XrootD Monitoring Validation

Derek Weitzel & Diego Davila

Summary
XRootD detailed monitoring (also called the f-stream) data stream was audited for correctness and completeness. This stream includes all file accesses that a XRootD server performs. A single OSG monitoring collector collects and processes monitoring data from all XRootD instances in the U.S. Several bug fixes were corrected in the monitoring collector, but none of that significantly affected the collected data. Features were added to the XRootD server that reduce the losses due to using the unreliable UDP protocol between the servers and the collector. **After these features and bug fixes were applied, the monitoring stream showed no loss of transfer monitoring data and was consistent with the server and client's behavior.**

Goals of the validation
Verify that the transfer accounting collected by the OSG XRootD Collector matches the usage from the XRootD server by the client. Additionally, verify that the data path from the collector to the visualization captures all usage.

Recommendations
The tests have shown that the weak point of the data pipeline is the unreliable communication between the XRootD servers and the central monitoring collector. The frame size feature added to XRootD resolved the lost packet issues for a single server. Even with this change, it is expected that losses will occur as the number of servers reporting to a collector increases. **We recommend that a scale test be performed to quantify message losses from the server to the collector.**

Tests Performed

All tests were performed using a single XRootD server running in a docker container within a fermicloud VM at FNAL. The monitoring collector was running in a docker container and hosted at the University of Nebraska-Lincoln.

We carried out 4 different types of tests: *full file, byte range, vector range* and *long lived* using a unreleased version of XRootD v5 (presumably v5.0.1) (da7d2c8fcbc677ab650142cce7475d4c147ff47f) built directly from the master branch for the XRootD server. For the *full file* and the *byte range* tests we used 2 different configurations: one that uses the "frame size feature" (see section "XRootD - UDP Fragmentation - fbuff addition") that we will call "patched" and another one that does not use it and we will call it "unpatched". For the rest of the tests we only used the "patched" version.

The difference in the configuration between “patched and “unpatched” is just the addition of the following attribute and value: `fbsz 1400` to the `xrootd.monitor` directive. This will limit the maximum size of a UDP package sent to the monitor collector to be 1400 bytes.

In the following we describe the 4 types of tests performed and their results.

Full file. This set represents the most basic tests, in which a set of files is requested at a specific rate. From the results of these tests (see Appendix-A/ table 1) one can notice that the higher the rate of files requested the more data was getting lost for the “unpatched” version. For the “patched” version no data was lost.

Using the *tcpdump* tool on the XRootD server side we were able to spot a pattern that indicated that the packets getting lost were significantly bigger than those successfully delivered to the collector. This observation lead to a change requested for the XRootD server that is described later on this document (see section “XRootD - UDP Fragmentation - fbuff addition”)

Byte range. In this set of tests, instead of requesting a full file, a range-of-bytes is requested. Both the file and the *seek* or the initial byte to be read are defined randomly, the number of bytes to be read is fixed for any given test but varies among different tests.
As one might expect the results for the “unpatched” version on these tests (see Appendix-A/ table 3) are similar to the the *full file* ones given that the situation is also very similar; many read operations executed within the same time window will produce big packets that are likely to get lost. In the case of the “patched” version no data was lost (see Appendix-A/ table 4).

Vector Range. In this type of test more than one byte range requests are packed together in a single request. The attribute “vector size” in the tests (see Appendix-A/ table5) defines the number of byte range requests packed into the vector for a given test. In every test the size of the byte ranges inside the vector are fixed and both the file and the seek of each byte range are defined randomly. In these tests no data was lost.

Long lived. This type emulates a job that requests multiple byte ranges of a single file over a period of time leaving idle time in between requests. In every test we emulate one or more jobs running in parallel. Every job will perform a fixed number of byte range requests of a specific size, the requests are separated by a defined time window of N-minutes and are done on a random file and initial byte position.

Ideally we would like to see all read operations performed by a job within a long period of time to be summarized in a single record within the collector but while conducting the tests (see appendix-A/ table 6) we found out that the XRootD server will send an artificial file-close operation to the collector after the client has been idle for 5 minutes, then send a new open-file operation if the client gets active afterwards. On the collector side this will look as if the user was opening, reading and closing a file several times which in turn will produce several records.

The above will produce monitoring data that shows the right amount of bytes being read by the client but having the records being chopped every 5 minutes could lead one to think that the client is opening and closing files again and again and not just being idle between reads. This has implications for how the monitoring data is interpreted by people who analyze it. We suggest that this is documented (where?) such that monitoring consumers understand this feature.

Bugs fixed

Collector - Missing *appinfo*

The *appinfo* attribute is used by VOs to add an arbitrary tag to file transfers. The *appinfo* was not included in the final data output from the collector and sent to the database. The collector incorrectly assumed that an identifier provided with the appinfo information matched the identifier with a connection. The identifier does not match the connection identifier. The collector was modified to use several attributes also transmitted with the appinfo information in order to match with a connection.
Resolution: Collector Bug Fix (several commits)

Collector - Message sent to wrong destination on reconnect

After a long period of idle time, the connection between the collector and the message bus could be closed. When the collector prepares to send the next message to the message bus, the collector will re-establish the connection but push the message into the StashCache queue rather than the WLCG usage queue. The original destination of the message was not saved after the reconnection to the queue, which caused the message to go to the default StashCache destination instead of the WLCG queue.
Resolution: Collector Bug Fix

XRootD - Sequence Numbers

XRootD sends sequence numbers with each packet. The collector originally assumed, incorrectly, that the sequence number should be incrementally increasing with each received packet. An examination of the XRootD code revealed the sequence number was incremented whenever a monitoring packet was created, even if it was not sent to one of the two possible destinations. Therefore, the sequence numbers could not be relied upon to detect missing packets. After a conversation with the developers, XRootD implemented destination specific sequence numbers that can be used to detect missing packets.
Resolution: XRootD Improvements

XRootD - UDP Fragmentation - fbuff addition

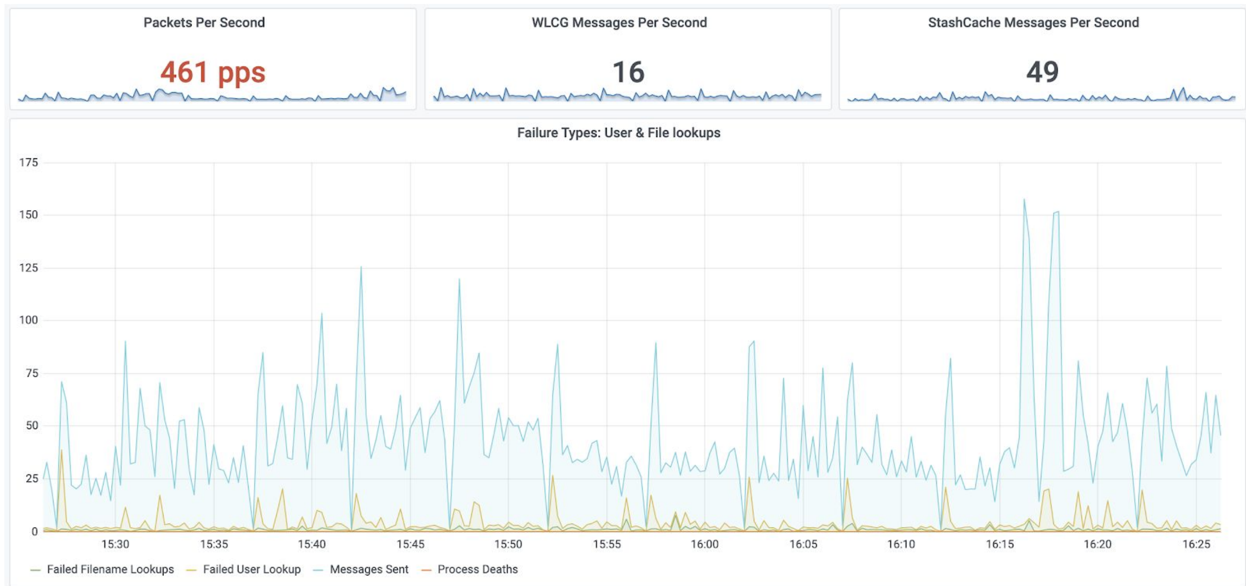
XRootD sends monitoring events in UDP packets. These monitoring packets can get large if many events occur within a flush window. These packets can become larger than the MTU that

the network can support, which will lead to packet fragmentation. UDP Fragmentation has been documented as a cause for significant packet loss [Cloudflare] [‘87 HP Paper]. During our tests, we were able to replicate near 100% packet loss when the packets became larger than the standard MTU. To mitigate UDP packet fragmentation, we submitted a change to XRootD that can limit the packet size sent to monitoring destinations.
Resolution: XRootD Improvements

Collector Improvements

Visibility of Operations

The monitoring collector was augmented with internal monitoring using Prometheus. A dashboard was created to display the internal status of the monitoring collector (shown below).



Appendix-A Test Results

Table 1. Full file unpatched

Id	Num. files requested	Request rate	Num. files correctly recorded 1	Num. files correctly recorded 2	Num. files correctly recorded 3	Num. files correctly recorded avg	avg success percentage
ff_01	10	1/s	10	10	10	10,00	100,00%
ff_02	10	5/s	10	10	10	10,00	100,00%
ff_03	10	10/s	10	10	10	10,00	100,00%
ff_04	50	1/s	2	19	1	7,33	14,67%
ff_05	50	5/s	10	20	0	10,00	20,00%
ff_06	50	10/s	0	0	0	0,00	0,00%
ff_07	100	1/s	11	0	0	3,67	3,67%
ff_08	100	5/s	15	0	0	5,00	5,00%
ff_09	100	10/s	0	0	0	0,00	0,00%
ff_10	100	20/s	0	0	0	0,00	0,00%

Table 2. Full file patched

Id	Num. files requested	Request rate	Num. files correctly recorded 1	Num. files correctly recorded 2	Num. files correctly recorded 3	Num. files correctly recorded avg	success percentage
ff_01	10	1/s	10	10	10	10	100,00%
ff_02	10	5/s	10	10	10	10	100,00%
ff_03	10	10/s	10	10	10	10	100,00%
ff_04	50	1/s	50	50	50	50	100,00%
ff_05	50	5/s	50	50	50	50	100,00%
ff_06	50	10/s	50	50	50	50	100,00%
ff_07	100	1/s	100	100	100	100	100,00%
ff_08	100	5/s	100	100	100	100	100,00%
ff_09	100	10/s	100	100	100	100	100,00%
ff_10	100	20/s	100	100	100	100	100,00%

Table 3. Byte range unpatched

Id	Num. of requests	Request rate	Range size	Num. operations correctly recorded	success percentage
br_01	10	1/s	512KB	10	100,00%
br_02	10	5/s	1MB	10	100,00%
br_03	10	10/s	5MB	10	100,00%
br_04	50	1/s	512KB	25	50,00%
br_05	50	5/s	1MB	5	10,00%
br_06	50	10/s	5MB	0	0,00%
br_07	100	1/s	512KB	16	16,00%
br_08	100	5/s	1MB	0	0,00%
br_09	100	10/s	5MB	0	0,00%
br_10	100	20/s	10MB	20	20,00%

Table 4. Byte range patched

Id	Num. of requests	Request rate	Range size	Num. operations correctly recorded 1	Num. operations correctly recorded 2	Num. operations correctly recorded 3	Num. operations correctly recorded avg	success percentage
br_01	10	1/s	512KB	10	10	10	10	100,00%
br_02	10	5/s	1MB	10	10	10	10	100,00%
br_03	10	10/s	5MB	10	10	10	10	100,00%
br_04	50	1/s	512KB	50	50	50	50	100,00%
br_05	50	5/s	1MB	50	50	50	50	100,00%
br_06	50	10/s	5MB	50	50	50	50	100,00%
br_07	100	1/s	512KB	100	100	100	100	100,00%
br_08	100	5/s	1MB	100	100	100	100	100,00%
br_09	100	10/s	5MB	100	100	100	100	100,00%
br_10	100	20/s	10MB	100	100	100	100	100,00%

Table 5. Vector range

Id	Num. of requests	Request rate	Vector size	Range size	Num. operations correctly recorded 1	Num. operations correctly recorded 2	Num. operations correctly recorded 3	Num. operations correctly recorded avg	success percentage
vr_01	10	1	3	512KB	10	10	10	10	100,00%
vr_02	10	5	5	1MB	10	10	10	10	100,00%
vr_03	10	10	10	1.5MB	10	10	10	10	100,00%
vr_04	50	1	3	512KB	50	50	50	50	100,00%
vr_05	50	5	5	1MB	50	50	50	50	100,00%
vr_06	50	10	10	1.5MB	50	50	50	50	100,00%
vr_07	100	1	3	512KB	100	100	100	100	100,00%
vr_08	100	5	5	1MB	100	100	100	100	100,00%
vr_09	100	10	10	1.5MB	100	100	100	100	100,00%
vr_10	100	20	20	1.5MB	100	100	100	100	100,00%

Table 6. Long lived patched

Id	job_id	Conn. duration (m)	Num. of requests	minutes between requests	Byte range size	job recorded
ll_01	0	3	3	1	512KB	OK
ll_02	1	6	3	2	1MB	OK
ll_02	2	15	5	3	5MB	OK
ll_02	3	20	5	4	10MB	OK
ll_03	4	25	5	5	512KB	OK
ll_03	5	50	5	10	1MB	OK*
ll_03	6	60	3	20	5MB	OK*
ll_03	7	90	3	30	10MB	OK*

(*) The amount of bytes read reported is ok, but the number of *file-open* and *file-close* operations is not accurate.