

# MariaDB ash\_sampler tool

By: Karl Arao

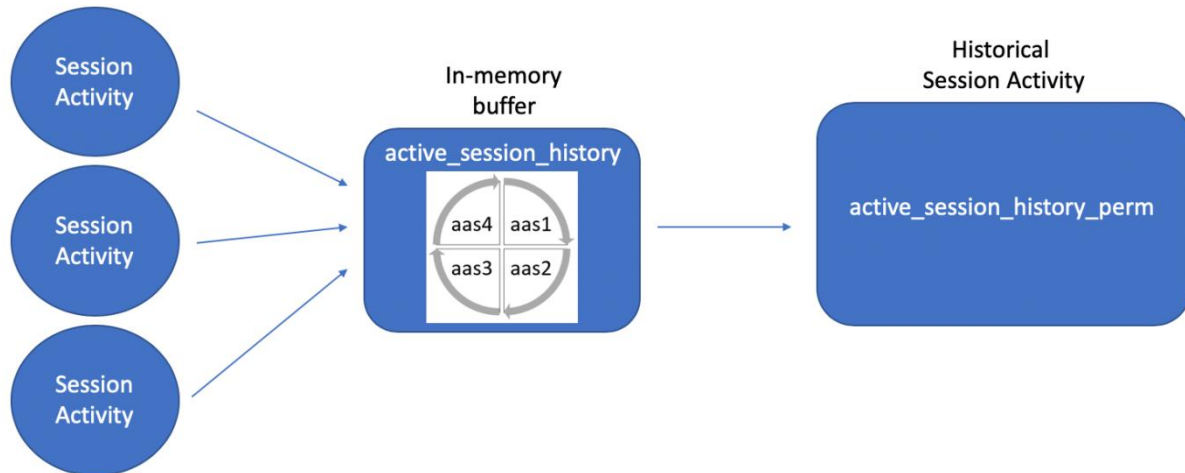
## Table of Contents

- THE ASH\_SAMPLER TOOL .....2
- TOOL REQUIREMENTS .....3
- STEP 1) DOWNLOAD AND INSTALL THE TOOL .....3
- STEP 2) START THE ASH\_SAMPLER .....4
- STEP 3) STOP THE ASH\_SAMPLER .....4
- STEP 4) EXPORT THE DATA .....4
- STEP 5) VISUALIZE AND CORRELATE WITH CLOUDWATCHER .....5
  - SYSTEM METRICS USING CLOUDWATCHER.....6
  - WORKLOAD DRILL DOWN USING ASH\_SAMPLER DATA .....8
    - by current sql and thread state .....9
    - state by sql and long running sqls .....9
    - deadlock sessions .....10

## The ash\_sampler tool

This tool takes a snapshot of an active session activity with 1 second granularity. The data is initially gathered in a circular buffer 4 table memory storage engine. And once each buffer gets filled the data is de-staged to a longer-term permanent table for historical reporting.

This concept is similar to Oracle's v\$active\_session\_history (in-memory) and dba\_hist\_active\_sess\_history (longer term) performance views and also AWS RDS Performance Insights (only available on RDS MariaDB 10.2.21 and higher) which are used for troubleshooting complex performance problems.



Each session sample is 1 row and the underlying data is divided into two dimensions and measures. This allows for data drilldowns and different ways of aggregating the measures and slicing the dimensions. This detailed data is useful for correlating the database activity across sessions and also with system metrics (CPU, memory, IO, network, etc.)

Time dimension	Session dimension	Measures
<ul style="list-style-type: none"><li>• snap_time</li></ul>	<ul style="list-style-type: none"><li>• thd_id</li><li>• conn_id</li><li>• user</li><li>• db</li><li>• command</li><li>• state</li><li>• statement_digest</li><li>• current_statement</li><li>• full_scan</li><li>• last_statement_digest</li><li>• last_statement</li><li>• last_wait</li><li>• source</li><li>• pid</li><li>• program_name</li></ul>	<ul style="list-style-type: none"><li>• time</li><li>• statement_latency</li><li>• lock_latency</li><li>• rows_examined</li><li>• rows_sent</li><li>• rows_affected</li><li>• tmp_tables</li><li>• tmp_disk_tables</li><li>• last_statement_latency</li><li>• last_wait_latency</li></ul>

## Tool Requirements

- The tool needs to have the `performance_schema` turned on. On AWS RDS parameter groups set the `performance_schema` to 1.
- Execute the command below to verify the `performance_schema`

```
mysql> SHOW GLOBAL VARIABLES LIKE 'performance_schema';
Variable_name  Value
performance_schema  ON
```

## Step 1) Download and Install the tool

- Download the tool on your laptop at [https://github.com/karlarao/ash\\_sampler-mariadb](https://github.com/karlarao/ash_sampler-mariadb)

```
wget https://github.com/karlarao/ash_sampler-mariadb/archive/master.zip

$ unzip master.zip
Archive:  master.zip
2e30d93a7a91c6650ef4f3994dba0773cd7b56b6
   creating: ash_sampler-mariadb-master/
  inflating: ash_sampler-mariadb-master/01_create_objects.sql
  inflating: ash_sampler-mariadb-master/02_ash_sampler.sql
  inflating: ash_sampler-mariadb-master/LICENSE
  inflating: ash_sampler-mariadb-master/README.md
```

- On your text editor replace “mariadb2” with the username that you'll use for monitoring or use the `sed` commands below

```
sed 's/mariadb2/your_username_here/g' 01_create_objects.sql.bak > 01.sql
sed 's/mariadb2/your_username_here/g' 02_ash_sampler.sql > 02.sql
```

- Open the file `01_create_objects.sql` and execute the commands using the monitoring user
- Download and install DBeaver Community Edition at <https://dbeaver.io/download/>
- Open the file `02_ash_sampler.sql` using DBeaver and execute the commands using the monitoring user

## Step 2) Start the ash\_sampler

- Connect to your MariaDB AWS database and run the ash\_sampler
- The recommended input parameters (1,1,16) means infinite gathering, for every 1 sec, and create 16MB in-memory buffers

```
$ mysql -s -h mariadb2.us-east-1.rds.amazonaws.com --port=3306 -u mariadb2
-p<your_password_here> -D mariadb2
mysql: [Warning] Using a password on the command line interface can be
insecure.
mysql>
mysql> call ash_sampler(1,1,16);
```

## Step 3) Stop the ash\_sampler

- To stop the collection CTRL-C or kill the session running the ash\_sampler

```
mysql> show processlist;
+----+-----+-----+-----+-----+-----+-----+-----+-----+-----+
Id    User      Host                                     db    Command Time    State    Info    Progress
+----+-----+-----+-----+-----+-----+-----+-----+-----+-----+
4     rdsadmin  localhost:11948                         mysql Sleep    1        NULL    0.000
121   mariadb2  cpe-69-200-224-180.nyc.res.rr.com:53421 mariadb2 Sleep    16600    NULL    0.000
122   mariadb2  cpe-69-200-224-180.nyc.res.rr.com:53422 mariadb2 Sleep    16707    NULL    0.000
170   mariadb2  cpe-69-200-224-180.nyc.res.rr.com:57005 mariadb2 Sleep    9271     NULL    0.000
200   mariadb2  cpe-69-200-224-180.nyc.res.rr.com:59937 mariadb2 Query    429      User sleep SELECT SLEEP(
NAME_CONST('in_interval',1.00)) INTO @sleep 0.000
202   mariadb2  cpe-69-200-224-180.nyc.res.rr.com:60092 mariadb2 Query    0        init    show processlist 0.000

mysql> kill 200;
```

## Step 4) Export the data

- Using DBeaver Community Edition. Export the data from the following:
  - active\_session\_history
    - to export the recent in-memory data that is not yet de-staged to permanent storage
  - active\_session\_history\_perm
    - this contains the historical data since ash\_sampler is first started

NOTE: Whenever the ash\_sampler is restarted it clears the in-memory data so make sure to export the contents to CSV or put it in a table using CTAS before starting the ash\_sampler to not lose any performance data.

```
-- on DBeaver execute the two SQLs and export to CSV
select * from active_session_history;
select * from active_session_history_perm;
```

Follow the screenshots below:

### 1) Export data

The screenshot shows a SQL query editor with the following query:

```
select * from mariadb2.active_session_history;  
select * from mariadb2.active_session_history_perm;
```

Below the query, a table titled 'active\_session\_history\_perm' is displayed. The table has columns: snap\_time, thd\_id, conn\_id, and user. The first row is selected, and a context menu is open over it, showing options like Cut, Copy, Paste, Advanced Copy, Advanced Paste, Edit, Filter/Order, View/Format, Navigate, Layout, Export data ..., Open With, Generate SQL, and Refresh.

	snap_time	thd_id	conn_id	user
1	2019-07-20 07:56:30	29	7	mariadb2@
2	2019-07-20 07:56:30	29	7	mariadb2@
3	2019-07-20 07:56:30	29	7	mariadb2@
4	2019-07-20 07:56:30	29	7	mariadb2@
5	2019-07-20 07:56:30	29	7	mariadb2@
6	2019-07-20 07:56:30	29	7	mariadb2@
7	2019-07-20 07:56:30	29	7	mariadb2@
8	2019-07-20 07:56:30	29	7	mariadb2@
9	2019-07-20 07:56:30	29	7	mariadb2@
10	2019-07-20 07:56:30	29	7	mariadb2@
11	2019-07-20 07:56:30	29	7	mariadb2@
12	2019-07-20 07:56:30	29	7	mariadb2@
13	2019-07-20 07:56:30	29	7	mariadb2@
14	2019-07-20 07:56:30	29	7	mariadb2@
15	2019-07-20 07:56:30	29	7	mariadb2@
16	2019-07-20 07:56:30	29	7	mariadb2@
17	2019-07-20 07:56:30	29	7	mariadb2@
18	2019-07-20 07:56:30	29	7	mariadb2@
19	2019-07-20 07:56:49	29	7	mariadb2@

### 2) Select open new connection

The screenshot shows the 'Settings (Table to Files, CSV)' dialog box in the Data Transfer tool. The 'Extraction settings' section is expanded, showing 'Maximum threads: 1', 'Extract type: Single query', and 'Fetch size: 10000'. The 'Format settings' section is also expanded, showing 'Formatting: <Connection's default>', 'Binaries: Set to NULL', and 'Exporter settings'.

**Extraction settings**

- Maximum threads: 1
- Extract type: Single query
- ☒ Open new connection(s)
- ☐ Select row count
- Fetch size: 10000
- ☐ Selected columns only
- ☐ Selected rows only

**Format settings**

General

- Formatting: <Connection's default>
- Binaries: Set to NULL

**Exporter settings**

Name	Value
File extension	csv
Delimiter	,
Header	top
Characters escape	quotes
Quote character	"
Quote always	<input type="checkbox"/>
Quote never	<input type="checkbox"/>
NULL string	<input type="checkbox"/>
Format numbers	<input type="checkbox"/>

### 3) Set file name, Click Next

The screenshot shows the 'Output' dialog box in the Data Transfer tool. The 'General' section is expanded, showing 'Directory: /Users/kristofferson.a.arao/', 'File name pattern: ash\_mariadb2\_export', 'Encoding: UTF-8', and 'Insert BOM' checked. The 'Results' section is also expanded, showing 'Open output folder at end', 'Execute process on finish', and 'Show finish message' checked.

**General**

- ☐ Copy to clipboard
- Directory: /Users/kristofferson.a.arao/
- File name pattern: ash\_mariadb2\_export
- Encoding: UTF-8
- ☒ Insert BOM
- ☐ Write to the single file
- ☐ Compress
- ☐ Split output file Maximum file size: 10000000

**Results**

- ☐ Open output folder at end
- ☐ Execute process on finish
- ☒ Show finish message

### 4) Click Finish

The screenshot shows the 'Confirm' dialog box in the Data Transfer tool. The 'Objects' section is expanded, showing a table with columns: Source Container, Source name, Target Container, and Target name. The 'Source settings' and 'Target settings' sections are also expanded, showing various configuration options.

**Objects**

Source Container	Source name	Target Container	Target name
MariaDB - aws m...	select * from m...	/Users/kristofferson.a.arao/	ash_mariadb2_ex...

**Source settings**

Table settings:

- Open new connection(s): Yes
- Extract type: SINGLE\_QUERY
- Select row count: No
- Selected rows only: No
- Selected columns only: No

**Target settings**

Files settings:

- Write to the single file: No
- Directory: /Users/kristofferson.a.arao/
- File name pattern: ash\_mariadb2\_expor
- Encoding: UTF-8
- Insert BOM: Yes
- Compress: No
- Binaries: SKIP
- Encoding: HEX

## Step 5) Visualize and correlate with Cloudwatcher

The result is a CSV data set which can be visualized and analyzed in Tableau

```
$ head ash_mariadb2_export.csv
"snap_time","thd_id","conn_id","user","db","command","state","time","statement_digest","current_statement","statement_latency","lock_latency","rows_examined","rows_sent","rows_affected","tmp_tables","tmp_disk_tables","full_scan","last_statement_digest","last_statement","last_statement_latency","last_wait","last_wait_latency","source","pid","program_name"
2019-07-20 07:56:30,29,7,mariadb2@cpe-69-200-224-180.nyc.res.rr.com,mariadb2,Query,User sleep,0,c527d205346af37ae7d52fb0747e4109,"SELECT a.table_schema, a.table_name, slee
p(1) FROM information_schema.tables a, information_schema.tables b where a.table_sc",494296746000,154000000,0,0,2,0,YE,,,,,,,,,"64594",mysql
2019-07-20 07:56:31,29,7,mariadb2@cpe-69-200-224-180.nyc.res.rr.com,mariadb2,Query,User sleep,1,c527d205346af37ae7d52fb0747e4109,"SELECT a.table_schema, a.table_name, slee
p(1) FROM information_schema.tables a, information_schema.tables b where a.table_sc",1497894054000,154000000,0,1,0,2,0,YE,,,,,,,,,"64594",mysql
2019-07-20 07:56:32,29,7,mariadb2@cpe-69-200-224-180.nyc.res.rr.com,mariadb2,Query,User sleep,2,c527d205346af37ae7d52fb0747e4109,"SELECT a.table_schema, a.table_name, slee
p(1) FROM information_schema.tables a, information_schema.tables b where a.table_sc",2501633044000,154000000,0,2,0,2,0,YE,,,,,,,,,"64594",mysql
2019-07-20 07:56:33,29,7,mariadb2@cpe-69-200-224-180.nyc.res.rr.com,mariadb2,Query,User sleep,3,c527d205346af37ae7d52fb0747e4109,"SELECT a.table_schema, a.table_name, slee
p(1) FROM information_schema.tables a, information_schema.tables b where a.table_sc",3506193113000,154000000,0,3,0,2,0,YE,,,,,,,,,"64594",mysql
2019-07-20 07:56:34,29,7,mariadb2@cpe-69-200-224-180.nyc.res.rr.com,mariadb2,Query,User sleep,4,c527d205346af37ae7d52fb0747e4109,"SELECT a.table_schema, a.table_name, slee
p(1) FROM information_schema.tables a, information_schema.tables b where a.table_sc",4510484168000,154000000,0,4,0,2,0,YE,,,,,,,,,"64594",mysql
2019-07-20 07:56:35,29,7,mariadb2@cpe-69-200-224-180.nyc.res.rr.com,mariadb2,Query,User sleep,5,c527d205346af37ae7d52fb0747e4109,"SELECT a.table_schema, a.table_name, slee
p(1) FROM information_schema.tables a, information_schema.tables b where a.table_sc",5514364478000,154000000,0,5,0,2,0,YE,,,,,,,,,"64594",mysql
2019-07-20 07:56:36,29,7,mariadb2@cpe-69-200-224-180.nyc.res.rr.com,mariadb2,Query,User sleep,6,c527d205346af37ae7d52fb0747e4109,"SELECT a.table_schema, a.table_name, slee
p(1) FROM information_schema.tables a, information_schema.tables b where a.table_sc",6518152741000,154000000,0,6,0,2,0,YE,,,,,,,,,"64594",mysql
2019-07-20 07:56:37,29,7,mariadb2@cpe-69-200-224-180.nyc.res.rr.com,mariadb2,Query,User sleep,7,c527d205346af37ae7d52fb0747e4109,"SELECT a.table_schema, a.table_name, slee
p(1) FROM information_schema.tables a, information_schema.tables b where a.table_sc",7523094600000,154000000,0,7,0,2,0,YE,,,,,,,,,"64594",mysql
2019-07-20 07:56:39,29,7,mariadb2@cpe-69-200-224-180.nyc.res.rr.com,mariadb2,Query,User sleep,9,c527d205346af37ae7d52fb0747e4109,"SELECT a.table_schema, a.table_name, slee
p(1) FROM information_schema.tables a, information_schema.tables b where a.table_sc",8562704969000,154000000,0,8,0,2,0,YE,,,,,,,,,"64594",mysql
```

The environment used in this example is AWS RDS db.t2.micro with MariaDB 10.1.23 database.

The workload used are the following:

1) adhoc SQL that runs for 300 seconds

```
SELECT a.table_schema, a.table_name, sleep(1)
FROM information_schema.tables a, information_schema.tables b
where a.table_schema <> 'karl'
limit 300;
```

2) mysqlslap

```
$ mysqlslap -h mariadb2.c3yfiwlyerg0.us-east-1.rds.amazonaws.com --port=3306 -u
mariadb2 -pwelcome1 --concurrency=25 --iterations=10 --auto-generate-sql --number-of-
queries=100000
```

3) deadlock scenario

```
create table innodb_deadlock_maker(a int primary key) engine=innodb;
insert into innodb_deadlock_maker(a) values(0), (1);
-- connection 0
set transaction isolation level serializable;
start transaction;
select * from innodb_deadlock_maker where a = 0;
update innodb_deadlock_maker set a = 0 where a <> 0;
-- connection 1
set transaction isolation level serializable;
start transaction;
select * from innodb_deadlock_maker where a = 1;
update innodb_deadlock_maker set a = 1 where a <> 1;
-- session waits are: "Searching rows for update"
```

Below are some example screenshots:

System metrics using Cloudwatcher

- CPU Utilization peaked at 70%
- The example workload is heavy on Write IOPS and peaked at around 350-400 IOPS

aws Services Resource Groups				Karl Arao N. Virginia Support	
DB identifier mariadb2	CPU 37.33%	Info Available	Class db.t2.micro	Role	Region & AZ us-east-1c
Instance	Current activity 5 Connections	Engine MariaDB			

Connectivity & security Monitoring Logs & events Configuration Maintenance & backups Tags

CloudWatch (17)

Legend: mariadb2

Q



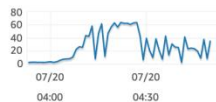
Add instance to compare

Monitoring

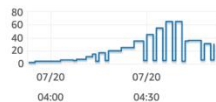
Last Hour

< 1 2 3 > ⚙

CPU Utilization (Percent)



DB Connections (Count)



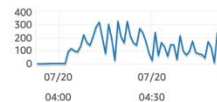
Free Storage Space (MB)



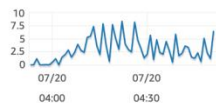
Freeable Memory (MB)



Write IOPS (Count/Second)



Read IOPS (Count/Second)



aws Services Resource Groups				Karl Arao N. Virginia Support	
DB identifier mariadb2	CPU 37.33%	Info Available	Class db.t2.micro	Role	Region & AZ us-east-1c
Instance	Current activity 5 Connections	Engine MariaDB			

Connectivity & security Monitoring Logs & events Configuration Maintenance & backups Tags

CloudWatch (17)

Legend: mariadb2

Q



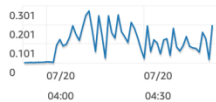
Add instance to compare

Monitoring

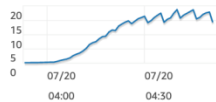
Last Hour

< 1 2 3 > ⚙

Queue Depth (Count)



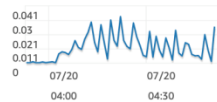
Binary Log Disk Usage (MB)



Write Throughput (MB/Second)



Read Throughput (MB/Second)

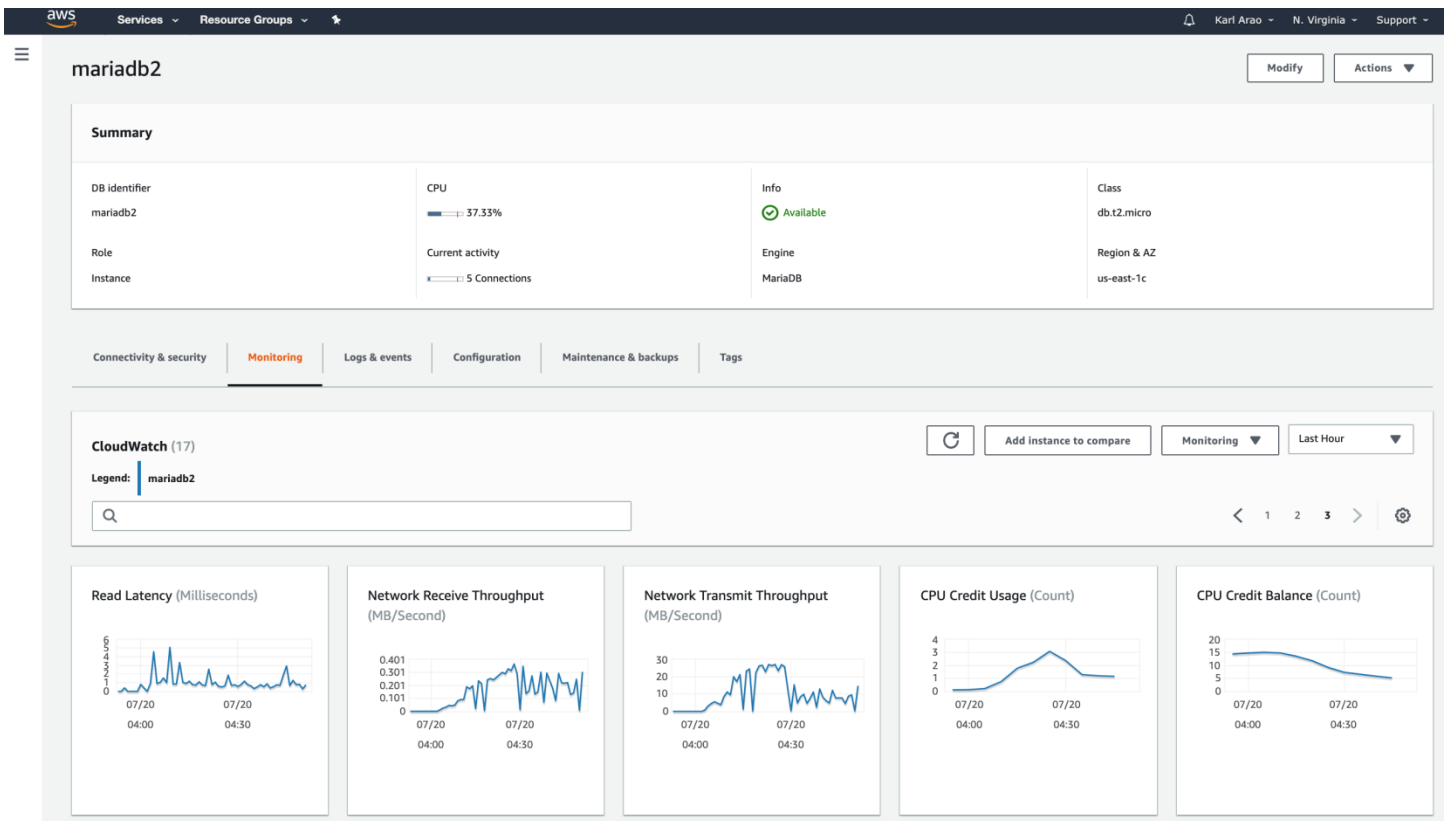


Swap Usage (MB)



Write Latency (Milliseconds)





## Workload drill down using ash\_sampler data

- Cloudwatcher system metrics correlated with ash\_sampler session data provides a lot of insights on solving complex performance problems.
- The ash\_sampler data can be sliced and diced using the available dimensions. And the five Ws can be asked/answered:
  - who is executing the SQL
  - what the application is doing
  - when is it doing that slow process
  - where most of the time is going
  - why are the sessions locking and causing deadlocks



The screenshot displays two performance metrics in Tableau: 'current sql' and 'thread state'. Both charts show data over time, with a significant peak around July 20. The 'current sql' chart has a tooltip showing a snapshot taken at 2019-07-20 08:34:16, with 26 active SQL statements. The current statement is 'SELECT intcol1,charcol1 FROM t1'. The 'thread state' chart shows the number of active threads, with a peak around July 20. The bottom of the image shows the Tableau interface with various tabs and a dashboard view.

The figure consists of two histograms side-by-side, comparing the distribution of states for 'state\_by sql' and 'long\_running sqls'. Both histograms have a y-axis labeled 'State' and an x-axis labeled 'Snap Time' with markers for 2019, Q3, July, and 20. The 'state\_by sql' histogram shows a distribution across states like Null, init, Opening tables, preparing, query end, Sending data, Unlocking tables, update, User sleep, and Waiting for table metadata. The 'long\_running sqls' histogram shows a similar distribution but with a significant peak in the 'Sending data' state. A legend on the right lists current statements such as 'SELECT a.table\_sche...', 'SELECT count(\*) fro...', 'SELECT intcol1,charc...', 'SELECT NAME, VALU...', 'SELECT value FROM ..', and 'SHOW GLOBAL VARI...'.

deadlock

