Bring network and time together using Linux tracing

Alexander Aring
RedHat, Inc.
Ottawa, Canada
aahringo@redhat.com

Abstract

This paper is about visualizing a distributed network protocol by using the trace-cmd [12] time synchronized trace events feature. As example, we use the Linux Distributed Lock Manager [13] (DLM) protocol to visualize lock states over time in the Jumpshot [7] viewer.

Trace-cmd is the user space tracing utility to control the Linux in-kernel tracing subsystem. Recently a new feature was introduced to record multiple Linux machines tracing events with their timestamps synchronized across those machines.

The Linux Distributed Lock Manager (DLM) subsystem is a distributed network protocol used by Linux clusters to control mutual access to shared resources. Current DLM debugging methods are limited by dumping lock states via command line interfaces e.g. debugfs. Those dumps can only be taken sequentially and without being time synchronized. Means it will not represent all lock states at one time. Additionally, those cli dumps need to be merged on your own to see a connection between them.

The slog2sdk [6] containing the viewer Jumpshot will be used to represent the DLM protocol lock states over time by using a GANTT chart [14]. Therefore, a trace converter dlm2slog2 [1] was developed to build a bridge between those components of trace-cmd/Linux tracing subsystem and slog2sdk.

In this paper I will show what the steps are to record a time synchronized DLM trace by using trace-cmd and how those are converted to visualize them in Jumpshot. This approach can be adapted to other distributed network protocols as well and is not limited for debugging use cases. Moreover, we will look into possible new ideas on how to use time synchronized tracing events in a distributed network.

Keywords

Distributed Lock Manager (DLM), slog2, Jumpshot, trace-cmd, Linux Tracing, Message Passing Interface (MPI), Precision Time Protocol (PTP), Kernel-based Virtual Machine (KVM), Time Synchronization, Global Clock, Distributed Network, Cluster, dlm2slog2

Introduction

This section will show the short introduction into the relevant topics to give a basic knowledge about them.

Cluster

A machine used in a cluster is named a node. Multiple nodes can form a cluster. Those nodes can communicate to each other and having an addressing scheme of a node ID. A node ID is number representation of a machine inside a cluster. Besides nodes a cluster contains shared resources which can be accessed by those nodes. A shared block device is an example for a shared resource and can be used among the cluster nodes by an e.g. a cluster file system such as GFS2 [9] or OCFS2 [10]. To form a cluster you usually need a cluster manager running on all cluster nodes. An example for a Linux cluster manager is corosync [8].

Distributed Lock Manager

A Distributed Lock Manager can be used in a cluster to control mutual access of shared resources among cluster nodes. There are several different DLM implementation out there. One historical implementation is the OpenVMS DLM implementation which isn't in any way Open as Open Source. In this paper we only look at the Linux DLM implementation that can be found in fs/dlm of the Linux kernel source code.

Lockspace A lockspace contains a set of locks. The user can create one or several DLM lockspaces per cluster. A DLM lockspace itself is a cluster resource and cluster nodes can join a DLM lockspace. A lockspace is referenced by it's lockspace name or global ID.

Locks A lock is represented as a lock block (lkb) inside Linux DLM subsystem. Such a lkb is a local representation only but points to a resource block (rsb) which is a lockspace wide representation. In this paper we only handle rsbs as lockspace wide lock representation.

Resource Name A rsb contains a unique lockspace wide resource name (resname) identifier. By using the resname a node can request lock operations to a specific instance.

Modes DLM supports different Lock modes. The simplest lock modes are an exclusive and null. There exists more lock modes, but this is out of scope of this paper. Depending on the mutual access use-case the shared resource can be protected in a different lock mode. There exists a compatibility matrix to see which mode is compatible or incompatible to each other. Lock modes incompatible to each other can't be used twice at the same time for the same lock inside a lockspace.

Master DLM works with lock masters. Each rsb points to a lock master node and can be any node in the lockspace. The lock master maintains the lock and accept requests for lock operations. If the lock master itself performs a lock request to the lock it's master to it there this will be considered as local request.

Linux Tracing

In this paper we only handles the upstream Linux tracing functionality. This contains the Kernel Tracing subsystem and their user space provided software.

trace-cmd The utility tool trace-cmd uses libtracefs and libtraceevent to provide a simple interface to interact with the Linux tracing subsystem. Such as record traces, parse and print them on the command line.

Time synchronization A recently introduced feature in trace-cmd is to record traces among several machines and store their per CPU clock offset as metadata in their recorded trace files. In this paper we will use this feature to graphical displaying the lock modes over time.

Libraries All libraries provided by the Linux Tracing subsystem for user land are written in C. The Linux tracing subsystem currently provides the following libraries which are used for this paper work:

libtracefs To control the Linux tracing in-kernel subsystem there exists a UAPI offered by file system API.

libtraceevent Higher abstraction of representing a recorded trace events in C.

libtracecmd A static library inside trace-cmd. Can do all trace-cmd functionality inclusive parse recorded file traces by trace-cmd.

Important in this paper is libtracecmd which depends on libtracefs and libtraceevent.

Trace Event A trace event has a system such dlm to represent the dlm subsystem trace events as group and a name for example the function which was be traced. Trace Events also has fields which are either common fields like the timestamp as the event was recorded or the recorded PID or user specific fields which can be provided by parameters.

Kernel-based Virtual Machine

In this paper we will use only a virtual Linux cluster. This means our cluster machines are guest machines only. Those machines using the KVM subsystem with interacting with the host machine.

CPU clock offset Trace-cmd can do a read KVM debugfs entries to determine virtual CPU clock offsets with the host. This method of trace-cmd for time synchronization will be used in this paper.

VSOCK Sockets To communicate from host to guest machine and vice versa trace-cmd can use VSOCK sockets. This is a special socket type with an address scheme of Context ID (CID). A CID is a number representation depends on the used environment.

slog2sdk

The slog2sdk is a Open Source SDK for generate, convert and view slog2 files. It is developed by Argonne National Laboratory for Advanced Numerical Software (LANS). Originally programmed for visualizing MPI applications but not depending on any MPI interface specification. It is written in Java. Inside the slog2sdk is the graphical viewer Jumpshot and a Java library traceTOslog2 to create slog2 tracing files.

slog2 file format The slog2 file format mainly tackles an issue with very big tracing files. It is described more detailed in the slog2 paper [7] but in general it's about to reading full trace files to display them in a viewer. Under the assumption that storage is cheap, and trace files can be really large loading the whole trace file in a viewer on an overall view level can end in high memory pressure. To solve this issue slog2 introduced a level of detail view. In short slog2 uses a tree-based file structure and each level represents a more combined view (aka preview events) of the underneath detailed trace view. An overall view of the slog2 file in Jumpshot will not read and parse the whole file, it is only a part of the trace file. If the user wants a more detailed view, then only the necessary section in the trace file will be read and parsed.

Topologies Slog can store with additional x and y coordinate information different topologies. A topology is a drawable component that the slog2 viewer can draw in a graphical way. There are three different topologies:

Event One x/y coordinate only and can represent a trace event as a pin needle.

State Two x and one y coordinate. Can represent a state over time like shown in a GANTT diagram.

Arrow Pair of x/y coordinate. Can show a communication between nodes.

Jumpshot The Jumpshot viewer can open slog2 files and represent the tracing file in a graphical way. It has zooming capability and shows a legend of instances of slog2 topologies.

Preview Events Jumpshot has a concept of preview events. A preview event exists to scale up the level of detail in a specific time interval of a slog2 file. If zoomed in the preview events will be shown up as scalar topology types. This is however as mentioned the idea behind slog2 as it does not read the full trace file when opening the file.

SWIG

In this paper we will use SWIG [11]. SWIG is a compiler to provide bindings from one programming or scripting language to another one.

Interaction of used Software Projects

In this section we discuss how to connect the topics discussed in the introduction section and how we connect them to together.

Workflow

At first, we can define a generic workflow about the necessary steps and map them by the used Software project. Those steps are:

Recording Linux DLM Tracing and trace-cmd

Filtering and Converting libtracecmd, dlm2slog2 and traceTOslog2

Viewing Jumpshot

The steps Linux DLM, recording and viewing can be accomplished by using the existing software components mentioned in the instructions. A missing piece and the main part of this section is to describe the idea of dlm2slog2 which acts as a bridge between the steps of recording and viewing. Each step has requirements which the following text will talk about.

Linux DLM Tracing

We need to have requirements to the Linx DLM subsystem that we can record the ongoing DLM API on each node.

Tracing Events Trace events are being used to record the DLM API usage on a specific machine. There are several ways to records trace events like explicit introducing trace events or using a dynamic way with kprobe. In this paper we are only using explicit trace events which are declared by the TRACE_EVENT() macro inside the Linux kernel. Those are considered more stable trace events and rarely changed, whereas dynamic kprobe trace events can be changed any programming changes affected to them. Although trace events are never be considered to be stable.

The DLM functionality which will be traced are:

dlm_lock() This function will be called to request a lock mode. This can either be a conversion or an initial lock request (if the lkb didn't exist before). A lock mode must always be given and a resource name for the rsb.

dlm_unlock() This function can request a unlock of a DLM lock. Note that the lifetime of an lkb/rsb can be ended in this case whereas the lock mode NL can unlock a lock as well by using dlm_lock.

ast All lock requests are asynchronous calls, if a result arrived which can be successful or non-successful the ast (asynchronous system trap) callback for the specific lock request will be called.

bast If the node and a specific lock occurs lock contention on another node because the lock modes are incompatible the bast (blocking ast) callback will be called, and the user gets aware of this circumstance. As parameter the caused incompatible lock mode will be given.

Each of the trace events has user specific fields such as lockspace ID or resource. Those fields will be used to keep track of the lock requests and their result when analyzing DLM trace events.

DLM Hierarchy DLM has a hierarchy between lockspaces, lock resources and nodes. This hierarchy looks like the following:

- Lockspace A
  - Lock Resource A
    - Node A
    - Node ...
  - Lock Resource ...
- Lockspace ...

This hierarchy needs to being used to represent slog2 topologies for the user. It allows a simple combined per node ID lock resource view.

Capturing with trace-cmd

The utility tool trace-cmd provided by the Linux tracing subsystem can be used to record DLM trace events into a trace-cmd specific trace file format. Such functionality is necessary to provide the recorded trace events from the DLM subsystem to the user. Therefore, we need to tell trace-cmd which trace events need to be recorded. We either do a record the whole DLM tracing system or specific apply filters for the necessary trace events. To represent the DLM state close to the users view we provide all exported DLM API trace calls. Those are:

- dlm:dlm_lock_start
- dlm:dlm_lock_end
- dlm:dlm_unlock_start
- dlm:dlm_unlock_end
- dlm:dlm_ast
- dlm:dlm_bast

The start and end trace events takes an issue when recording asynchronous system calls and will be described later in this paper.

Time Synchronization The trace-cmd utility tool recently introduced a feature about recording time synchronized traces for several machines. Each machine will still record their own trace-file with the recorded trace events given by a filter parameter. Trace-cmd stores several per CPU clock offsets as trace file metadata. Those offsets reference to a global clock source on the host machine.

Trace-cmd uses different approaches to communicate and perform a time synchronization to calculate a clock offset. This can be done by an IP based or VSOCK based socket communication. The time synchronization method can either be PTP or KVM (in trace-cmd known as x86-tsc).

In our setup we only deal with a cluster with virtual machines. For this reason we can use VSOCK sockts for communication which should be the preferred solution for a virtual machine environment. As time synchronization method the kvm based should be used for the same reason. Those are only available in a virtual machine environment. If a bare metal cluster is used, it is required to use other methods for communication and time synchronization such as IP based communication or PTP time synchronization.

slog2sdk

In this work the slog2sdk was not be changed. A fork was made of slog2sdk [2] to update the build system to provide the build of traceTOslog2 and Jumpshot with a recent Java compiler.

dlm2slog2

This section describes the idea of dlm2slog2 which is needed to transform Linux DLM traces to the slog2 file format. Between input and output there is a processing of the input trace files to map them into higher slog2 topologies, such as states. In this section we will describe how this step works.

Parsing We need to parse Linux trace files in Java to analyze the recorded Linux DLM traces. To parse Linux trace files we can use the C libtracecmd library. Cause the slog2sdk is written in Java we use swig [11] to generate the necessary Java bindings. Those bindings offer a higher-level Java library named tracecmd.jar which statically links to all necessary depending on libraries such as libtracecmd, libtraceevent and libtracefs. A new project trace-cmd-java [3] was created to generate those necessary Java bindings to libtracecmd.

Trace-cmd delivers as record output a per machine trace file. This trace-file need to be merged. To keep track on which node ID the trace file belongs to we need a mapping between the trace file and their node ID inside the cluster.

Asynchronous API Call and Trace Event Merge The following section will describe an overcome problem while trying to record an asynchronous API call with the Linux Trace subsystem. The ast and bast callbacks could arrive before dlm_lock() and dlm_unlock() returns, due the fact of the behavior of an asynchronous function call. To handle this particular race we need to record those functions as start, means before the lock request is initiated, and end, means after the lock request was initiated. The Linux trace subsystem has currently no higher-level mechanism to combine those two callbacks into one which is necessary here. It is required to merge those trace events into one and respecting the conditions of an DLM asynchronous API call.

Because the behavior of the DLM API we can only expect an ast callback if dlm_lock or dlm_unlock call is successful. In DLM that means that the function returns zero. If the function returns non-zero then it is considered as failure and no ast callback will occur. Because keeping the trace events chronologically we need the timestamp at the start of the function call but at this point we don't know if the function is successful or non-successful. For this reason a merge between start and end of those two trace events need to be done which contains the timestamp from start and the return value of end trace event. All other additional user specific trace event fields should stay the same. To doing such merge there is a mapping required between start and end trace event that application specific is.

This mapping is done by using the lockspace ID, lock ID and PID in both trace events. That means if a start event occurs, those fields will be remembered and being matched with an upcoming end trace event. The timestamp of start event will be taken and merged with the same fields inclusive the return value of the end event.

Filtering A filter is required due the fact that slog2 scales into the X-Axis but not into the Y-Axis. If later in the viewing step in Jumpshot are too many locks required to be displayed into the Y-Axis it might end in high memory pressure as the Y-Axis displays the DLM hierarchy.

To filter out any resources from the recorded trace files we analyze all provided trace files and provide them as a list of combination of lockspace ID and resource name to the user. The user has the choice to select specific locks which might be interested. At the converting process only the selected locks will be considered.

traceTOslog2 To generate slog2 as an output file the slog2sdk offers the Java library traceTOslog2. The traceTOslog2 requires providing all topologies that should be part of the generated slog2 file. In this paper we only use the topologies event and state.

The slog2 event topology will be simply mapped to a Linux trace event. For states, we need to parse Linux trace events and interpret their time interval. This means from a successful dlm_lock or dlm_unlock call and an upcoming ast callback can be assumed. During this operation we don't assume another call can happen. This behavior is against the DLM API and would probably end in a non-successful DLM API call. The time between function call and ast callback can be intepreted as lock change is in progress. Another usage of state primitives is the actual lock state on a per node perspective. To separate different slog2 events and states a color can be defined that in a later graphical view it can be easily separated from each other.

Line ID A line ID is a slog2sdk concept and represents one Y-Coordinate in the slog2 file format. In out case the line ID is a generated hash value of the DLM hierarchy as lockspace ID, lock resource name and node ID. Using a Map from line ID to a list of trace events allows us to collect all trace events in a sorted list according the timestamp. Having those trace events in such order allows to sequentially parse the DLM API operations. During the sequentially parsing of trace events per line ID the slog2 topologies can be generated such as event and state as mentioned earlier.

YCoordMap Besides topologies traceTOslog2 can declare an optional YCoordMap for the slog2 file format. A YCoordMap describes a mapping between line ID and a hierarchy structure in the Y-Axis. The Y-Axis can be separated into columns to display them in a hierarchy way.

This feature need to declared in a multidimensional array and belongs to single Y-Coordinate. Such a YCoordMap will be used to reflect the DLM hierarchy structure. In our case the YCoordMap will map column indices to our defined hashed line ID which takes lockspace ID, resource name and node ID in account.

Generate slog2 file

Taking the process mentioned before in account we can generate a slog2 file as output. At the same time it is a converting from Linux recorded tracing files into the slog2 file format with additional interpreting of DLM specific trace events and visualize them in slog2 specific topologies.

Jumpshot

Jumpshot is a standalone Java application that is part of the slog2sdk. It is the main application to view slog2 files in a graphical way. As mentioned it uses preview states in kind of level of detail behavior to scale the X time Axis to allow loading of large trace files. The Y-Axis will be displayed as provided by the user specific YCoordMap.

How to use Step-by-Step Guide

In this section we will show a step-by-step guide how to use every mentioned software project. At the end a slog2 file of the DLM activity will be created and can be viewed by Jumpshot.

Environment

The environment used in this paper is a three node virtual node cluster. They are connected via a software bridge on the host to offer IP based communication. Although it offers VSOCK socket communication as well. Each virtual machine can be addressed therefore by its CID. There is a mapping between VSOCK and cluster node ID:

VSOCK CID	Cluster NodeID
5	1
4	2
3	3

The mapping values depend on virtual machine setting and cluster setup. They might be different on a different machine. For this paper we assume such a mapping from CID to node ID.

trace-cmd

This section will describe how to use trace-cmd to record Linux DLM traces. Starting a time synchronized trace recording will generate four trace-cmd specific trace files. Those trace files are necessary as input for dlm2slog2.

Cluster Guest Machines On each cluster node we need to start a trace-cmd agent. A trace-cmd agent will act as a background service to accept new clients to start trace events recording on the specific machine. This can be done by:

```
$ sudo trace-cmd agent
```

Afterwards on the command line the process will block and the following message will appear.

```
listening on @3:823
```

This means that the agent is listening on CID 3 to accept new clients.

Host Machine On the host machine we need to connect to all cluster nodes and start to record Linux traces. This can be done by:

```
$ sudo trace-cmd record -p nop \
  -A 5 -e dlm:dlm_lock_start -e dlm:dlm_lock_end \
  -e dlm:dlm_unlock_start -e dlm:dlm_unlock_end \
  -e dlm:dlm_bast -e dlm:dlm_ast \
  -A 4 -e dlm:dlm_lock_start -e dlm:dlm_lock_end \
  -e dlm:dlm_unlock_start -e dlm:dlm_unlock_end \
  -e dlm:dlm_bast -e dlm:dlm_ast \
  -A 3 -e dlm:dlm_lock_start -e dlm:dlm_lock_end \
  -e dlm:dlm_unlock_start -e dlm:dlm_unlock_end \
  -e dlm:dlm_bast -e dlm:dlm_ast
```

The parameter -A is the CID which should be connected to record traces. The argument -e will add a whitelist filter for a specific trace event or tracing system.

If everything goes well the following message will appear:

```
Negotiated kvm time sync protocol with guest 3
Negotiated kvm time sync protocol with guest 4
Negotiated kvm time sync protocol with guest 5
Hit Ctrl^C to stop recording
```

It shows that trace-cmd started recording time synchronized DLM traces. At this point every DLM usage will be recorded until trace-cmd receives SIGINT. After receiving the SIGINT signal trace-cmd will quit and the following files are available:

```
trace-3.dat trace-4.dat trace-5.dat
```

Those files are the per machine trace file which are necessary for dlm2slog2. By default, the naming scheme uses the CID of the virtual machine inside it's trace file name. Note there is a host trace file trace.dat which is out of interest as trace-cmd stores metadata information into that file.

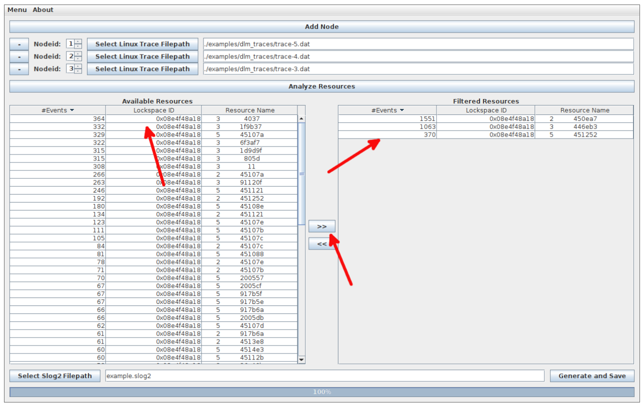
dlm2slog2

The developed program dlm2slog2 is a GUI application. It takes a mapping between node ID and Linux DLM trace files as input, and its output is a generated slog2 file. There exists a way to preconfigure the necessary input and output parameters by:

```
dlm2slog2 \
--traces 1=./examples/dlm_traces/trace-5.dat:2=./examples/
    dlm_traces/trace-4.dat:3=./examples/dlm_traces/trace-3.dat \
--slog2 example.slog2
```

Pressing an analyze button will analyze all provided trace will and lookup all resource names which where recorded of the trace event. Available lock resources will be presented in a table and can be moved from available resources to a filtered table. The filtered table acts as whitelist and only those who are moved to the filtered table will be considered for the generated slog2 file.

An output slog2 file path must be defined to provide an output file for the slog2 trace file. Clicking on Generate and Save button will start parsing and interpret the provided DLM Linux traces and convert them to slog2 file format. In this process the slog2 will use different topologies to represent trace events as events and lock states as GANTT like chart.



-	Nodeid:	Select Linux Trace Filepath	
-	1	Select Linux Trace Filepath	./examples/dlm_traces/trace-5.dat
-	2	Select Linux Trace Filepath	./examples/dlm_traces/trace-4.dat
-	3	Select Linux Trace Filepath	./examples/dlm_traces/trace-3.dat

Available Resources

#Events	Lockspace ID	Resource Name	
364	0x08e4f48a18	3	4037
332	0x08e4f48a18	3	1f9b37
329	0x08e4f48a18	5	45107a
322	0x08e4f48a18	3	6f3af7
315	0x08e4f48a18	3	1d9d9f
315	0x08e4f48a18	3	805d
308	0x08e4f48a18	3	11
266	0x08e4f48a18	2	45107a
263	0x08e4f48a18	3	91120f
246	0x08e4f48a18	5	451121
192	0x08e4f48a18	2	451252
180	0x08e4f48a18	5	45108e
134	0x08e4f48a18	2	451121
123	0x08e4f48a18	5	45107e
111	0x08e4f48a18	5	45107b
105	0x08e4f48a18	5	45107c
84	0x08e4f48a18	2	45107c
81	0x08e4f48a18	5	451088
78	0x08e4f48a18	2	45107e
71	0x08e4f48a18	2	45107b
70	0x08e4f48a18	5	200557
67	0x08e4f48a18	5	2005cf
67	0x08e4f48a18	5	917b5f
67	0x08e4f48a18	5	917b5e
66	0x08e4f48a18	5	917b6a
66	0x08e4f48a18	5	2005db
62	0x08e4f48a18	5	45107d
61	0x08e4f48a18	2	917b6a
61	0x08e4f48a18	2	4513e8
60	0x08e4f48a18	5	4514e3
60	0x08e4f48a18	5	45112b

Filtered Resources

#Events	Lockspace ID	Resource Name	
1551	0x08e4f48a18	2	450ea7
1063	0x08e4f48a18	3	446eb3
370	0x08e4f48a18	5	451252

Select Slog2 Filepath: example.slog2 — Generate and Save — 100%

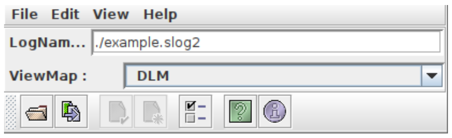
Figure 1: dlm2slog2 GUI with shows available resources (left side) and filtered resources (right side)

Figure 1 shows an example screenshot of dlm2slog2 the red arrows shows the related parts involved into resource filtering.

Jumpshot

The created slog2 file generated by dlm2slog2 from the previous step should be used here.

If a slog2 file is opened, three windows will be showing up. A legend window, File Information window and a Graphical Timeline window. For a reference of the full functionality please see the Jumpshot manual [5].



LogNam... ./example.slog2

ViewMap : DLM

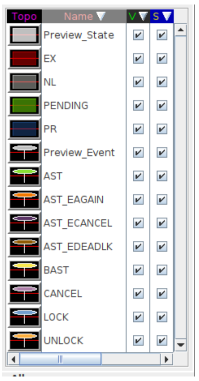
Figure 2: Jumpshot main window with example.slog2 opened and ViewMap is set to DLM

Main The main window as figure 2 shows the opened files and a Menubar with additional functionality. One section is the ViewMap which defaults points to the DLM ViewMap.

Selecting the ViewMap to Identity Map will switch to a one dimensional line ID view. The Y-Axis will switch to the line ID view which makes not much sense because the line ID is a hashed value of lockspace ID, lock resource name and node ID.

Legend The legend window as figure 3 shows the slog2 topologies and their specific colors. The block topology is hereby a state topology and the needle are events that will be drawn component in the timeline window. Topolgies EX, NL and PENDING are representing the lock mode in a lock state. All others represent dlm_lock, dlm_unlock, ast and bast handling. There exists additional handling to colorize a specific event e.g. if the ast returned EAGAIN as result.

Timeline Figure 4 shows the timeline window that is the most important window because it actually shows the DLM locking behavior in a graphical clustered view. If the DLM



Topo	Name	V	S
▭	Preview_State	✓	✓
🟥	EX	✓	✓
▭	NL	✓	✓
🟩	PENDING	✓	✓
🟦	PR	✓	✓
●	Preview_Event	✓	✓
●	AST	✓	✓
●	AST_EAGAIN	✓	✓
●	AST_ECANCEL	✓	✓
●	AST_EDEADLK	✓	✓
●	BAST	✓	✓
●	CANCEL	✓	✓
●	LOCK	✓	✓
●	UNLOCK	✓	✓

Figure 3: Jumpshot legend window and various topologies representing DLM events and states

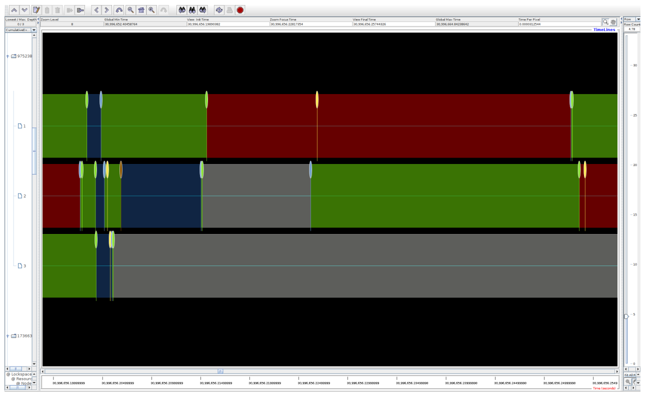


Figure 4: Jumpshot timeline window representing events and states

ViewMap is selected there exists a column separation on the Y-Axis that means a cluster-wide lock operation can be tracked during time. The user can zoom out and in and scroll the timeline.

Future Work

In this section we will show possible future work what we can do with such time synchronized tracing feature in general or DLM specific.

Adaptation to Distributed Networks

This paper shows how the DLM network protocol was used to present their use-case in a graphical way. The reason why this is possible are global networking wide exchanged identifier such as lockspace ID and resource name for entities like locks. The Linux tracing subsystem and slog2sdk offers a very generic API that other distributed network protocols can be adapted to a similar adaptation.

Cluster Kernel Application Trace Events

Additional to the DLM lock states we should show application events from the DLM user to give the user the possibility to know what the application is doing under certain lock states. We probably can add another column in the DLM ViewMap to provide those as a one-to-one mapping from Linux tracing events and slog2 events.

Contention States

To debug lock behavior we should introduce a state to show if a lock is in contention state. If lock modes are incompatible to each other in a lock conversion from one mode to another the lock state could be blocked e.g. switching from any lock mode to exclusive. This can be useful for the application developer to show lock contention and if possible to avoid them.

Internal DLM cluster communication

We could use the arrow topology of slog2 to show DLM communication within cluster nodes. This will help to debug DLM network communication.

Continuous Integration Testing

We can use Linux time synchronized DLM traces to validate DLM behavior. DLM has compatible locking modes, we can check if incompatible locking modes are used in the cluster by any node at the same time. If so this behavior should be debugged for a possible issue in DLM. Such test can be integrated into the CKI project [4].

User Space Tracing

Currently, the Linux Tracing subsystem has a pending feature request to introduce user space tracing functionality. With user space tracing functionality we can use the same approach used in this paper for user space applications. It's not known if time synchronization works in this environment as well and is one interested point to try out.

Runtime Kernel Optimization

Another non-debugging idea is to collect locking stats by using Linux tracing subsystem. This can be used to change the DLM lock master to reduce network communication. However, if this really brings any advantages needs to be figured out.

Another optimization would be to track network flow with skbmark values and changing qdisc behavior.

Live Tracing

Currently, we don't support any live tracing functionality yet. That means we always have trace files and convert them to generate a slog2 file. A stream like based input for DLM traces and a direct converting into slog2 on the fly could be done.

References

[1] Aring, A. dlm2slog2. https://gitlab.com/netcoder/dlm2slog2/-/wikis/home.

[2] Aring, A. fork of slog2sdk. https://gitlab.com/netcoder/slog2sdk.

[3] Aring, A. Java bindings for trace-cmd. https://gitlab.com/netcoder/trace-cmd-java.

[4] Authors, T. C. P. CKI Project Documentation. https://cki-project.org/.

[5] Chan, A.; Gropp, W.; and Lusk, E. Jumpshot-4 Users Guide. https://www.mcs.anl.gov/research/projects/perfvis/software/viewers/jumpshot-4/usersguide.html.

[6] Chan, A.; Gropp, W.; and Lusk, E. slog2sdk. https://www.mcs.anl.gov/research/projects/perfvis/download/index.htm#slog2sdk.

[7] Chan, A.; Gropp, W.; and Lusk, E. 2008. An efficient format for nearly constant-time access to arbitrary time intervals in large trace files. *Scientific Programming* 16(2-3):155–165.

[8] Corosync. Corosync - The Corosync Cluster Engine. https://corosync.github.io/corosync/.

[9] Linux Kernel Documentation. Global File System 2. https://www.kernel.org/doc/html/latest/filesystems/gfs2.html.

[10] Linux Kernel Documentation. OCFS2 filesystem. https://www.kernel.org/doc/html/latest/filesystems/ocfs2.html.

[11] SWIG. Simplified Wrapper and Interface Generator. https://www.swig.org/.

[12] Tracing, L. trace-cmd. https://trace-cmd.org/.

[13] Wikipedia. Distributed Lock Manager (DLM). https://en.wikipedia.org/wiki/Distributed_lock_manager.

[14] Wikipedia. GANTT chart. https://en.wikipedia.org/wiki/Gantt_chart.