Verifying MySQL Slave Sync Using mk-table-checksum and mk-table-sync

Sheeri K. Cabral, PalominoDB, Inc.

Abstract

At no point in time during replication does a slave verify its data matches the data on its master, yet often it is critical that a slave be in sync with the master. The only way to know if a slave is in sync with its master is to compare the data sets. Comparing data must be done at the same point in time on the master and slave, which introduces overhead and locking. The flexible nature of mk-table-checksum allows small amounts of data to be checksummed frequently. Should a data discrepancy be found by mk-table-checksum, mk-table-sync is designed to work with the results of mk-table-checksum to avoid duplication of efforts.

MySQL Replication

Replication in MySQL is a one-way operation. One MySQL instance, called a *slave*, is configured to connect, authenticate, read and apply binary logs from a certain point on another MySQL instance, called the *master*. The slave has two separate threads to handle replication: an I/O thread that connects to the master, reads the binary logs and saves information to a file on the slave called the *relay log*; and an SQL thread that reads the relay log and applies the information to the slave database. A master can be aware of a connected slave via `SHOW SLAVE HOSTS` if the slave is configured to register with the master[1], and the slave's I/O thread appears in the master's processlist.

A master's binary logs can be in `STATEMENT`, `ROW` or `MIXED` format. The default format is `STATEMENT` format, which stores changes as SQL statements. `ROW` format stores how row data should be changed, and was introduced in MySQL 5.1. `MIXED` format stores changes in `STATEMENT` format, automatically switching to `ROW` format under certain conditions[2]. Semisynchronous replication, introduced in MySQL 5.5, depends on the master receiving acknowledgement from a slave I/O thread. Both the binary log and what a slave replicates can be restricted to a partial set of databases or tables. Both the binary log and what a slave replicates can be restricted to a partial set of databases or tables.

Data Integrity Loss on a Slave

At no point in time during replication does a slave verify its data matches the data on its master. Non-deterministic queries can cause different data on the master and slave. Replication does not restrict what types of queries can be run on the slave, enabling writes to occur directly on the slave. Corrupt master binary logs, corrupt slave relay logs and human error when configuring a slave can cause data changes to be missed or duplicated. Queries on non-transactional tables change data immediately, with no possibility of rollback in case of partial completion. In-memory tables such as temporary tables do not persist between binary logs nor across a MySQL instance restart, making transactions that use such tables not completely deterministic in replication. Data can be out of sync due to MySQL bugs or network errors. If the data is out of sync, the SQL statements from `STATEMENT` and `MIXED` binary logging can cause the data to drift even further out of sync.

A MySQL slave is often used for purposes that require an exact copy of the master data, such as backup, reporting or offloading read queries from the master. Compromised data integrity can go undetected for a long time, during which time the data can become more and more out of sync. If a slave is not in sync with its master and is promoted to replace the master, there is no one source of truth for the data in the database. Different data can cause replication to stop entirely due to missing foreign keys and duplicate primary/unique keys. Slaves with a partial copy of the master may be desired, such as an instance that only replicates logs or an ETL instance that creates additional data on the slave. In these cases, data integrity is still critical for the subset of data being replicated[3].

Finding Out of Sync Data

Ensuring that a slave is in sync with its master requires comparing data from the master and slave at a given point in time. Comparing data is difficult when the data changes constantly and replication is not synchronous. The Maatkit tool mk-table-checksum allows data on a master and slave to be checksummed at the same point in time while

1 http://dev.mysql.com/doc/refman/5.5/en/show-slave-hosts.html

2 http://dev.mysql.com/doc/refman/5.5/en/binary-log-mixed.html

3 http://dev.mysql.com/doc/refman/5.5/en/replication-rules.html

minimizing overhead and locking. mk-table-checksum uses replication itself to ensure the master and slave are checksummed at the same point in time.

How mk-table-checksum Works

When the `--replicate` option to mk-table-checksum is given, the checksum is run and the result is written into the table specified (the *checksum table*). The checksum table must contain at least the following fields:

- db - the name of the schema containing this subset of data.
- tbl - the name of the table containing this subset of data.
- chunk - the ordinal position of this subset of data within the table, or 0 for the entire table
- boundaries – the filter used for this subset of data (e.g. `id` >=1001 and `id`<2000)
- master_cnt – the row count of the subset of data on the master
- master_crc – the checksum of the subset of data on the master
- this_cnt – the row count of the susbset of data on this instance.
- this_crc – the checksum of the subset of data on this instance

mk-table-checksum changes the binary logging format to `STATEMENT` for the session and performs a `REPLACE` on the checksum table, inserting a row using literal values for db, tbl, chunk and boundaries and functions for this_cnt and this_crc. Then it runs a `SELECT` on the checksum table to retrieve the values of this_cnt and this_crc, which it then stores into master_cnt and master_crc using an `UPDATE` on the checksum table with the literal values.

On the master, master_cnt and this_cnt are the same, and master_crc and this_crc are the same. When the `REPLACE` statement is replicated to the slave, the functions to find this_cnt and this_crc are repeated. Thus on the slave, the values of this_cnt and this_cnt reflect the count and checksum for the subset of data on the slave. When the `UPDATE` statement is replicated to the slave, the literal values for master_cnt and master_crc are set.

If a subset of data on the slave matches the same subset on the master, then the checksum table on the slave will show identical values for master_crc and this_crc, and identical values for master_cnt and this_cnt.

Checksumming Data Subsets

By default, mk-table-checksum will run a checksum against all tables and all databases, checksumming the concatenated values of each field in a given table. The tables to be checksummed can be limited using options to include or ignore databases and tables. Partial tables can be checksummed by limiting the rows checksummed using an arbitrary filter (`--where`) or a temporal value (`--since`), and by using a subset of fields (`--columns`).

Any subset of data to be checksummed can be split into *chunks* to minimize overhead and lock time. The *chunk size* can be specified as a number of rows or a data size. mk-table-checksum uses an index on a numeric or date field to split a subset of data into chunks. If a suitable index is not found, the table cannot be split into chunks, and either the table is skipped (version 6839 or later) or the entire table is checksummed (prior to version 6839, or if the `--unchunkable-tables` option is set). To checksum a subset of chunks, use the `--modulo` and `--offset` options. The flexible nature of mk-table-checksum allows it to checksum small amounts of data frequently.

Limitations

Due to the serialized nature of replication, using mk-table-checksum with `--replicate` can cause replication lag on the slave and excessive write blocking on the master if the amount of data being checksummed is large, though mk-table-checksum can be configured to sleep between checksum commands. Using mk-table-checksum as described in this paper will replicate to all the slaves of a master. False positives and false negatives are possible, depending on the algorithm used and in the rare case of checksum collisions. mk-table-checksum does not compare schemas; using it as described in this paper can cause replication to break if the slave schema is different from the master schema. A table with a storage engine that cannot use row-level locking will be blocked from any writes while the checksum occurs. If replication is off or broken, using mk-table-checksum as described in this paper will not work; if replication is lagging far behind, it can take a long time for the slave to show the results of the checksum.

Data Discrepancies

If the checksum table on a slave shows that a subset of data is different from the master, determine what data is different and, if possible, how the discrepancy occurred. An export tool such as mysqldump can be used to extract the data subset from the master and slave for comparison. Examining the data discrepancy may yield the cause of the issue, such as when a floating point number is rounded differently on a master and slave. In some cases, the issue is apparent but the cause is not, such as when a datetime value is offset by a consistent value. In other cases, no

pattern identifying an issue can be found by examining the data discrepancies.

Finding and fixing the cause of a data discrepancy is extremely important to keeping your master and slave in sync. If there is a systemic issue, syncing the data only fixes the problem temporarily. Though this is not an exhaustive list, data discrepancies can be caused by:

- Different MySQL versions on the master and slave
- Triggers or events running on a slave
- Deterministic queries, or queries with deterministic functions that are not safe for replication[4]
- Queries or transactions using temporary tables
- Queries or transactions using non-transactional storage engines
- Data changed directly on the slave

Fixing Data Discrepancies

A large discrepancy may warrant restoring from a known good backup. If the data discrepancy follows a pattern, it may be possible to use DML to fix the data on the slave so it is in sync with the master. Another Maatkit tool, mk-table-sync, may be used to fix discrepancies. As with mk-table-checksum, mk-table-sync defaults to syncing all databases and tables but can be limited to include or ignore databases, tables, and fields. The `--replicate` option can be used to specify a checksum table for mk-table-sync to use as a subset of data to sync, so that data does not need to be re-checksummed.

To get a list of queries that mk-table-sync will run, use the `--print` option. The data is only sync'd if the `--execute` option is used. mk-table-sync can be run against a slave to sync to its master, against a master to sync with all the currently connected slaves, or against a master and list of slaves.

Other Checksum and Sync Methods

MySQL has a built-in `CHECKSUM TABLE` command. This command obtains a read lock on an entire table and checksums all the fields and all the rows. There is no way to checksum a subset of a table, and it can be difficult to checksum a master and slave at the same time using this method – this method is not *replication-aware*.

The MyISAM storage engine recognizes the `CHECKSUM` parameter in table DDL, and updates the `CHECKSUM` field of `INFORMATION_SCHEMA.TABLES` and `SHOW TABLE STATUS` when the table is written to. There is no locking necessary to retrieve the checksum of a table with this parameter defined. However, it is only for tables using the MyISAM storage engine, only entire tables can be checksummed, and this method is not replication-aware.

Comparing logical exports has the benefit of seeing the data discrepancies without a separate process. Depending on the tool used (e.g. `SELECT INTO OUTFILE`, mysqldump, MySQL Workbench), table subsets can be exported and then compared to minimize locking. This method uses I/O and disk space for the exports, though the exports may be able to be done remotely. It is difficult to automate comparison with logical exports, and this method is not replication-aware.

Comparing and syncing physical files (e.g. using rsync or DBRD) does not give any information other than file names as to what is out of sync. It cannot help determine the cause of data discrepancies, and can easily cause corruption if a sync is interrupted. The process of comparing and syncing physical files is not replication-aware.

Conclusion

To verify that data on a slave matches the same data on its master, data must be compared at the same point in time on the master and slave. This can be difficult for MySQL instances that cannot be locked while the entire data set is compared. If data discrepancies are uncovered, the cause should be investigated and fixed to avoid the same data discrepancies in the future. mk-table-checksum can be used to frequently checksum small amounts of data, avoiding excessive locking and overhead while allowing comparison of data.

Acknowledgements

Thanks to Dossy Shiobara of Panoptic for reviewing.

About the Author

Sheeri K. Cabral was the first Oracle ACE Director for MySQL, has a master's degree in computer science specializing in databases from Brandeis University and a background in systems administration. Unstoppable as a volunteer and activist since age 14, Cabral founded and organizes the Boston, Massachusetts, USA, MySQL User Group and is the creator and co-host of OurSQLCast: The MySQL Database Community Podcast. She is the founder and current treasurer Technocation, Inc., a not-for-profit organization providing educational resources and grants for IT professionals. She wrote the *MySQL Administrator's Bible* and has been a technical editor for high-profile O'Reilly books such as *High Performance MySQL 2nd Edition* and *CJ Date's SQL and Relational Theory*.

4 http://dev.mysql.com/doc/refman/5.5/en/replication-features-functions.html