Loading Data Onto Xpand

The most common methods of loading data onto Xpand are:

1. Take a mysqldump from an existing MySQL instance, then import the dump with clustrix_import. This is the fastest way to get data onto Xpand.
2. Use LOAD DATA INFILE to import CSV files. Xpand performs best when data is pre-sliced and the import can be done in parallel.

This is often followed by setting up replication with Xpand as a slave from the MySQL instance. Once the Xpand Replication Slave is caught up, then the application can be cut over to Xpand and the MySQL instance retired.

mysqldump and clustrix_import

Using mysqldump

Ensure a consistent transactional backup suitable for setting up replication

When performing the mysqldump from your MySQL server, ensure you provide the proper arguments, particularly for single-transaction and master-data. Note that since MyISAM tables are not transactional, data for those tables can continue to change while mysqldump runs. To get a consistent dump of MyISAM tables, it's necessary to disable writes entirely or lock all tables during the dump. Since this is generally not feasible on a production cluster, it may be necessary to dump from an existing slave instead, where replication to that slave can be stopped for the duration of the dump.

Ensure the dump completes successfully

To avoid having mysqldump interrupted by a network connection reset or similar issue, Xpand Support recommends using the screen utility to run mysqldu mp. If you do any amount of serious work at the command line, screen is indispensable. Regardless of whether you use screen or some other method to invoke and monitor, always check the tail of the dump file to make sure the dump completed successfully. You should see some session variable sets. If you see the middle of a multi-row insert instead, your dump was interrupted, or the file was otherwise truncated. Either way, you're unlikely to be pleased with the results of restoring such a file.

Don't bother with the mysql database (users and permissions should be copied with clustrix_clone_us ers)

Avoid dumping internal MySQL databases such as mysql, which are of no use to Xpand. Xpand will dutifully create a mysql database and restore the contents, but they will have no effect on the functioning of the system, as would be expected if the dump were restored to a MySQL server. In particular, users and permissions cannot be propagated this way. See Migrating User Permissions for information on how to use clustrix_clone_users.

clustrix_import

Standard MySQL practice is to import mysqldump by redirecting to the mysql client on the shell command line, or using the source command within the mysql client. Note that this method can result in very long import times as it fails to take advantage of Xpand parallel processing.

clustrix_import is a Python script that reads mysqldump output and loads the data into a Xpand cluster in a multi-threaded fashion. It can be run directly on the cluster (in which case the dump should be staged in the /data/clustrix directory, which has plenty of space), or from any Linux client with Python 2.4 and MySQLdb (MySQL driver for Python) 1.2.1.

The tool is available on your cluster in /opt/clustrix/bin.

clustrix_import first connects to the cluster at the given address and determines the number and IP address of all nodes. It then spins up 8 (default) threads per node and distributes the inserts across these threads. To obtain optimal slicing, it creates tables with a large slice count (3x number of nodes in the cluster), and then reslices smaller tables back to a smaller slice count. It also checks for poor distribution, where an index is dominated by a single value, and fixes this. See Distribution Key Imbalances in Data Distribution.

In addition to parallelism and optimal data distribution, clustrix_import has a few "handrails", such as checking whether a binlog exists, since in most cases you would not want all the statements associated with an import going into a replication stream, particularly DROP statements. Also, clustrix_import has a locking mechanism that prevents multiple instances of clustrix_import from running. Since a single instance will already fully utilize cluster resources, additional instances could reduce overall throughput or be destabilizing.

For additional information, please see clustrix_import.

Loading Data Without clustrix_import

If clustrix_import cannot be used to import your data, you can take some proactive measures to ensure efficient data population. While the Rebalancer can ultimately rectify just about any problem created during initial data load, poor slicing and distribution can result in much longer import time, and it may take quite some time for the Rebalancer to achieve optimal data distribution.

Pre-slicing Tables

number of slices = number of nodes

When populating large tables (i.e.10GB or larger) it is advantageous to set the table slice count when the table is created and *before* loading data. This avoids the problem of "racing the Rebalancer", wherein the Rebalancer recognizes that the table needs more slices and begins the splitting process while still actively loading data. This results in longer data load time. If you can estimate the size of the data you are importing (potentially by importing some number of rows and checking the size in system.table_sizes), a good rule of thumb is a little more than 1 slice per 1GB. Generally, you want at least one slice per node for optimal load distribution. Setting the global variable hash_dist_min_slices to the number of nodes will achieve the same result.

To set slice count at table creation time, simply append SLICES=N to the end of your CREATE statement. You can also reslice the table with ALTER TABLE foo SLICES=N. Note that in both cases, the slicing for the base representation and all indexes are set to N.

Pre-slicing for tables > 100GB

For very large tables (larger than 100GB), you may wish to independently set the slice count for the base/primary representation (which contains all columns of each row) and the indexes (which will contain the columns included in the index, as well as the column(s) of the primary key). Generally, indexes will require fewer slices than the base representation, since the tuples are much narrower; how many fewer slices are required depends on how wide the full table is (particularly how many varchar or blob columns), and whether an index includes such a wide column. Instead of estimating based on column count/size for indexes, you may also load a small but representative portion of your data into Xpand, and then use the system.index_sizes table to ascertain the relative sizes of the base representation and indices.
You can set slicing for individual indexes by including SLICES=N within the index definition itself. Place the SLICES = keyword before the comma that separates multiple indexes or before the closing parenthesis of the last index.

```
CREATE TABLE ...
     PRIMARY KEY (`id`),
     KEY `index_bids_on_user_id` (`user_id`) SLICES=12,
     KEY `index_bids_on_timestamp` (`timestamp`) SLICES=24,
     KEY `index_bids_on_auction_id` (`auction_id`) SLICES=12
     ) SLICES=36;
```

Note that slicing specified after the closing parenthesis applies to the base representation of the table and to all indices for which explicit slicing was not specified.

See the information on slices for further information.

Anticipating table growth

In addition to the guidelines above, consider also how much your table is expected to grow over time. You may wish to slice your tables into 0.5GB slices if you anticipate rapid table growth.

Parallelize Data Load

Besides slicing and distribution, the main factor in import speed is the degree of parallelism. A single-threaded import process fails to take proper advantage of the Xpand parallel architecture, and may run even more slowly than on a MySQL instance. Consider how the data load process could potentially be divided to increase parallelism. For example:

- For LOAD DATA INFILE, you can split files into smaller chunks and run them in parallel.
- If your application loads data into the database directly, consider whether this data load can be performed in a multi-threaded fashion, with each thread connecting a different session via a load balancer, to distribute the front-end connections across the cluster nodes.

Use Multi-Row Inserts

Where possible, aggregate single insert statements into larger multi-row statements. Xpand handles these multi-row statements more efficiently, particularly since it reduces the per-row transactional overhead. Combining parallelism with multi-row inserts should provide optimal data load performance. (This is the same thing that clustrix_import does).