mysqlsla v2 Replays

This document is about working with mysqlsla v2 replays. A replay is a highly compact binary file representing the text data parsed from the MySQL log.

Replays can be saved at three different times during the operation of mysqlsla: **after parsing the log**, **after running any analyses** (**nth-percent**, **explain**, etc.), and **after sorting and pruning** the results.

Replays are later replayed with the **replay** option and allow log data to be re-filtered, re-analyzed, re-sorted, or merged with new text log data tremendeously faster than re-parsing the original text log.

mysqlsla v2 Replays
Table of Contents

Usefulness

DBAs rarely parse log files one single time and never again. Instead, they tend to re-parse logs for various reason: forgetting to set an option, seeing the results and then realizing that some SQL statements need to be filtered, or just simply wanting to see the results another way.

If the log file is small, re-parsing is no problem. But more and more I hear of people parsing multi-gig log files and that creates a problem: re-parsing huge log files over and over is a waste of both your time and the server's time. Replays fix this problem.

First of all, replays are highly compact. Let's take the example of a 700M log file that I use for testing. As a **post-parse replay** replay, it is compacted into 80M. The degree of compaction varies but generally the bigger the log the bigger the benefit. Such compaction is nothing magical; gzip does the same thing. Text files are inherently full of wasted space.

Secondly, replaying is much faster than re-parsing. Parsing my 700M log file takes 129 seconds from start to **standrd report**. Replaying the 80M replay takes only 9 seconds and produces the exact same standard report. Sure, 2 minutes faster is not blowing anyone out of their seat, but the difference is nonetheless remarkable.

"But wait," you say, "the text log has to be parsed at least once in order to produce the replay so we must still do it the slow way!" This is true but it misses the purpose of replays: to avoid *re*-parsing.

Limitations

« Top

Every replay is replayed from the start. This means that every replay starts with parsing and filtering. This is important because it allows replays to act like pseudo-logs: any replay can be re-meta-filtered, re-statement-filtered, re-grepped, re-analyzed, and re-sorted.

However, there are a few limitations. First of all, the replay saves only one sample of each unique query. Therefore, **grep** will not work well.

Second limitation: any meta-property noted as "Only for meta-property filter" in **mysqlsla v2 Filters** cannot be used as a filter condtion for a replay. These meta-properties are examined query-by-query for every query in the log but as we just noted: the replay saves only one sample of each unique query.

Third limitation: total users (all log-wide unique "user@host IP") does not work with replays. This is because total users is created directly from the log while parsing and saved in another data structure which is not saved with the replay and therefore cannot be re-created from the replay.

Fourth limitation: replays cannot be **merged** with other replays. Replays can only be merged with logs.

The final limitation is more like a side effect: there is no way to isolate subsets of queries inside a larger set in order to see the relation between the two. mysqlsla is designed to chop logs, to continually filter and remove queries in order to arrive at the smallest result. If you begin with a large, unfiltered replay and then create from it a smaller, more filtered replay, this 2nd replay is calculated only in relation to itself.

Let's take an example of this. Consider an unfiltered replay with 200 random queries. Then you filter that replay allowing only INSERT and UPDATE statements (statement-filter = "+INSERT,UPDATE"). And let's pretend that 50 INSERTs and 50 UPDATEs are found. The question is: in relation to what should these INSERTs and UPDATEs be calculated: the replay with 200 queries from which they came or the new result of 100 queries to which they belong? If the former then 50 INSERTs and 50 UPDATEs is each 25% of total queries (200). But if the later then each is 50% of total queries (100).

At present, mysqlsla calculates the new results in relation to themselves. A **future feature** may add an option to change this.

Post-Parse

« Top

The **post-parse replay** is saved immediately after parsing and filtering the log. If no meta or statement filters (or grep) are used, then the post-parse replay is identical to the original log. In this way, you can simply compact a log for later replaying.

If any filters are used, then the post-parse replay represents the log without the statements removed by the filters.

Post-Analyses

« Top

The **post-analyses replay** is saved after all extra query analyses have been performed. At present, this does not help much because I removed analysis filters from mysqlsla (they may be re-added later). It does help, however, if plan only to re-sort a result of queries which had been previously analyzed. Then, with a post-analyses replay, the queries can be re-sorted without having to be re-analyzed.

Post-Sort

« Top

The **post-sort replay** is saved after the queries have been sorted according to **sort** *and* pruned according to **top**. It is not possible to saved sorted but unpruned queries. If you want something like that, save a post-analyses replay.

This replay is very small: only **top** queries large.

If you want to create a super-compact representative sample of the original log, you could set top to 100 or 1000 or however many queries you think would make a good representative sample. Then you could re-play, re-filter and re-analyze this sample very quickly.

Merging Logs with Replays

« Top

Data from new logs can be merged with the data saved in a replay (of the same log type). This is similar to simply specifying multiple log files on the command line and parsing them all at once. The difference, of course, is that it is a lot faster with replays.

Let us imagine a dilligent DBA who, every week, analyzes a large **MySQL Proxy** log (as a **user-defined log** for which he has written a wonderful **custom report**). Furthemore, he cannot simply analyze each week's log by itself but must merge the new log data with all previous weeks. The problem is already apparent: week by week the log parsing will become more and more difficult until the DBA hurls his **red Swingline stapler** through a window.

The solution: merge each week's log into a single replay. (And, of course, save the original text log for backup.) The replay will also increase in size and decrease in speed but at a tremendeously slower pace than parsing and re-parsing the text logs.

The most ideal solution would be to save each week as its own replay and then merge the replays but this is a noted **limitation**. (It is also a **future feature**.)

Finally, the obvious question I have no answer yet is: how do you merge logs with a replay? The command line is simply: mysqlsla -lt TYPE NEW_LOG -replay REPLAY_FILE.

If a replay is loaded with the **replay** option and logs are given on the command line, the mysqlsla automatically merges the two and updates all the appropriate meta-values.

Making Replays from Replays

« Top

At any time, no matter what, a replay can be saved with one of the three replay options (**post-parse-replay**, **post-analyses-replay**, **post-sort-replay**). That means replays can be made from replays.

For example, a replay as the result of a replay merged with new logs can be made (as our example DBA mentioned in the previous section would do). The command line is: mysqlsla -lt TYPE NEW_LOG -replay OLD_REPLAY_FILE -post-parse-replay NEW_REPLAY_FILE. post-parse-replay could also be either post-analyses-replay or post-sort-replay.

Or, you could load a large, unfiltered replay and make from it a "sub-replay" of just SELECT statments (but remember the **limitation** that this sub-replay will be calculated only in relation to itself). The command line is: mysqlsla -replay UNFILTERED_REPLAY_FILE -sf "+SELECT" -post-parse-replay SUB_REPLAY_FILE -slient. Note two things: one, no **log-type** option because two, **silent** prevents any report from being made therefore we do not need to know the log type. Then, to replay the sub-replay and make a **standard report** for the SELECTs: mysqlsla -lt LOGTYPE -replay SUB_REPLAY_FILE.