abstract

Operating computing systems, file systems, and associated networks at unprecedented scale offer unique challenges for fault monitoring, performance monitoring and problem diagnosis. Conventional system monitoring tools are insufficient to process the increasingly large and diverse volume of performance and status log data produced by the world's largest systems. In addition to the large data volume, the wide variety of systems employed by the largest computing facilities present diverse information from multiple sources, further complicating analysis efforts. At leadership scale, new tool development is required to acquire, condense, correlate, and present status and performance data to systems staff for timely evaluation.

This paper details a set of system monitoring tools developed by the authors and utilized by systems staff at Oak Ridge National Laboratory's Leadership Computing Facility, which includes the Cray XT5 Jaguar. These tools include utilities to correlate I/O performance and event data with specific systems, resources, and jobs. Where possible, existing utilities are incorporated to reduce development effort and increase community participation. Future work may include additional integration among tools and implementation of fault-prediction tools.

1 Introduction

The Spider filesystem at the Oak Ridge Leadership Computing Facility is the largest single fileystem the world.[6] It provides scratch space and short-term storage for the users of the OLCF's computing resources.

Spider is built around the Lustre parallel filesystem and consists of 192 servers, 96 storage controllers, and 13,440 hard drives. It is capable of storing over 10 petabytes of data and can sustain 240 GB/s of I/O. There are over 26,800 clients spread across most of the OLCF's supercomputers. Fig. 1 shows the overall Spider architecture.

2 Challenges of Monitoring Systems at Scale

There are several challenges associated with monitoring and maintaining a filesystem the size of Spider. First, there is simply a lot of hardware to monitor. Looking for problems in one syslog file might be acceptable, but scanning 192 such files isn't. We need tools that can aggregate data from many sources and present it in an understandable fashion.

Second, Spider is a shared resource. Users access Spider from our supercomputers, our visualization systems and even remotely. One misbehaving or poorly optimized application can affect everyone else. We need tools that can quickly diagnose performance problems and trace them back to the application that is causing them.

Finally, the sheer size of the filesystem can make certain routine maintenance tasks impractical. Tools that don't take advantage of the filesystem's parallelism simply run too slowly to be useful. For example, searching the entire filesystem with 'find' can take 48 or more hours. We've had to develop new tools that do take advantage of parallelism in order to accomplish our routine maintenance tasks.

3 Tools Developed At ORNL

One of the challenges of deploying the largest filesystem in the world is the fact that you run into problems that no one has ever seen before. Of course, this means that no tools exist to help solve these problems. As such,

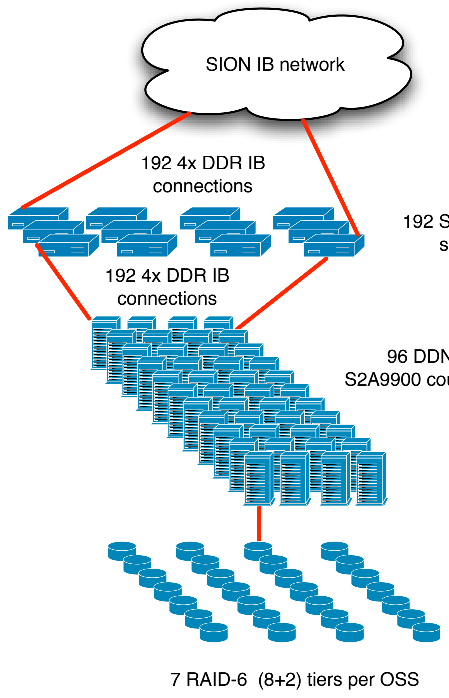
SION IB network
192 4x DDR IB connections
192 Spider OSS servers
192 4x DDR IB connections
96 DDN S2A9900 couplets
7 RAID-6 (8+2) tiers per OSS

Figure 1: Overall Spider architecture

we've had to develop a variety of tools ourselves in order to meet the specific needs we have for monitoring Spider.

3.1 MDSTrace

One of the largest challenges of Spider is tracking down pathalogical I/O patterns generated by applications. This is especially true for metadata operations that can cause perceptible pauses during interactive use. Tracking down the source of these issues using the server-side client statistic files quickly becomes a CPU hog with 26,800 clients – each file would need to be opened and read to establish a snapshot both before and after the monitoring period. In the case of Spider, even if we could afford the CPU usage, the server-side client statistics had to be disabled due to the memory overhead it required. To help overcome this blind spot, we developed MDSTrace.

MDSTrace captures a nominal sixty second slice of RPC traffic to the metadata server, and generates a report about what the server was doing, on behalf of which applications, and how long it took to perform each action. As a first step, MDSTrace initiates collection of the RPC trace log and the application node mapping in parallel. After the collection period, it then aggregates the RPC data by application, generating various statistics:

- Number of operations performed, by type
- Number of distinct nodes performing operations
- Minimum, maximum, and average processing times
- Minimum, maximum, and average queue times

These statistics are also rolled up per machine, and for the entire center. Using these statistics, it is possible to pinpoint the application(s) potentially causing performance to suffer, and to work with the project liasons to improve their I/O patterns to both improve their runtimes as well as lessen the impact on the metadata server.

MDSTrace is not without limitation. Under periods of heavy loading, the snapshot period is shorter than the nominal sixty seconds, due to memory exhaustion of the Lustre debug log. MDSTrace is currently run once every ten minutes, which can cause short, bursty, performance issues to go unreported. These issues will be addressed in future work.

3.2 DDNTool

The Spider filesystem is built around storage controllers from Data Direct Networks. These controllers have an API that allows performance and fault information to be queried over the network. Since there are 96 separate controllers in the filesystem, querying each one individually is impractical. Some means of regularly polling the controllers and aggregating the results is required.

The utility which does this is called DDNTool. It manages the connections with the DDN storage controllers, polls each one for various pieces of information at regular rates and stores this information in a MySQL database.

It should be noted that the database isn't being used to accumulate any kind of history: new data is constantly overwriting the old data. Rather, the database is being used for the other features it provides: simultaneous access from multiple clients, a well known API for querying the data and language bindings for just about every popular programming language. One other benefit of this architecture is that the DDN hardware only has to support a single connection. Since the client programs all connect to the MySQL server, multiple users can be monitoring the system without the risk of overloading the DDN hardware and slowing their performance.

Merely storing the data in a database isn't sufficient, of course. Several clients have been written so far to query the database for specific information and display it in an understandable format. A good example is a simple Python script that queries all the temperature sensors (14 sensors per DDN, 1344 in total) and displays only those who's temperature is above a specified value. Because the data is already in a database, this client only requires a single SQL query and less than 50 lines of Python code. Several other clients have been written to query other data and even the standard 'mysql' client is useful for simple queries.

Another example use of the DDNTool is for capturing historical performance information. Using this tool a number of metrics are captured such as storage system bandwidth usage and aggregate I/O operations per second (IOPs). Figure 2 illustrates the max hourly data rates delivered by 1/2 of the total storage controllers over the month of March. IOPs over the same period are illustrated in Figure 3. This and other performance related data has been collected using the DDNTool and archived into our performance analysis database for over 6 months as of this writing.

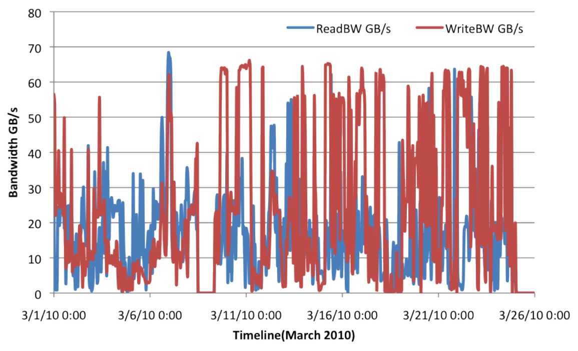

Figure 2: Max Data Rates (hourly) on 1/2 of the storage controllers

3.3 Long Term Monitoring of DDNTool Data

Current work in this area pulls performance and fault data from the DDNTool database and uses that information to generate web pages that show read/write peaks on a per minute basis, generates graphs showing the delay in committing a SCSI transaction to disks and also updates the total amount stored on disk daily.

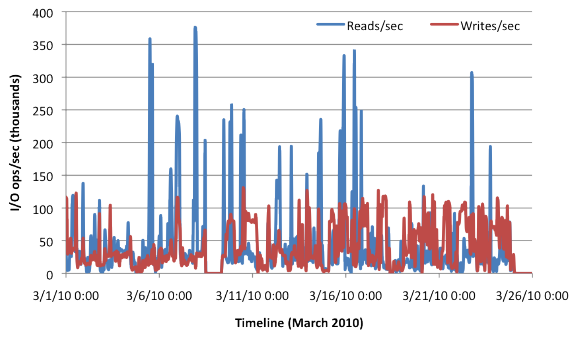

Figure 3: Max IOPs (hourly) on 1/2 of the storage controllers

4 Open Source Tools

The open source community has created a variety of tools and utilities that we use. Some are designed for general-purpose server administration and some are Lustre-specific.

4.1 Nagios

The health of the filesystem can be closely tied to the health of the underlying hardware and software that runs the filesystem. We monitor the health of the hardware and software by using Nagios[1] to query several items via SNMP. Nagios can query via SNMP or can be set up in a server/agent configuration. It can also be extended by using various popular scripting languages and querying custom SNMP OID's. We use this approach for several of our queries.

OSS health is monitored by querying a variety of parameters. First, we query the Dell OpenManage stack for chassis health (CPU temperature, power supply voltage and on-line status and fan speed). Next, we wrote custom Nagios extensions that monitor the health of the Infiniband connections to the back-end storage as well as the health of the Infiniband connection to the center wide Scalable I/O Network (SION). Additionally, we wrote a custom extension for monitoring the multipath status for the connections to our back-end storage. We also monitor the ssh daemon and ping the host to be sure that the Ethernet management network is healthy. Finally, the system load average is polled and graphed so we can attempt to correlate high system load on specific OSS nodes to application behavior.

4.2 Simple Event Correlator + Syslog

The Simple Event Correlator [2] combined with the Lustre server syslogs allows the administration staff to generally parse error messages from the Lustre software stack and from processes running on the Lustre servers and to gain more knowledge about the health and status of the filesystem. Rules are written to alert administrators of certain conditions, as well as combinations of conditions. For example, one of the SEC rules looks for syslog messages that are printed at boot time and generates an alert indicating that that particular server has rebooted. Additional work is being undertaken to correlate these logs to other events within the compute environment to gain a complete picture of the I/O impact of scientific applications.

4.3 Ne2scan + Genhit + Purge + Fsfind

Since Spider is primarily used as scratch storage for the supercomputers, out of date files are subject to being purged after two weeks. To perform this task, we use a combination of tools developed or modified at at Lawrence Berkley National Laboratory's National Energy Research Supercomputing Center (NERSC): ne2scan[4], genhit, purge, and fsfind[3]. The combination of ne2scan, genhit and purge allow us to remove the out of date files with minimal impact to the users. Fsfind is a related tool that allows reasonably fast file queries.

4.3.1 Ne2scan

Ne2scan is a modification of the e2scan utility that comes from e2fsprogs. These modifications were done by staff at NERSC. Lustre ships a modified e2scan that is aware of Lustre attributes and this work extends those modifications. Ne2scan scans the metadata from Lustre and outputs it as plain text that can be further processed with the other utilities discussed in this section. The act of scanning the metadata is not intrusive to applications and user interactive sessions because it does not modify anything and it does not use the Lustre POSIX layer to read the metadata. The metadata is instead read directly from the block device. In our current configuration, the working set of file metadata fits in memory on the Metadata Server (MDS), so any disk reads are almost invisible to the filesystem users. This could change in the future, but it is highly unlikely that user performance would ever be significantly impacted by the invocation of ne2scan.

4.3.2 GenHit

The genhit utility allows the administration staff to specify a date range and filter the results from ne2scan. We use this utility to generate the listing of files to delete based on the OLCF's Data Retention/Sweep policy. This utility requires access to the output of ne2scan and sufficient storage to write its own output file. It does not, however, have to run on the metadata server.

4.3.3 Purge

The purge utility will use the output of genhit and delete files from the filesystem. If first performs a stat on each candidate file using the Lustre client POSIX layer. It verifies that the mtime, ctime, and atime meet all the requirements for deletion. If they are all met, then the utility calls unlink on the path and moves to the next candidate file.

All candidate files should meet these requirements or they would not have been included in the genhit output. However, it is possible that a file may have been accessed between the time ne2scan was run and purge finally attempts to delete the file. In such a case, the file will not be deleted.

The whole process - Ne2scan + Genhit + Purge - usually takes 48-72 hours to complete. The exact time required mainly depends on the number of files in the filesystem.

4.3.4 Fsfind

The fsfind utility also makes use of the ne2scan output. This utility is used to generate information that is useful for profiling the data stored in the filesystem. The Spider administrators can query for users that have files on specific Lustre Object Storage Targets (OST) or for all files on a particular OST. Administrators can also query for files with certain striping patterns (groups of OST's). Additionally, they can query for all files with specific user or group permissions or for files that are setuid or setgid. Because it only reads the ne2scan output it also does not need to run on the MDS and queries only take a few minutes instead of several hours.

5 Future Deployment

There are a number of other tools that we haven't deployed yet, but are either considering or have plans to deploy.

5.1 LMT

The Lustre Monitoring Tool [5], provides another visual representation of filesystem activity and performance. This tool was successfully deployed on past filesystems, and work is underway to include the Lustre server side agent as the OLCF returns from the next available site-wide filesystem outage.

5.2 Infiniband Fabric Monitor

The deployment of an Infiniband Fabric Monitoring solution would help us to find under performing or dead links in the fabric. It could also help us to better visualize the network, and help to pinpoint bottle-necks in our inter-switch linking.

5.3 Lustre Server-side Client Statistics

Lustre server-side client statistics were abandoned in the current production version of Spider because of the way that memory was allocated on the Lustre server. At our high client count (26,800), memory utilization for server-side client stats was over 11GB. Each server only has 16GB and our diskless deployment of the base operating system consumed 1GB. With the client statistics and OS in memory we were experiencing kernel panics and out of memory (OOM) conditions fairly regularly. We worked with the filesystem engineers at Sun/Oracle to disable this feature, and helped them to reduce the impact of this feature in future releases. We will add back this feature when we deploy a Lustre 1.8.X filesystem in the near future.

5.4 Lustre DU Replacement

One of the things the admin staff would like to know about the filesystem is which users and projects are using the most space. The standard 'du' command could report this, but would take days to run.

As a replacement for 'du', we wrote another utility that reads the output of ne2scan. This utility records the user and group ownership of each directory as well as the size of all files stored under that directory and stores that information in a MySQL database. For space reasons, the size and ownership of individual files is not stored.

Obtaining the size of files is a little bit tricky since size data is not stored on the metadata server. The size of a file is computed from the size of all of its component objects and that information is stored on the OSS's. We could obtain the size by going through the lustre client layer, but that would take too long. (That's exactly how the standard 'du' works, after all.)

Instead of going through the POSIX layer, we wrote a small daemon process that runs on each OSS. This process reads the block devices directly and can look up an object's size from its object id. The main process reads the pathname and user and group ownership information from ne2scan output and then requests sizes of each object from the daemons.

By storing the resulting size, name and ownership information in a database, du-like queries can be run in real-time. Note that although the queries return very quickly, the data is only as current as the most recent run of ne2scan.

This utility has been written and undergone preliminary testing but is not in production yet.

6 Conclusions

As we continue to deploy tools and help improve the I/O performance of scientific applications, it becomes more clear that the ability to identify which applications are causing specific I/O patterns is crucial to our success. Also, our efforts in real-time monitoring of the back-end hardware and analysis of that data will lead to better fault detection and fault prediction. Finally, the use of open-source tools that required only minor modifications allowed us to focus more closely on the more challenging and demanding problems specific to deployments at this scale.

References

[1] Nagios. `http://www.nagios.org`.

[2] Simple Event Correlator. `http://simple-evcorr.sourceforge.net/`.

[3] N. P. Cardo. Reaping the benefits of metadata. In *Lustre User Group*, 2010.

[4] N. P. Cardo and C. Whitney. Reaping the benefits of metadata. In *Lustre User Group*, 2010.

[5] J. Garlick. Lustre Monitoring Tool (lmt). `http://code.google.com/p/lmt`.

[6] G. M. Shipman, D. A. Dillow, S. Oral, and F. Wang. The spider center wide file system; from concept to reality. In *Proceedings of the Cray User Group Conference*, 2009.