On-line schema changes on Clustrix

On-Line Schema Changes on Clustrix

Clustrix has the ability to perform schema changes on-line without blocking reads or writes to a table. These changes can range from as simple as adding an index or column, to completely redesigning the schema. Many other database systems require partial or full table locks while schema changes process each row moving the data structure into its new form.

Clustrix avoids the locking problem by creating a new temporary container (or set of containers) and copying the data into its new structure. Simultaneously, a temporary transaction log records any write or update transactions run against the old table during the ALTER's execution and are applied to the new container at the end of the copy. Once all the original records have been copied and logged transactions have been processed, the ALTER transaction commits, the new table goes live and the old table is dropped.

Read and write queries against the table before the ALTER commits see the original table schema. Reads and writes after the ALTER commits see the new schema. From the perspective of any single query or user, the ALTER is instantaneous.

Preparation

While the Clustrix database engine is designed to easily support online schema changes for tables of any size, following best practices helps to minimize negative and unforeseen impacts to your application. Typical best practices include:

- Backup your data.
- Testing the new schema with your application's queries in an isolated environment. Compare `EXPLAIN <query>` plans before and after the change.
- Perform schema change(s) off-peak or during a maintenance window.
- Ensure you have adequate disk space (see below).
- Understand impacts to replication and revise plan accordingly.

The first three of the above are entirely within your control, but the last two deserve some explanation

Estimating Disk Space Requirements

To understand how much space you need free before running an ALTER, you first need an understanding of Clustrix's garbage collection process, which we refer to as BigC.

BigC is essentially a checkpoint which constantly moves forward through time as transactions complete. The BigC value of the cluster (shown in system.stats) will always show the point in time at which the longest currently-running transaction has started. Any explicit transaction will "pin" BigC. While garbage collection is pinned, the cluster will log any write, delete, update, and DDL transactions into the undo-log for the possible event in which the long-running transaction fails and needs to be rolled back.

Because ALTERs are DDL transactions, they pin BigC which means that all garbage collection is suspended during the ALTER.

In order to calculate how much free space is required for an ALTER you should evaluate each of the following:

- Current space used.
- Minimum free space desired. ClustrixDB will kill long-running transactions if free space falls under 5%, this will cause your ALTER to rollback. Plan on reserving 10% free space at a minimum.

- The size of the table to be ALTERed. We'll need room for a copy. You can get an estimate from `system.table_sizes`
- Estimated amount of time it will take to complete the ALTER. This depends on the size of your cluster (e.g. it's faster with more servers). Use the speed of previous, smaller ALTERs to extrapolate.
- New data growth rate.
- Undo-Log growth rate. The undo-log for all INSERTS, UPDATES, DELETES will accumulate during the ALTER.
- If replicating, the binlog growth rate. Binlog trims will still run, but the disk space will not be free'd until BigC is un-pinned.

When you add up all the planned growth and size of the table copy plus the current space consumed, you should still have at least 10% of disk remaining in your plan. If not, trimming back binlogs, pruning tables, or expanding the cluster with additional servers is recommended before initiating the ALTER.

Replication Concerns

Performing a ALTERs in a replicated environments requires additional planning. The ALTER can be performed in-line with replication, or independently, depending on whether schema structure is changed. It is highly recommended whenever possible to allow the ALTER to run over the replication stream to avoid errors in replication where transactions are looking for either the old or new schema when the other is present.

Running the ALTER over replication means that the ALTER must execute sequentially, first on the master, then on the slave. Furthermore, the slave must process the ALTER sequentially within the replication stream, pausing all other writes while it works through the ALTER. For large table ALTERs, this can cause the slave to fall behind the master by some amount. Plan to adjust binlog retention on the master and monitor replication regularly to ensure the slave does not fall off the back of your master's binlog retention during this process.

When replicating via row-based binlogs, column synchronization is critical. RBR sends both the old and new rows to the slave and the old must match before the new is applied. Alters that change columns must be executed within replication and in sequence to avoid slave errors trying to applying writes for one schema to the other.

When replicating via statement-based binlogs, more flexibility is allowed provided that the statements execute equally well on the old and new schema. However, for large concurrent transaction environments, knowing whether all statements are schema safe may be impossible.

Exceptions to these column matching concerns include ALTERs adding an index, changing the storage type, or number of slices. Since these do not affect the insertion or update of rows, they may be executed in parallel outside the replication stream.

Scale-Out Concerns

The performance of ALTER on user data will vary depending on your particular environment and workload. Take the following as guidance and adapt based on your own experience and testing.

Simple Small Table

For tables around N*10^6 rows or with light concurrency, ALTERs can be done live and should complete within minutes depending on the cluster size, row size, and overall load. Cache and query plan refreshes occur regularly and the automatic refresh should handle any performance issues.

Medium Scale-Out Issues

Tables that are being accessed with high concurrency or have more than N*10^7 rows may have a detrimental

impact on cluster performance immediately after the ALTER completes. This is due to the database's internal caches holding stats and query plans for the prior-state of the table. Current versions of Clustrix may not refresh or flush these caches quickly enough to avoid performance impacts so we recommend following the ALTER with the following flush commands as follows:

```
mysql> ALTER TABLE ….
$ clx cmd ‘mysql system -e “call qpc_flush()”’;
$ clx cmd ‘mysql system -e “call pdcache_update()”’;
```

Note that qpc_flush and pdcache_update are done on a per-node basis. Therefore, we need to execute these using the `clx cmd` utilty.

Chaining these commands together in the shell as `command && command && command` is recommended to avoid delays between the completion of the ALTER and flushing of the caches.

Extreme Scale-Out Concerns

For tables with more than N*10^8 rows, an ALTER may take many hours or even several days to complete. Pay extra careful attention to all of the rules and recommendations above regarding space considerations, concurrency, et cetera. In addition, once the ALTER completes, the BigC garbage collection may cause performance impact to your system for an extended period of time while it catches up. Should this occur during your peak-hours, you may want to temporarily throttle BigC via the adjustment of the following system variable to its minimum value of 524,288:

```
mysql> set global layer_copy_speed = 524288;
```

Once layer_merges has caught up (verify this by looking at system.layer_merges <NEEDS HOWTO>), this variable can be restored via `set global layer_copy_speed = default;`

If you have adequate free space you may want to pin BigC with a separate transaction so you can control the release to a time of your choosing. You can accomplish this with a separate session and an explicit BEGIN; with no COMMIT. Once off-peak, either run COMMIT or ctrl+c out of that session. However, we've found that in most cases, reducing layer_copy_speed is sufficient to avoid excessive impact to production workloads. Keep in mind that artifici ally pinning BigC is subject to the `idle_trx_timeout` killer. Be sure to check that global in the database to prevent that session from being killed.

Example

A 2 TB table will require an estimated ~50 hours to alter with a given row size. The cluster undergoes ~1.5 TB per day of write transactions (including any binlogs). A conservative estimate would be that there should be at least 6TB available space on the cluster to allow the alter to complete.

TODO: show math here

Example alter scenario:

```
mysql> set global layer_copy_speed = 2097152; (optional)
```

Session A:

```
mysql> BEGIN;
```

Session B:

```
mysql> alter table database.hugetable add column (‘foo’ varchar(255));
```

---- many hours later, after alter completes, and cluster is off peak load ----

Force pdcache_update and qpc_flush.

```
$ clx cmd ‘mysql -e “call system.task_run (‘pdcache_update’)”
$ clx cmd ‘mysql -e “call system.qpc_flush()”;
```

Session A:

```
mysql> COMMIT; (or ^+c)
```

- Monitor cluster performance over the next few minutes/hours - depending on how large the ALTER was - You can examine the system.layer_merges relation for status of post-alter cleanup.

Once completed, re-set layer_copy_speed to default:

```
mysql> set global layer_copy_speed = default;
```