DIAS-Logging-System Documentation v0.1

2018-11-27

- ☐ Show memlog

Introduction

The purpose of the *DIAS-Logging-System* is to provide appropriate logging to support the debugging and real-time monitoring of peer-to-peer applications such as *DIAS*.

Initially, logging within *DIAS* was performed using *log4j*, a classic logging tool widely used by Java developers that works well on simple applications but has major shortcomings in peer-to-peer applications. Already with 30 peers in a network, the basic logfiles become difficult to handle, for the following reasons:

- 1. Multiplicity of log files. Since there is one logfile per peer, it is inconvenient to browse through N different logfiles
- 2. Requirement to write a multitude of Python and/or other scripts for post-processing logfiles
- 3. Difficulty to perform complex event processing to correlate data points generated within peers and amongst peers, throughout the network
- 4. Query times get long, especially as the size grow large when simulations run over several weeks
- 5. It is not guaranteed to have a decent network file system (such as NFS) that can support heavy writing. Such was the case on the ETH Euler supercomputer

Architecture

Thus we designed the *DIAS-Logging-System* around a single PostgreSQL database with the following four objectives:

- 1. Simplified analysis
    - Leverage the powerful SQL language, rather than ad-hoc scripts such as Python and bash to post-process logfiles
    - Using SQL, it becomes easy to understand interactions within components of a peer and between peers, using built-in JOIN operations. This also paves the way for complex event processing, where different types in different parts of the system can be easily brought together for correlation analysis
    - Enable time-series analysis, e.g to observe the evolution of aggregates or memory usage over time

- 2. Support for multiple peers on multiple hosts
    - With peers located across on different servers, possibly on different networks of the Internet, the logging data should be stored in a single database, accessible from all peers
    - Even when peers are on the same private network with a common shared file stotrage, NFS may not provide the necessary throughput to sustain

heavy logging from many peers

- 3. Lightweight, simple to use and efficient
    - Minimise impact on real-time applications
    - Global static
    - Simple drop-copy replacement for System.out.print statements

- 4. Scale and real-time
    - Scale to N peers running over periods of weeks and months
    - Fast access to the data, using database tuning techniques such as partitioning and indexing
    - The ability to perform analysis on the logging in real-time, and to allow for dashboards
    - fast write speeds several K msgs /sec

With these objectives in mind, *DIAS-Logging-System* was designed using the following code components.

- A single PostgreSQL database, containing all the logging data from all peers
    - using database tuning techniques such as partitioning and indexing
- A single persistence demaon, accessible by all peers in the network, that listens to ZeroMQ messages with logging information sent by the peers
    - simple Single location for the data -> single connection to database, commit rate
    - The daemon listens to logging information on a pull port, and persists all received messages to the PostgreSQL database
    - Fast write speeds supporting several thousand msgs /sec sustained over time
- The users ionly Simple logging functions, with the same behavior as basic printing functions such ZeroMQ messaging
    - RawLog
    - EventLog
    - MemLog

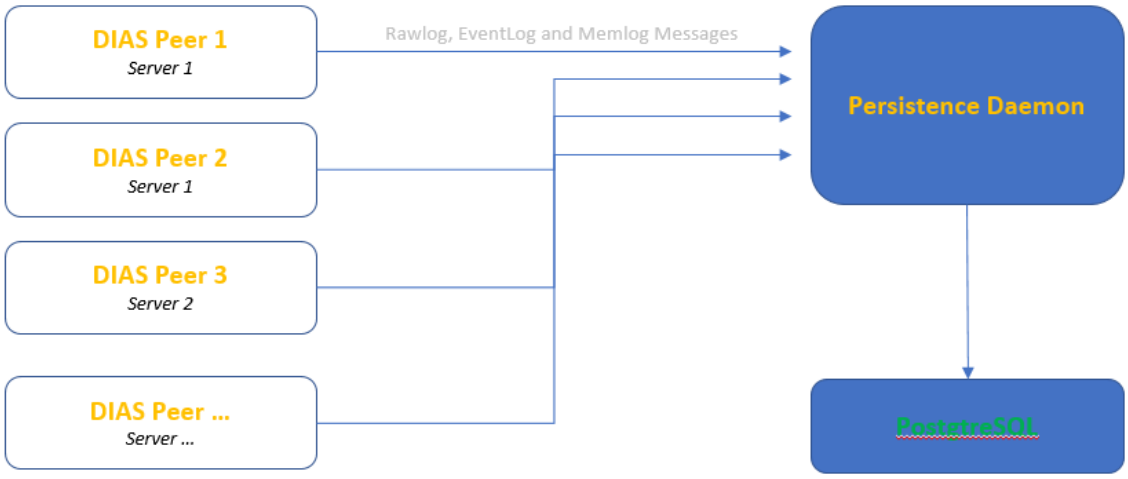

Figure {}. Design of the DIAS-Logging-System

Code examples

Currently there are three supported mechanisms for logging data

- Rawlog, free-form logging, used as a replacement to System.out.print
- Memlog, used to sample memory footprint of Java objects at regular intervals
- Eventlog, used to understand event sequences within peers and/or amongst sets of peers

1. RawLog

- *RawLog* is designed as a drop-copy replacement for System.out.printf
- The function takes 2 arguments
  - The log level, a number between 1 and 3 representing respectively an information, a warning or an error message
  - Any string to be persisted
- The *EventLog* automatically records the id of the calling thread allowing insights into potential issues regarding thread synchronisation

```
System.out.printf( "DIASlight created\n" );
```

```
RawLog.print(1,"DIASLight::SendBootstrapHello");
```

Figure {}. Java code to add a free-form logging message to the DIAS-Logging-System. The number '1' indicates level info

2. EventLog

- *EventLog* is designed to understand code execution sequence within a peer and across multiple peers
- It is different than *RawLog* in that it is structured, allowing for side-by-side comparison of events within and across peers
- Furthermore, the *EventLog* automatically records the id of the calling thread allowing insights into potential issues regarding thread synchronisation
- The function takes 4 arguments
    - The name of the calling class, e.g DIAS
    - The name of the calling function, e.g start
    - A tag or a variable
    - An optional value

```
EventLog.logEvent("DIAS", "start", "alreadyStarted",
Boolean.toString(alreadyStarted)  );
```

3. MemLog

- MemLog is designed to periodically measure total memory footprint of any Java object in memory
- It's great for detecting memory leaks that appear over time, e.g over several weeks of operation
- As explained below, MemLog uses the JAMM instrumentation agent, that traverses all elements and member variables of complex Java objects such as classes and containers
- Objects where memory monitoring is required need to be added with a call to add_object, that takes 3 arguments
    - Arguments one and two are simple tags that will allow you
        - In the example below, we are using the class in which the object belongs and the name of the instantiated object
    - The third argument is a reference to the Java object to be monitored

```
// memory logging
MemLog.add_object("DIASlight", "dumper", this.dumper);
```

Figure {}. Java code to add an object the MemLog. In this example, the instance of a protopeer.MeasurementFileDumper inside the DIASlight peerlet.

Querying the data

Since the data is stored in a PostgreSQL database with fully compliant SQL support, it is straightforward to examine the logs as well as perform complex search, join and analytics.

1. RawLog
1.1 Most recent messages

- Shows the 100 most recent *RawLog* messages, across all peers
- A powerful feature of the DIAS-Logging-System is the ability to observe log events across peers within a same window, as can be seen in the example below.

```
SELECT * FROM rawlog ORDER BY seq_id DESC LIMIT 100;
```

	seq_id	dt	network	peer	epoch	thread_id	error_level	txt
1	1982596336	2018-11-21 16:54:01.0	1	11	1542815641	14	1	Disseminator: openSession 9d42711b-f87e-472e-9ecc-73f26029357c -> tcp://178.63.43.201:57642
2	1982596335	2018-11-21 16:54:01.0	1	11	1542815641	14	3	Disseminator::openSession: session with 9d42711b-f87e-472e-9ecc-73f26029357c already found
3	1982596334	2018-11-21 16:54:01.0	1	10	1542815641	14	1	Disseminator: openSession 9d42711b-f87e-472e-9ecc-73f26029357c -> tcp://178.63.43.201:57642
4	1982596333	2018-11-21 16:54:01.0	1	10	1542815641	14	3	Disseminator::openSession: session with 9d42711b-f87e-472e-9ecc-73f26029357c already found
5	1982596332	2018-11-21 16:54:01.0	1	5	1542815641	14	1	Disseminator: openSession 9d42711b-f87e-472e-9ecc-73f26029357c -> tcp://178.63.43.201:57642
6	1982596331	2018-11-21 16:54:01.0	1	5	1542815641	14	3	Disseminator::openSession: session with 9d42711b-f87e-472e-9ecc-73f26029357c already found
7	1982596330	2018-11-21 16:54:01.0	1	7	1542815641	22	1	ok
8	1982596329	2018-11-21 16:54:01.0	1	7	1542815641	22	1	sending...
9	1982596328	2018-11-21 16:54:01.0	1	7	1542815641	22	1	response: {"ackText":"OK","errorText":"","acknowledgedMessageType":"Heartbeat","timeStamp":"2018-11-21 1
10	1982596327	2018-11-21 16:54:01.0	1	7	1542815641	22	1	Heartbeat received, responding with ACK_
11	1982596326	2018-11-21 16:54:01.0	1	7	1542815641	22	1	checkPairing : true
12	1982596325	2018-11-21 16:54:01.0	1	7	1542815641	22	1	incomingMessageType : Heartbeat

Figure {}. Latest 100 *RawLog* messages, across all peers

1.1 Only messages of a given peer, that contain certain text

```
SELECT * FROM rawlog WHERE peer = 13 AND txt LIKE 'LEAVE REQUEST IS TRUE!'
ORDER BY seq_id DESC ;
```

	seq_id	dt	network	peer	epoch	thread_id	error_level	txt
1	1982594042	2018-11-21 16:53:42.0	1	13	1542815622	14	1	LEAVE REQUEST IS TRUE!
2	1982586010	2018-11-21 16:52:21.0	1	13	1542815541	14	1	LEAVE REQUEST IS TRUE!

Figure {}. Searching for messages within a given peer that contain specific text

1.2 Warning and error messages

- The rawlog table is partitionned by error_level for fast lookup

```
SELECT * FROM rawlog WHERE error_level >= 2  ORDER BY seq_id DESC LIMIT
100;
```

	seq_id	dt	network	peer	epoch	thread_id	error_level	txt
1	1982596335	2018-11-21 16:54:01.0	1	11	1542815641	14	3	Disseminator::openSession: session with 9d42711b-f87e-472e-9ecc-73f26029357c already found
2	1982596333	2018-11-21 16:54:01.0	1	10	1542815641	14	3	Disseminator::openSession: session with 9d42711b-f87e-472e-9ecc-73f26029357c already found
3	1982596331	2018-11-21 16:54:01.0	1	5	1542815641	14	3	Disseminator::openSession: session with 9d42711b-f87e-472e-9ecc-73f26029357c already found
4	1982596264	2018-11-21 16:54:01.0	1	7	1542815641	14	3	Disseminator::openSession: session with 9d42711b-f87e-472e-9ecc-73f26029357c already found
5	1982596229	2018-11-21 16:54:01.0	1	8	1542815641	19	3	Disseminator::closeSession: no session with 99f9c5db-87c0-4f2f-afdf-2f73465a1269 found
6	1982596167	2018-11-21 16:54:01.0	1	8	1542815641	14	3	Disseminator::openSession: session with 9d42711b-f87e-472e-9ecc-73f26029357c already found
7	1982596155	2018-11-21 16:54:01.0	1	8	1542815641	14	3	Disseminator::openSession: session with 99f9c5db-87c0-4f2f-afdf-2f73465a1269 already found
8	1982596138	2018-11-21 16:54:00.0	1	9	1542815640	14	3	Disseminator::openSession: session with 9d42711b-f87e-472e-9ecc-73f26029357c already found
9	1982596136	2018-11-21 16:54:00.0	1	3	1542815640	14	3	Disseminator::openSession: session with 9d42711b-f87e-472e-9ecc-73f26029357c already found
10	1982596134	2018-11-21 16:54:00.0	1	10	1542815640	14	3	Disseminator::openSession: session with 9d42711b-f87e-472e-9ecc-73f26029357c already found
11	1982596132	2018-11-21 16:54:00.0	1	7	1542815640	19	3	Disseminator::closeSession: no session with 65eaa7a0-8f03-456f-9890-8beae5f55ae1 found
12	1982596129	2018-11-21 16:54:00.0	1	7	1542815640	19	3	Disseminator::closeSession: no session with eb751e87-cefd-4b63-9eef-71c9cd78b890 found

Figure {}. Warning and error messages

2. EventLog

2.1 Summary of events logged

```
SELECT classname,func,key,COUNT(*) FROM eventlog GROUP BY
classname,func,key ORDER BY classname,func,key ASC;
```

	classname	func	key	count
1	Aggregator	receiveDisseminatorReport	case	864
2	DIAS	runActiveState	hasNewSelectedState	37
3	DIAS	runActiveState	leaveRequest	2
4	DIAS	runActiveState	possibleStateChanged	26
5	DIAS	start	alreadyStarted	21
6	DIAS	start	finger	21
7	DIAS	start	is.carrier	21
8	DIASlight	receiveLeaveMessage	migrations.size	1
9	DIASlight	receiveLeaveMessage	NumOpenSessions	1
10	DIASlight	receiveLeaveMessage	peer	1
11	DIASlight	returnHome	peer	1
12	DIASlight	start	finger	2
13	DIASlight	start	is.carrier	2

Figure {}. Summary of the types of events stored in the event log

2.2 Most recent events

- The events in the eventlog table are persisted with milli-second resolution timestamp, which should provide enough resolution for most debugging applications
- In the below example you can observe the *DIAS* aggregation process taking place amongst the different peers in the network

```
SELECT * FROM eventlog ORDER BY current_time_millis DESC LIMIT 200;
```

	seq_id	dt	current_time_millis	network	peer	epoch	peer_event_counter	thread_id	classname	func	key	value
1	219192900	2018-11-21 16:54:01.0	1542815641096	1	10	1542815641	96	19	Aggregator	receiveDisseminatorReport	case	5
2	219192899	2018-11-21 16:54:01.0	1542815641095	1	10	1542815641	95	19	Aggregator	receiveDisseminatorReport	case	1
3	219192898	2018-11-21 16:54:01.0	1542815641095	1	7	1542815641	110	19	Aggregator	receiveDisseminatorReport	case	5
4	219192897	2018-11-21 16:54:01.0	1542815641095	1	7	1542815641	109	19	Aggregator	receiveDisseminatorReport	case	1
5	219192896	2018-11-21 16:54:01.0	1542815641093	1	4	1542815641	82	19	Aggregator	receiveDisseminatorReport	case	5
6	219192895	2018-11-21 16:54:01.0	1542815641093	1	4	1542815641	81	19	Aggregator	receiveDisseminatorReport	case	1
7	219192893	2018-11-21 16:54:01.0	1542815641089	1	3	1542815641	81	19	Aggregator	receiveDisseminatorReport	case	1
8	219192894	2018-11-21 16:54:01.0	1542815641089	1	3	1542815641	82	19	Aggregator	receiveDisseminatorReport	case	5
9	219192891	2018-11-21 16:54:01.0	1542815641087	1	9	1542815641	87	19	Aggregator	receiveDisseminatorReport	case	1

Figure {}. Most recent events in the event log

2.3 Tracking certain events

- In the below example, we observe the finger event. When each peer is started, it writes it's IP address and port to the *EventLog*
- This allows for simple retrieval of the fingers of each peer

```
SELECT * FROM eventlog WHERE key = 'finger' ORDER BY epoch ASC LIMIT 200;
```

	seq_id	dt	current_time_millis	network	peer	epoch	peer_event_counter	thread_id	classname	func	key	value
1	219191902	2018-11-21 16:50:31.0	1542815431623	1	3	1542815431	2	1	DIAS	start	finger	tcp://178.63.43.201:52080
2	219191905	2018-11-21 16:50:32.0	1542815432624	1	4	1542815432	2	1	DIAS	start	finger	tcp://178.63.43.201:57006
3	219191908	2018-11-21 16:50:33.0	1542815433681	1	5	1542815433	2	1	DIAS	start	finger	tcp://178.63.43.201:49726
4	219191910	2018-11-21 16:50:34.0	1542815434189	1	1	1542815434	1	1	DIASlight	start	finger	tcp://178.63.43.201:55113
5	219191913	2018-11-21 16:50:34.0	1542815434622	1	6	1542815434	2	1	DIAS	start	finger	tcp://178.63.43.201:60779
6	219191915	2018-11-21 16:50:35.0	1542815435295	1	2	1542815435	1	1	DIASlight	start	finger	tcp://178.63.43.201:53262
7	219191918	2018-11-21 16:50:35.0	1542815435555	1	7	1542815435	2	1	DIAS	start	finger	tcp://178.63.43.201:54486
8	219191921	2018-11-21 16:50:36.0	1542815436670	1	8	1542815436	2	1	DIAS	start	finger	tcp://178.63.43.201:58386
9	219191924	2018-11-21 16:50:37.0	1542815437761	1	9	1542815437	2	1	DIAS	start	finger	tcp://178.63.43.201:63589
10	219191927	2018-11-21 16:50:38.0	1542815438814	1	10	1542815438	2	1	DIAS	start	finger	tcp://178.63.43.201:62553
11	219191930	2018-11-21 16:50:39.0	1542815439850	1	11	1542815439	2	1	DIAS	start	finger	tcp://178.63.43.201:53265

Figure {}. Filtering on the event type (column 'key')

3. MemLog

3.1 Show the latest memory footprint for recorded objects for each peer

```
WITH with_last_peer_seq_id AS
(
    SELECT
        MAX(seq_id) AS last_peer_seq_id
        ,peer
        ,object_group_name
        ,object_name
    FROM
        memlog
    GROUP BY
        peer
        ,object_group_name
        ,object_name
)
/* SELECT * FROM with_last_peer_seq_id ORDER BY peer ASC; */
SELECT
    mem.*
FROM

    memlog mem
INNER JOIN
    with_last_peer_seq_id last_records
    ON
    last_records.last_peer_seq_id = mem.seq_id
--WHERE
--    mem.object_size_mb > 0.1
ORDER BY
    peer ASC
    ,object_size_mb DESC NULLS LAST
```

☐ Show memlog

3.2 Show the memory footprint of all recorded objects for a single peer, as a timeseries

```
SELECT
    *
FROM
    memlog
WHERE
    peer = 1
ORDER BY seq_id ASC;
```

- [ ] Show memlog

Using *DIAS-Logging-System* in your application

Before the *DIAS-Logging-System* can be used, it needs to be initalised inside each Java program.

1. Persistence Client

- The *Persistence Client* is required by the *RawLog, MemLog and EventLog* classes to send messages to be persisted to the *Persistence Daemon*
- The advantage of this design is that the *Persistence Client* and *Persistence Daemon* could be modified to store the data in other types of databases, without affecting the *RawLog, MemLog* or *EventLog* classes

```
import pgpersist.PersistenceClient;

/* ... */

persistenceClient = new PersistenceClient( zmqContext, daemonConnectString,
persistenceClientOutputQueueSize );
System.out.println( "persistenceClient created" );
```

Figure {}. Java code to initialise the PersistenceClient instance

2. RawLog

- The *RawLog* class is initialised with the persistence client that actually sends the messages to the *Persistence Daemon*
- Next, you can set the threshold level for logging. Default would be 2 and above, i.e only persist warnings and errors
    - Set this value to 1 to log all calls to *RawLog*
- Optionally, you specify the the peer in which your code is running
    - This allows RawLog to automatically add additional information to the logging, such as:
        - Peer ID
        - Epoch

- Optionally, you specify the the *DIAS Network Id*, which appears as a column in the rawlog table and allows distinguishing between different DIAS networks running on the same infrastructure

```
RawLog.setPeristenceClient(persistenceClient);

RawLog.setErrorLevelThreshold(rawLogLevel);        // log all (1); only log
warnings (2); errors (3)
RawLog.setPeer(newPeer);
RawLog.setDIASNetworkId(diasNetworkId);

System.out.printf( "RawLog setup with rawLogLevel %d\n", rawLogLevel );
```

Figure {}. Java code to initialise the RawLog class

3. EventLog

- The *EventLog* class is initialised similarly to the RawLog with a:
    - *Persistence client* that actually sends the messages to the *Persistence Daemon*
    - Optionally, you specify the the peer in which your code is running
    - Optionally, you specify the the *DIAS Network Id*, which appears as a column in the eventlog table and allows distinguishing between different DIAS networks running on the same infrastructure

```
EventLog.setPeristenceClient(persistenceClient);
EventLog.setPeer(newPeer);
EventLog.setDIASNetworkId(diasNetworkId);
```

Figure {}. Java code to initialise the EventLog class

4. MemLog

- The *MemLog* class is initialised similarly to the *RawLog* and *EventLog* classes above, with some notable differences
- First, a Java instrumentation class, that will take the actual memory measurements, is required for the *MemLog* and must be passed as an argument to *Memlog*. More information about Java instrumentation can be found at Oracle (see references below)
    - The particular instrumentation class used is JAMM (see references below). JAMM is a powerful Java instrumentation class that mesures the size of objects in memory. For containers, JAMM actually traverses the containers and measures the size of each element, further traversing each element if it is a container. This recursive approach allows proper estimation of the memory footprint of an object.
- Second, the period for taking snapshots
    - *Memlog* will periodically compute the memory footprint of each object that

was added with *add_object()*, at the period specificed in *startTimer()*

```
// initialise MemLog
final MemoryMeter     instrument = new MemoryMeter();

if( !MemoryMeter.hasInstrumentation() )
    System.out.printf( "MemLog: No instrumentation\n" );
else
{
    MemLog.setPersistenceClient(persistenceClient);
    MemLog.setMeasurementInstrument(instrument);
    MemLog.setPeer(newPeer);
    MemLog.setDIASNetworkId(diasNetworkId);
    MemLog.startTimer(60);

    System.out.printf( "MemLog setup\n" );

    MemLog.add_object("Peer", "Peer", newPeer);
}
```

Figure {}. Java code to initialise the MemLog class

Database Installation

Installation of the database, that contains the output of the logging, is simple and is done in three steps. The *DIAS-Logging-System* requries a running PostgreSQL database, that we will install in the first step just below.

1.Install postgres database

- Note that this will also start the database service

```
# on an Ubuntu system (Ubuntu 14 or higher)
sudo apt-get update
sudo apt-get install postgresql
```

Figure {}. Ubuntu commands to install the latest version of PostgreSQL server

2.Create a database

- Login into PostgreSQL, using your favorite SQL editor or *psql*
- Choose any name, in this case we choose the name *dias*
    - If you change the database name, you will also need to update the file conf/daemon.conf accordingly

```
CREATE DATABASE dias;
```

Figure {}. SQL commands to create the persistence database

3. Finally, create the tables that will store the logging information

- There are three tables to create
  - eventlog.sql
  - memlog.sql
  - rawlog.sql
- The source SQL can be found here: DIAS-Logging-System/sql/definitions
- Basic database optimisations such as table partitioning and indexing are provided out of the box

```
-- copy + paste the SQL in the above mentionned files to create the tables
CREATE TABLE eventlog ...
CREATE TABLE memlog ...
CREATE TABLE rawlog ...
```

Figure {}. SQL commands to create the persistence tables

Launch

Once the *DIAS-Logging-System* is installed, launch the *persistence daemon (*that listens for messages to be logged and writes them to the database).

```
cd DIAS-Logging-System
./start.daemon.sh deployments/localhost
```

Figure {}. Commands to launch of the persistence daemon

This will launch a *UNIX screen* session, inside of which the *persistence daemon* will run. In the screenshot below zou can see here that the daemon is listening on localhost:5433 for logging messages to be persisted

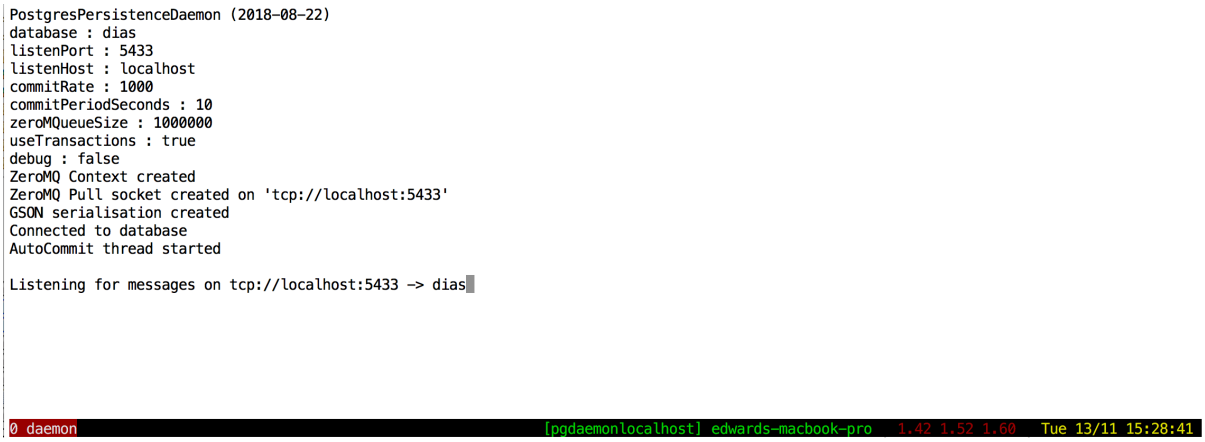

Figure {}. Screenshot of the running DIAS-Logging-System persistence daemon

References

- Java Instrumentation: https://docs.oracle.com/javase/8/docs/api/index.html?java/lang/instrument/Instrumentation.html
- JAMM Memory Instrumentation: https://github.com/jbellis/jamm