MySQL NDB Cluster 8.1 Release Notes

Abstract

This document contains release notes for the changes in each release of MySQL NDB Cluster that uses version 8.1 of the `NDB` (`NDBCLUSTER`) storage engine.

Each NDB Cluster 8.1 release is based on a mainline MySQL Server release and a particular version of the `NDB` storage engine, as shown in the version string returned by executing `SELECT VERSION()` in the `mysql` client, or by executing the `ndb_mgm` client `SHOW` or `STATUS` command; for more information, see MySQL NDB Cluster 8.1.

For general information about features added in NDB Cluster 8.1, see What is New in MySQL NDB Cluster 8.1. For a complete list of all bug fixes and feature changes in MySQL NDB Cluster, please refer to the changelog section for each individual NDB Cluster release.

For additional MySQL 8.1 documentation, see the MySQL 8.1 Reference Manual, which includes an overview of features added in MySQL 8.1 that are not specific to NDB Cluster (What Is New in MySQL 8.1), and discussion of upgrade issues that you may encounter for upgrades from MySQL 8.0 to MySQL 8.1 (Changes in MySQL 8.1). For a complete list of all bug fixes and feature changes made in MySQL 8.1 that are not specific to `NDB`, see Changes in MySQL 8.1.0 (2023-07-18, Innovation Release).

Updates to these notes occur as new product features are added, so that everybody can follow the development process. If a recent version is listed here that you cannot find on the download page (https://dev.mysql.com/downloads/), the version has not yet been released.

The documentation included in source and binary distributions may not be fully up to date with respect to release note entries because integration of the documentation occurs at release build time. For the most up-to-date release notes, please refer to the online documentation instead.

For legal information, see the Legal Notices.

For help with using MySQL, please visit the MySQL Forums, where you can discuss your issues with other MySQL users.

Document generated on: 2023-09-27 (revision: 27287)

Table of Contents

Preface and Legal Notices

This document contains release notes for the changes in each release of MySQL NDB Cluster that uses version 8.1 of the `NDB` storage engine.

Legal Notices

Documentation Accessibility

For information about Oracle's commitment to accessibility, visit the Oracle Accessibility Program website at
http://www.oracle.com/pls/topic/lookup?ctx=acc&id=docacc.

Access to Oracle Support for Accessibility

Oracle customers that have purchased support have access to electronic support through My Oracle Support. For information, visit
http://www.oracle.com/pls/topic/lookup?ctx=acc&id=info or visit `http://www.oracle.com/pls/topic/lookup?ctx=acc&id=trs` if you are hearing impaired.

Changes in MySQL NDB Cluster 8.1.0 (2023-07-18, Innovation Release)

MySQL NDB Cluster 8.1 is an Innovation release of NDB 8.1, based on MySQL Server 8.1 and including features in version 8.1 of the `NDB` storage engine, as well as fixing recently discovered bugs in previous NDB Cluster releases.

Obtaining NDB Cluster 8.1. NDB Cluster 8.1 source code and binaries can be obtained from https://dev.mysql.com/downloads/cluster/.

For an overview of changes made in NDB Cluster 8.1, see What is New in MySQL NDB Cluster 8.1.

This release also incorporates all bug fixes and changes made in previous NDB Cluster releases, as well as all bug fixes and feature changes made in mainline MySQL 8.1 (see Changes in MySQL 8.1.0 (2023-07-18, Innovation Release)).

IPv6 Support

- `NDB` did not start if IPv6 support was not enabled on the host, even when no nodes in the cluster used any IPv6 addresses. (Bug #106485, Bug #33324817, Bug #33870642, WL #15561)

Functionality Added or Changed

- **Important Change; NDB Cluster APIs:** The `NdbRecord` interface allows equal changes of primary key values; that is, you can update a primary key value to its current value, or to a value which compares as equal according to the collation rules being used, without raising an error. `NdbRecord` does not itself try to prevent the update; instead, the data nodes check whether a primary key is updated to an unequal value and in this case reject the update with Error 897: `Update attempt of primary key via ndbcluster internal api`.

Previously, when using any other mechanism than `NdbRecord` in an attempt to update a primary key value, the NDB API returned error 4202 `Set value on tuple key attribute is not allowed`, even setting a value identical to the existing one. With this release, the check when performing updates by other means is now passed off to the data nodes, as it is already by `NdbRecord`.

This change applies to performing primary key updates with `NdbOperation::setValue()`, `NdbInterpretedCode::write_attr()`, and other methods of these two classes which set column values (including `NdbOperation` methods `incValue()`, `subValue()`, `NdbInterpretedCode` methods `add_val()`, `sub_val()`, and so on), as well as the `OperationOptions::OO_SETVALUE` extension to the `NdbOperation` interface. (Bug #35106292)

Bugs Fixed

- **NDB Cluster APIs:** Printing of debug log messages was enabled by default for `Ndb_cluster_connection`. (Bug #35416908)

  References: See also: Bug #35927.

- **NDB Cluster APIs:** While setting up an `NdbEventOperation`, it is possible to pass a pointer to a buffer provided by the application; when data is later received, it should be available in that specified location.

  The received data was properly placed in the provided buffer location, but the NDB API also allocated internal buffers which, subsequently, were not actually needed, ultimately wasting resources. This problem primarily manifested itself in applications subscribing to data changes from `NDB` using the `NdbEventOperation::getValue()` and `getPreValue()` functions with the buffer provided by application.

  To remedy this issue, we no longer allocate internal buffers in such cases. (Bug #35292716)

- When dropping an `NdbEventOperation` after use, the `ndbcluster` plugin now first explicitly clears the object's custom data area. (Bug #35424845)

- After a socket polled as readable in `NdbSocket::readln()`, it was possible for `SSL_peek()` to block in the kernel when the TLS layer held no application data. We fix this by releasing the lock on the user mutex during `SSL_peek()`, as well as when polling. (Bug #35407354)

- When handling the connection (or reconnection) of an API node, it was possible for data nodes to inform the API node that it was permitted to send requests too quickly, which could result in requests not being delivered and subsequently timing out on the API node with errors such as Error 4008 `Receive from Ndb failed` or Error 4012 `Request ndbd time-out, maybe due to high load or communication problems`. (Bug #35387076)

- Made the following improvements in warning output:

  - Now, in addition to local checkpoint (LCP) elapsed time, the maximum time allowed without any progress is also printed.

  - Table IDs and fragment IDs are undefined and thus not relevant when an LCP has reached `WAIT_END_LCP` state, and are no longer printed at that point.

  - When the maximum limit was reached, the same information was shown twice, as both warning and crash information.

  (Bug #35376705)

- Memory consumption of long-lived threads running inside the `ndbcluster` plugin grew when accessing the data dictionary. (Bug #35362906)
- A failure to connect could lead `ndb_restore` to exit with code 1, without reporting any error message. Now we supply an appropriate error message in such cases. (Bug #35306351)
- When deferred triggers remained pending for an uncommitted transaction, a subsequent transaction could waste resources performing unnecessary checks for deferred triggers; this could lead to an unplanned shutdown of the data node if the latter transaction had no committable operations.

  This was because, in some cases, the control state was not reinitialized for management objects used by `DBTC`.

  We fix this by making sure that state initialization is performed for any such object before it is used. (Bug #35256375)
- A pushdown join between queries featuring very large and possibly overlapping `IN()` and `NOT IN()` lists caused SQL nodes to exit unexpectedly. One or more of the `IN()` (or `NOT IN()`) operators required in excess of 2500 arguments to trigger this issue. (Bug #35185670, Bug #35293781)
- The buffers allocated for a key of size `MAX_KEY_SIZE` were of insufficient size. (Bug #35155005)
- The fix for a previous issue added a check to ensure that fragmented signals are never sent to `V_QUERY` blocks, but this check did not take into account that, when the receiving node is not a data node, the block number is not applicable. (Bug #35154637)

  References: This issue is a regression of: Bug #34776970.
- `ndbcluster` plugin log messages now use `SYSTEM` as the log level and `NDB` as the subsystem for logging. This means that informational messages from the `ndbcluster` plugin are always printed; their verbosity can be controlled by using `--ndb_extra_logging`. (Bug #35150213)
- We no longer print an informational message `Validating excluded objects` to the SQL node's error log every `ndb_metadata_check_interval` seconds (default 60) when `log_error_verbosity` is greater than or equal to 3 (`INFO` level). It was found that such messages flooded the error log, making it difficult to examine and using excess disk space, while not providing any additional benefit. (Bug #35103991)
- Some calls made by the `ndbcluster` handler to `push_warning_printf()` used severity level `ERROR`, which caused an assertion in debug builds. This fix changes all such calls to use severity `WARNING` instead. (Bug #35092279)
- When a connection between a data node and an API or management node was established but communication was available only from the other node to the data node, the data node considered the other node “live”, since it was receiving heartbeats, but the other node did not monitor heartbeats and so reported no problems with the connection. This meant that the data node assumed wrongly that the other node was (fully) connected.

  We solve this issue by having the API or management node side begin to monitor data node liveness even before receiving the first `REGCONF` signal from it; the other node sends a `REGREQ` signal every 100 milliseconds, and only if it receives no `REGCONF` from the data node in response within 60 seconds is the node reported as disconnected. (Bug #35031303)
- The data node process printed a stack trace during program exit due to conditions other than software errors, leading to possible confusion in some cases. (Bug #34836463)

  References: See also: Bug #34629622.
- The log contained a high volume of messages having the form `DICT: index` *`index number`* `stats auto-update requested`, logged by the `DBDICT` block each time it received a report from

DBTUX requesting an update. These requests often occur in quick succession during writes to the table, with the additional possibility in this case that duplicate requests for updates to the same index were being logged.

Now we log such messages just before DBDICT actually performs the calculation. This removes duplicate messages and spaces out messages related to different indexes. Additional debug log messages are also introduced by this fix, to improve visibility of the decisions taken and calculations performed. (Bug #34760437)

- A comparison check in `Dblqh::handle_nr_copy()` for the case where two keys were not binary-identical could still compare as equal by collation rules if the key had any character columns, but did not actually check for the existence of the keys. This meant it was possible to call `xfrm_key()` with an undefined key. (Bug #34734627)

  References: See also: Bug #34681439. This issue is a regression of: Bug #30884622.

- When a data node process received a Unix signal (such as with `kill -6`), the signal handler function showed a stack trace, then called `ErrorReporter`, which also showed a stack trace. Now in such cases, `ErrorReporter` checks for this situation and does not print a stack trace of its own when called from the signal handler. (Bug #34629622)

  References: See also: Bug #34836463.

- Local checkpoints (LCPs) wait for a global checkpoint (GCP) to finish for a fixed time during the end phase, so they were performed sometimes even before all nodes were started.

  In addition, this bound, calculated by the GCP coordinator, was available only on the coordinator itself, and only when the node had been started (start phase 101).

  These two issues are fixed by calculating the bound earlier in start phase 4; GCP participants also calculate the bound whenever a node joins or leaves the cluster. (Bug #32528899)

- When an `ALTER TABLE` adds columns to a table, the `maxRecordSize` used by local checkpoints to allocate buffer space for rows may change; this is set in a `GET_TABINFOCONF` signal and used again later in `BACKUP_FRAGMENT_REQ`. If, during the gap between these two signals, an `ALTER TABLE` changed the number of columns, the value of `maxRecordSize` used could be stale, thus be inaccurate, and so lead to further issues.

  Now we always update `maxRecordSize` (from `DBTUP`) on receipt of a `BACKUP_FRAGMENT_REQ` signal, before attempting the allocation of the row buffer. (Bug #105895, Bug #33680100)

Index