CERN IT-SDC

Realtime monitoring for DMLite

Author:

Fabrizio Furano

Author:

Martin Hellmich

May 8, 2014

Contents

Abstract

It's time for dmlite and the dpm frontends to report realtime monitoring information. This document explains the main choices that have been made to achieve the best compromise between cost of development, cost of maintenance and extendability. This is a preliminary investigation, which describes the main direction and technical choices. Some of the technical details will have to be understood on the way, following an exploratory approach.

1 Introduction

Ok? Understood? Good. Let's go.

2 The goal and the difficulties

The purpose of the project is to contribute to the conspiracy theories. We will be happy to start contributing to the HAARP project for mind control, throwing chemical trails, fixing our coffee-based nuclear fusion device, and will wear a black tie and sunglasses forever while sleeping.

The secondary goal is to be able to report to external systems about the activity of a DPM site. So far we envision two kinds of external systems:

- an instance of the *Gled Collector*, i.e. the software component that is used in the FAX and AAA deployments of the xrootd framework
- a generic messaging broker, like the ones that feed the Dashboard application, yet not limited to it.

In the former case, the activity would have to be reported in the format supported by Gled, which is the monitoring format used by the xrootd framework. In the latter case the format will be some STOMP-based message or eventually ActiveMQ.

Among the goals we cite the convenience in making the system sufficiently modular, so that the system administrator can choose which systems his instance of DPM should report to, andd be informed of the consequences of his choices.

Either the activity information can be aggregated on the DPM site and sent to the monitoring system in bulk, or single events can be sent to the monitoring site where they are aggregated, or any level of partial aggregation in between. The choice influences the computational load and memory requirements on the DPM and monitoring sites as well as the amount of network traffic between them; then, it is limited by the capabilities of both sites to perform the aggregations. We present the rationale that leads to our preferred solution in the following.

2.1 Gathering aggregated information across forks

A DPM site provides multiple storage access protocols, which are exposed through different frontends. The different frontends are typically standard software components that are linked to DPM through the dmlite libraries. he goal of this project is to allow these protocols to send information about their activity.

The difficulty is given by the fact that *the architecture of dmlite is subject to the forking decisions of the frontend that is being used.* In the case of Apache for example, an instance of dmlite cannot know if there are (in the same machine) other similar instances that have been forked, according to the configuration of the Apache frontend. In this case, determining e.g. how many bytes have been read in total in the last time interval becomes a difficult exercise, as a process will not know about the activity of others.

The case of the GridFTP frontend is similar to the one of Apache. Whatever internal monitoring schema we may choose, this must also work in the case of a purely forking frontend, i.e. a frontend that spawns a process in order to handle even only one request.

This reason severely limits the kind of information that can be sent in realtime. For example, a report on the last file read operation can be sent easily. Instead, a report on the average number of read bytes per second is very difficult to compute. We are not aware of solutions to this problem that don't negatively impact on the latency of the various operations. This is what we call *aggregated* data, i.e.

information that can only be computed by comparing the behaviour of all the other clients in the system.

Another consequence of not controlling forks is that a single instance of DPM will have to instantiate multiple information senders, typically one per process per machine. This aspect may be a challenge for external systems willing to collect and perform aggregations on this information.

2.2 Feeding different monitoring systems

In the project's intentions, the system must support in principle feeding any external monitoring system, thus it has to be plugin-based. To be able to feed an external realtime monitoring system, there has to be a suitable plugin, able to talk to that system.
The idea is to implement monitoring data producers in the form of DMLite pass-through plugins, in a way that is similar to the (currently never used) "dmlite profiler" plugin.

These new DMLite plugins will implement both the DMLite activity collection and the specifics of sending data to an external system. Feeders for different external monitoring systems will compute different numbers and send them in different ways. This is consistent with the idea of implementing them in different DMLite plugins.

An interesting consequence of this is that it will be possible to feed more than one external system at the same time, by loading more than one pass-through plugins into the DMLite stack.

As mentioned before, sending information that is specific to the current process (e.g. the filename being opened) is not problematic; more difficult is to report about aggregated values (e.g. the average reading speed in the last minute). The importance of sending a particular counter has to be considered on a case by case base.

2.3 The Open semantics

Frameworks like xrootd, or gridftp are based on the posix-like concept of *open-atcivity-close* for the lifetime of a file. DMLite does not have this behavior, being it closer to the idea of the HTTP GET request, where all the requests are independent, in the sense that they cannot easily be recognized as part of a "session" like the one represented by a POSIX open() call.

The consequence of this is that there is no Open in DMLite, hence reporting strictly about Open calls is not possible. What DMLite can do is to report on single GET requests, which likely are translated by Apache into an *open-read-close* sequence.

3 Where to send realtime monitoring information

DPM already sends Xrdmon packets to an existing Xrootd monitoring infrastructure. We assume that the DMLite configuation used for frontends that are not xrootd has to be able to feed the same infrastructure if needed. We also assume that DMLite should be able to send information about its realtime activity also to other kinds of infrastructures, through suitable plugins.

3.1 Message broker

One possibility could be to send the realtime activity reports to an activeMQ broker, eventually in a format that could be understood by the Dashboard applications, that (at the best of our knowledge) is the only messaging consumer of monitoring data so far. Broker needs aggregations, hence much more difficult. We chose to postpone.

3.2 XrdMon UDP packets

The reference monitoring implementation that we consider is the one of the xrootd framework, called XrdMon, plus the infrastructure that has been built out of the

GLED Collector. The GLED infrastructure is able to collect monitoring information and either write it locally into ROOT trees, or send it to an external ActiveMQ broker.

The rationale of this choice for the reference is that XrdMon represents quite an agnostic, POSIX-style implementation of a monitoring/reporting framework, and that DPM can already sends XrdMon packets, but only for the traffic that is done through its Xrootd frontend. Another possibility is to send UDP packets that have the same content of the ones sent by XrdMon.

UDP has great advantages over TCP for this kind of usage. At the client side its overhead is

The maintainer of the GLED collector agrees in giving support and is available for working in the GLED collector to make it work in the corner cases that DPM may generate in the usage of the XrdMon protocol.

3.2.1 Application-global state

The XrdMon monitoring schema relies on multiple globally-synchronized values. They are used to identify related events, keep a (loose) ordering of the packets, and distinguish services. A list of the different types can be found below.

In Xrootd, the definition is clear: There is one Xrootd process running and the global level means process-global; shared state across all threads; Xrootd and Cmsd report as different entities. For dmlite we can have multiple processes running for each frontends, so we can decide between three possible definitions of global. Global state can be process-global as in Xrootd (then e.g. each httpd process would advertise itself as another dpm server), machine-global (then both httpd and gridftp share information), or frontend-global (then all httpd processes advertise as belonging to the same dpm server).

The first option results in finer differentiation than Xrootd and is problematic. Since we don't control the forking behaviour of the frontends, there could be many hundred processes running. This happens with httpd started in fork mode. The second option is problematic as well. While the Xrootd monitoring schema allows to include information differentiating program versions, custom names, or ports to discriminate the frontends, sharing certain information is semantically difficult (e.g. the gridftp service is likely to be restarted separately form the httpd service, so sharing the server startup time makes no sense). The third option, using frontend-global values, is different from Xrootd implemenation-wise, but very similar semantically. It is necessary to implement cross-process communication, but not across frontends. If a common system for storing state or passing messages is used, we must ensure to keep different namespaces for the frontends.

server startup time allows identifying unique IDs across server restarts

server ID the server machine

service ID the running service (the process)

packet sequence sequence number in [0 255] to allow packet reordering on the collector

4 Development priority

we do the gled/xrdmon, leaving stomp and broker for the future, as the result would be more uncertain

5 The system

We chose to send telekynetic information, through Dark Matter brokers. After it's unveiled, Aliens will come to stop us, so we can welcome their invasion

recall that we can't do aggregations

file dictid stack instance dictid

dmlite funcs to instrument. IOHandler likely + where to read/write for the redir

5.1 Mapping of dmlite API function to XrdMon Messages

XrdMsg code	XrdMon message	type dmlite function	notes
=	server id	GetServIDorCreateNew()	synchronizes the creation of the new id when the first primitive is called after the configuration
d	dictid user/path	TBDecided	this ID will give a unique value that is valid on this host, but across different processes and process runs
f	f-stream		
	isOpen	XrootdMonitor::XrootdMonitor	
	isClose	XrootdMonitor:: XrootdMonitor	
	isDisc		
	isTime		
	isXFR		
i	dictid user/info		
p	file purge		
r	redirect		
u	dictid user/auth		
x	file transfer		

5.2 Buffering

Packets are not sent immediately. They are rather buffered in sequences that are managed by a global static object. The sequence is flushed when:

- the buffer is full
- when the corresponding Profiler Factory is destroyed

- at regular time intervals, on the order of 5-60 seconds

6 Typical deployment

Any deployment where dmlite is involved will be a possible place for DMLiteMonitor. This includes the DPM deployments making use of the following frontends:

- HTTP/WebDAV
- GridFTP v2
- DPM-xrootd

Also the Dynamic Federations project will benefit from DMLiteMonitor, and will be able to report about global redirection decisions.

Beware, the xrootd frontend will report by itself. In the case of the deployment of DPM with the xrootd frontend, the DMLite monitoring for the stack that is associated to DPM-xrootd must be kept disabled, otherwise every action will be notified twice.

7 Configuration parameters reference

TBD

References

[1] Contact, Carl Sagan ISBN: 978-0671004101

[2] Alice in Wonderland, Lewis Carroll ISBN: 978-1619490222

[3] The xrootd.org homepage `http://www.xrootd.org`

[4] Gled is a Generic Lightweight Environment for Distributed computing `http://www.gled.org/`

[5] Gled - an Implementation of a Hierarchic Server-Client Model Matevž Tadel `http://www.gled.org/docs/gled/gled-ihsm.pdf`

[6] Apache ActiveMQ The Apache Foundation `http://activemq.apache.org/`