ScaleDB Cluster Manual

This is the Reference Manual for ScaleDB Database System version 1.02. This Manual explains the installation and operation of a ScaleDB Cluster. For support, please contact us at support@scaledb.com.

Legal Notices

Copyright © 2013, ScaleDB Inc. All rights reserved. The ScaleDB software and related documentation are provided under a license agreement containing restrictions on use and disclosure and are protected by patents and intellectual property laws. A copy of the license agreement is available at http://www.scaledb.com/agreement.php. The information contained herein is subject to change without notice and is not warranted to be error-free. If you find any errors, please report them to us at support@scaledb.com.

Table of Contents

Legal Notices ........................................................................................................................................... 1
Overview: .............................................................................................................................................. 2
    Software and Hardware Requirements .......................................................................................... 5
    Cluster Setup Considerations: .................................................................................................. 5
    Routing queries to a database node ............................................................................................ 7
Installing Components ......................................................................................................................... 7
    The ScaleDB cluster files system ............................................................................................... 8
        The user data ........................................................................................................................ 10
        The Log Files ..................................................................................................................... 11
        The Registry Files .............................................................................................................. 12
        Summary of the different file types .................................................................................... 13
Data Load ........................................................................................................................................... 13
Limits on ScaleDB Tables .................................................................................................................. 14
Configuration Files: ......................................................................................................................... 15
    MySQL Configuration file: ...................................................................................................... 15
        The ScaleDB configuration files: ....................................................................................... 15
        Configuration files and connecting nodes to the cluster: ................................................ 16
Starting a Cluster ............................................................................................................................... 17
    Connection parameters ........................................................................................................... 21
Overview:

ScaleDB is an integrated platform of database and storage services to manage online transaction processing (OLTP) and data warehousing applications. ScaleDB is deployed as a cluster of servers that operate on shared data delivering high-performance and high-availability database services for a wide range of applications using commodity hardware or on the cloud.

ScaleDB operates at multiple tiers. At the database tier, multiple database engines share access to the entire collection of data. At the storage tier, data is placed on multiple storage nodes. For MySQL or MariaDB it provides a way to share data among multiple database server instances.

Non MySQL users may use the ScaleDB functionality through the ScaleDB native API. The ScaleDB API is available here: http://code.google.com/p/scaledb-interfaces/source/browse/
Note, this manual provides examples assuming a MySQL or MariaDB database user. However, the same functionality is available through the native API for non-MySQL usage.

ScaleDB provides scalability and high-availability enhancement for MySQL and MariaDB. For scaling, it eliminates the need to partition your data. High Availability is provided at the storage layer transparently by managing a mirrored set of the data.

An overview of the ScaleDB product is available here: http://www.scaledb.com/pdfs/ScaleDB_Cloud_DBMS.pdf

ScaleDB is installed on a cluster of servers. The servers can be physical or virtual machines connected by a TCP based network. Each connected machine is configured to serve as a database node, a storage node or a cluster manager node. A typical cluster would have multiple database nodes, multiple storage nodes and 2 cluster manager nodes. The number of database and storage nodes in a cluster depends on the data volumes, the characteristics of the database usage and the number of users of the database.

Therefore, each node in the cluster requires installation of one of the following software components:

1) **The database nodes** - Each database node is a physical or virtual machine running MySQL or MariaDB database server and ScaleDB database engine. The ScaleDB database engine on the database node is considered by MySQL/MariaDB as a local storage engine.

   With ScaleDB, multiple database nodes are connected to a cluster whereas each database may query and update the entire data set. Conflicts between databases are automatically resolved by the system.

2) **The storage nodes** - Each storage node is a physical or virtual machine running ScaleDB’s Cache Accelerator Server (CAS) software. A storage node maintains the data (on local or remote physical storage) and provides a shared data persistency and caching facility for the database nodes.

   A cluster setup has at least one storage node. However, large databases may utilize multiple storage nodes, enabling the data to be distributed among the nodes. Data distribution is done automatically on any number of nodes which are assigned for data storage.

   The setup of the storage nodes is done in 2 dimensions: The first dimension manage the distribution of the data. For example, with 2 storage nodes, half of the data is placed on each node. With 10 storage nodes, 1/10 of the data is placed on each node. Each distribution of the data is considered a volume. Therefore, the data is distributed to N volumes whereas each volume maintains a portion of the entire data set. The second dimension provides copies of the data for high availability. Therefore, each volume may be configured to maintain a copy of the data. When data is written, it is being written at a block level and is placed on one volume. If the volume is configured to include a mirror, the data is written twice - on the main node of the
volume and on the mirror. With mirror nodes, if a storage node fails (the main or the mirror), the system continues to operate with no downtime.

The diagram below shows a setup of 4 volumes. Each volume is supported by 2 storage nodes – a main storage node and its mirror. Each storage node is a physical or virtual machine with CAS software instance. In this example, the entire data set is distributed over the 4 volume. When a block of data is written, it is being send to one of the volumes, and is written twice – on the main node and its mirror.

3) The Cluster Manager (CM) - a physical or virtual machine executing the CM code to coordinate the processing among the nodes in the cluster. This node manage locking, messaging and data sharing between nodes.

To provide High Availability, a cluster is configured with a second CM machine (Standby CM). If a cluster is initiated without a Standby CM node, if the CM node fails, operation would resume when a new CM node is initiated. With a standby CM, fail-over is automated such that if the CM fails, the standby CM manages the cluster without downtime.

These machines are connected with TCP based networks to allow the following:
- a) Messaging between that database nodes and the Cluster Manager.
- b) To support IO operations such as read from and write to the storage nodes.
These processes may share the same network or use different networks.

The diagram below shows 2 database nodes at the database tier. Each database node is configured with MySQL or MariaDB as the DBMS and ScaleDB as the database engine. The storage tier organizes the data in 3 volumes. Therefore, each volume maintains 1/3 of the data and each write is placed on 2 nodes (main and mirror).
Software and Hardware Requirements

ScaleDB does not require special hardware. Database nodes, storage nodes and cluster manager nodes can be configured on virtual or physical machines. The nodes need to be connected with a TCP based network. For functional or application compatibility testing, a ScaleDB cluster can be configured on a single machine. For performance testing and production, additional servers and optionally fast interconnects are needed.

Cluster Setup Considerations:

The number of database nodes in the cluster – When a database node is fully utilized, queries can be routed to additional database nodes and leverage the additional CPU, RAM and processing capability of the additional nodes. The number of database nodes in the cluster is usually set such that there is sufficient processing power on the database nodes to efficiently support all users requests at any given point of time.
The number of storage nodes in the cluster – There are 2 considerations that determine the number of storage nodes in a cluster: High Availability and Performance.

Each data volume can be configured with one or two storage nodes. However, to support High Availability, each volume requires 2 storage nodes.

If each volume is configured with a mirror node, a data write of a database node is completed only if it is placed on both – the main and the mirror node.

The number of volumes is determined by the data set size:
By increasing the number of volumes, a larger portion of the data fits in memory and additional CPUs are available to support the I/O requests.

With N volumes, the data is split into N parts such that every volume maintains 1/N of the data. If the size of RAM on each node is equal or larger than 1/N of the size of the data, most data requests (to read and write data) is satisfied without the need to access the disk drives. If the data size on a particular node is larger than its RAM, most frequently used data is available in RAM and the less frequently used data is retrieved from disk and placed in RAM whenever needed.

Configuring the cache size – The cache size needs to be configured for the storage node and the database nodes.
When a storage node reads data from a disk drive, it is placed on the local cache of the storage node in order to support requests for the same data without disk I/Os.
When a database node requests data from a storage node, the data is send from the storage node to the database node and is placed at the local cache of the database node. If data is available on the local cache of the database node, requests for the data is satisfied from the local cache without data requests and data transfer from the storage volume that manage the needed data.

ScaleDB maintains 3 types of cache segments on each storage node and each database node. Configuring a system with sufficient memory for each segment is a key component in achieving high performance:

The following are the types of cache segments that need to be configured for the storage nodes and the database nodes:
   a) Cache segment for index blocks
   b) Cache segment for data blocks.
   c) Cache segment for blob data.

These segments keep copies of blocks that are frequently used.
For storage nodes, large enough segments minimize the number of disk I/Os.
For the database nodes, large enough segments minimize the request to transfer data from the storage nodes to the database nodes.
A good starting point to determine the ratio between each type of segment is to estimate the ratio between the size of the indexes, structured data and unstructured data. This ratio can be determined by considering the ratio of the size of the files maintaining the indexes, the table data and the unstructured data.
However, if the unstructured data (such as blobs) is less frequently retrieved, it may be required to reduce the size of memory configured for unstructured data and increase the size provided for table data and indexes.

A database node is configured with a 4\textsuperscript{th} type of cache which is used to accelerate queries. This cache is a \textit{cache segment for a dynamic hash table}. When a row is retrieved to satisfy a query, the row is placed in the dynamic hash such that subsequent requests to the same row are satisfied from the hash without an index lookup.

The best way to determine the memory size for each segment is by testing different configurations. For a given RAM, different databases would show different performance characteristics when the ratio between the different segments size changes. The total cache size on each machine needs to be such that as much RAM as possible is available for caching as long as there is no Operating System swapping of memory to disk.

\textbf{Routing queries to a database node}

With ScaleDB, every database node in the cluster can process any query. However, in some cases, and when intelligent routing is possible, performance and throughput can increase when queries to the same database or table are routed to the same node. If query A touches the same index and data blocks as query B, and both are executed on the same database node, there are less conflicts to resolve in the cluster and less data shipment between nodes.

\textbf{Installing Components}

To setup a cluster, 3 types of nodes need to be installed:

One or more database nodes, where database processes are performed, one or more storage nodes, these nodes maintain the data for, and one or two cluster manager nodes that synchronize the processes in the cluster.

The Database Nodes – Each database node is configured with MySQL or MariaDB DBMS and ScaleDB database engine. (use these rpms for CentOS/RedHat/Fedora).

The Storage Nodes – Each storage node is configured with ScaleDB storage node software. For the storage node software, download cache\_server executable from ScaleDB website (www.scaledb.com/supportsuite).

In this document, storage nodes are assigned with an identification number and a role (main or mirror). For example, the main storage node in the first volume is referred ad Cache Accelerator 1 (or CAS1) - primary and CAS1 – mirror refers to its mirror.

The Cluster Manager - Download cluster\_manager executable from ScaleDB website (www.scaledb.com/supportsuite).
In this document, the main cluster manager node is referred as CM1 and the standby is referred as CM2.

The ScaleDB cluster files system

ScaleDB manage data in files. These files are placed on one or more storage nodes. Each storage node is connected to the cluster and the files that are managed by each storage nodes are available to all the database nodes in the cluster. There is no need for dedicated Network File System (NFS). When a database node needs data, a message is sent to the storage node that maintains the data and the storage nodes replies with the data needed.

The following demonstrates the setup of the directories on the storage nodes. The specific configuration is defined in the configuration files.

a) A set of directories that manage the user data. Each directory contains the data of a logical database. For example: the directory /home/bob/mycluster/scaledb_data/test manages all the data of a database with a logical name 'test' and the directory /home/bob/mycluster/scaledb_data/tpc manages the data of the database with the logical name 'tpc'. These directories need to be created on all the storage nodes of the cluster.

b) A default directory for all databases which have not been assigned with specific directories. The default directory is also used to manage ScaleDB meta data. The default directory needs to be created on all the storage nodes of the cluster.

c) The nodes of volume 1 manage the log files, and therefore, to manage the logs, each node of volume 1 is set with the directory /home/bob/mycluster/scaledb_logs.

**Note 1:** We distinguish between private files and public files. Private files are files that are accessed by a single instance and are not shared among instances of the cluster. For example, the MySQL frm files are private as each MySQL/MariaDB server instance owns a set of frm files. Public files are files that are shared among multiple instances of the cluster. For example, the data files (containing the user data) are shared among the database instances of the cluster and are considered public files.

**Note 2:** ScaleDB does not create directories. The directories need to be set by the administrator prior to usage. Data directories are usually created such that all the files of a particular logical database are contained in a particular directory. If a directory for a logical database is not available, the files of the logical database are placed in a default directory.

**Note 3:** Make sure that there are proper access rights to create and delete files in the directories being used.
Note 4: A private file is placed in a directory accessible to the process using the file. A public file is placed in a directory on a storage node as it is accessible to all the database nodes in the cluster.

Note 5: A Public file is one of 2 types:

a) A striped file, such as a data or an index file. A striped file is organized in blocks. Each block is assigned to a volume and is managed by the storage node of the volume. If the volume is supported by a mirror node, a copy of the block is available on the 2 nodes of the volume (on the main node and on the mirror).

b) A non-striped file, such as a log file, is maintained on the storage nodes of volume 1 (the main node and its mirror)

Note 6: the directories maintaining a non-striped file needs to be created on each node of Volume 1 (on the main node and its mirror).

In the diagram below, 3 database instances (marked green) are using 2 Volumes (the storage nodes marked gray) and 2 cluster managers (marked blue) - main and standby. All servers are connected by a network.

In the configuration below, the private files for each database node are placed on the local disk of each database node. The private file of each cluster manager is placed on the local disk of the cluster manager.

The public files are placed on the disk drives of the storage nodes.

Each striped file is found on each storage node and non-striped files are available on the disk drives of the storage nodes in Volume 1.
There are 5 types of files that are managed in a ScaleDB cluster:

MySQL/MariaDB frm file - the frm files represent the meta-data derived from the table’s definitions. These files are created and used by the MySQL/MariaDB server instance and can be placed on any disk drive accessible to the MySQL/MariaDB server instance.

ScaleDB does not modify the content of the frm files. However, when a MySQL or MariaDB node creates an frm file, ScaleDB stores a copy of the file on the storage nodes such that it is (transparently) available to all the MySQL servers connected to the cluster. If a new database node joins an existing cluster, the frm files are on the storage nodes and therefore available to the new connected database node.

The database files - ScaleDB files containing the user data and indexes. These files are available for update and search to all the MySQL database instances in the cluster and are placed on the storage nodes.

ScaleDB log files - provide the means to recover failed nodes. These files needs to be available to the ScaleDB processes and are placed on Volume 1 (CAS1 – main and CAS1 – mirror).

The size of the log files is determined by the configuration parameters. When the file reaches the limit, a new file is opened and is updated with the new logged data. ScaleDB automatically deletes log files which are not needed for recovery.

ScaleDB registry files - provide information on members of the cluster. These files are used by ScaleDB processes and are placed in the default directory of Volume 1 (CAS1 – main and CAS1 – mirror).

ScaleDB configuration file - provide setup information for the ScaleDB processes in the cluster. The main configuration file is placed on Volume 1 (CAS1 – main and CAS1 – mirror). Individual nodes may use private configuration file which can be placed on any drive accessible to the individual node.

The user data

The user data is maintained on the storage nodes and is available to all the database nodes in the cluster. When a database node needs data, it sends a message to the storage node that maintains the data requesting the block that contains the data. This block is sent by the storage node to the database node. Frequently used blocks in a database node are kept in the cache segment of the database node for later use. Frequently used blocks in a storage node are kept in the cache segment of the storage node for later use. When a database node needs to update data on a storage node, it will send the block containing the data to the storage node (and its mirror, if available) that maintain the data.
When a user defines a table, ScaleDB organizes the data of the table and its indexes in one or more physical files. The number of files supporting a table depends on the amount of data, the number of keys defined and the type of data:

ScaleDB data files maintain the rows of the table. ScaleDB index files maintain the indexes for the particular table. ScaleDB blob files maintain the unstructured data of a table.

These files are partitioned to disk based blocks. When a block is written, it is sent to one volume in the cluster. If the volume is backed by a mirror node, the block will be sent to both - the main storage node and the mirror.

On a particular storage node, if a file exceeds a predefined size, it is split into multiple physical files. These files are assigned with file numbers which make up part of their file names. The max file size can be defined in the configuration file.

The file names for the user data include the database name, table (or index) name, partition id, and a file number.

Below is the method used to name the files:

data files - the name of the file ends with .dat.sdb
index files - the name of the file ends with .idx.sdb
unstructured data files - the name of the file ends with .ovf.sdb

The location of the files on the disk drives of the storage nodes is defined by the configuration parameters.

The default location for user data is defined in the configuration file. The example below, sets the default location to /scaledb/data:

```
scaledb_data_directory   = /scaledb/data
```

The default location can be changed for particular (user defined) databases. The example below places all the data of mydb dbms at scaledb/data/mysb:

```
scaledb_db_directory mydb = /scaledb/data/mydb
```

**The Log Files**

The ScaleDB log files are used to recover node failure and support rollback operations. The log files are placed on Volume 1 of the storage tier.
The log files are placed on the shared storage such that in the case of a failure of a database node, a different database node can access the log files of the failed node and roll back the uncommitted data of the failed node.

Each database node in the cluster uses multiple log files. The names of the log files have 3 parts as follows:
The OS name of the machine of the database server
A number that represents the log file id
A prefix with the following string - .log.sdb

The location of the log file for all the database nodes is specified in the configuration file as in the example below:

```
scaledb_log_directory = /fast_disks/scaledb_logs
```

Therefore, the directory /fast_disks/scaledb_logs on the drives of the nodes in Volume 1 contain the log files of each node in the cluster.

**Note 1:** The log files needs to be able to support a rollback of the largest transaction. Therefore, ScaleDB maintains as many files as needed such that rollback and recovery of the longest running transaction is supported.

**Note 2:** During data loads, the load transaction may be very large and contain millions of records therefore requiring lots of disk space for log files. If it is not necessary to maintain the load as a single transaction, set the scaledb_autocommit to cap the number of DMLs in the transaction. This will commit the load data every time a specified number of DMLs was processed. This process will manage the load process with a smaller number of log files. For more information on data loads, see the chapter Data Load in this document.

Log files are created and deleted dynamically. The maximum size of each log file is determined by the configuration file and the number of log files depends on the currently running transactions. If a transaction is open, it is represented in the log file, and this log file is deleted only when there are no open transactions represented in the file.

To maintain unused log files on the file system use the following setting in the configuration file:

```
scaledb_delete_log_files = false
```

The default setting is true and it will delete log files which are not necessary for rollback or recovery.

**The Registry Files**

The registry files are created and modified by ScaleDB transparently while the system is running. These files maintain information about the nodes connected to the cluster. These files are named with .registry ending.
Summary of the different file types

The table below summarizes the different file types and their location in the cluster:

<table>
<thead>
<tr>
<th>File Type</th>
<th>Private</th>
<th>Public</th>
<th>Location</th>
</tr>
</thead>
<tbody>
<tr>
<td>Frm files</td>
<td>X</td>
<td></td>
<td>Disk available to the database node</td>
</tr>
<tr>
<td>Public Configuration File</td>
<td></td>
<td>X</td>
<td>On the nodes of Volume 1</td>
</tr>
<tr>
<td>Private Configuration File</td>
<td>X</td>
<td></td>
<td>Disk available to the node</td>
</tr>
<tr>
<td>User Data</td>
<td></td>
<td>X</td>
<td>On each storage node</td>
</tr>
<tr>
<td>Log Files</td>
<td></td>
<td>X</td>
<td>On the nodes of Volume 1</td>
</tr>
<tr>
<td>Registry Files</td>
<td></td>
<td>X</td>
<td>On the nodes of Volume 1</td>
</tr>
</tbody>
</table>

Data Load

During data loads, the load transaction may contain millions of records. Large transactions may have the following impact:

a) Large memory may be needed to represent the uncommitted data.
b) Large disk space may be needed for the log files.

To minimize the usage of internal memory and disk space, as well as to get the best performance for the load, one or all of the following settings may be used:

a) Issue a write lock on the table being updated.
b) Set a value for scaledb_autocommit. for example:

```
scaledb_autocommit = 1000000
```
This setting will issue a commit every 100,000 DML operations.

When setting autocommit value, the following needs to be considered:
Auto-commit is a system-wide definition. Therefore, it should be set only if no other very large user transaction is used. If the auto-commit is set to 100,000, then if a different user transaction is such that it has 100,000 uncommitted rows, it will be automatically committed, which is not a desired operation. At the end of the load disable auto-commit.

If a user transaction is operating with auto-commit (for example, MySQL can set auto-commit for every dml operation), the ScaleDB value for auto-commit has no impact as MySQL will issue a commit.

Auto-commit is set as follows

```
scaledb_autocommit = 100000
```

Auto-commit may be disabled as follows:

```
scaledb_autocommit = 0
```

**Limits on ScaleDB Tables**

ScaleDB treats columns with fixed length and columns with variable length differently. Fixed length columns are columns with the following data types: numeric types, date and time, string types which are not variable length.

Variable length columns are columns defined as varchar, varbinary, text and blobs. The fixed length columns are contained in rows of fixed length. These rows are placed in data blocks and stored in a file (or several files depending on the data size and the configuration). The variable length columns may be stored with the table rows or separately in an overflow file.

These are the limits on ScaleDB tables:

1) A row without variable length fields is limited to 8100 bytes.
2) The size of a variable length column is limited to 2 GB.
3) The number of user databases is limited to 32K.
4) The number of tables in each database is limited to 32K.
5) The number of partitions in each table is limited to 32K.
6) The number of indexes per database is limited to 32K.
Configuration Files:

MySQL Configuration file:

To leverage ScaleDB as the storage engine, the main MySQL configuration file (for example $HOME/mycluster/my.cnf) is updated as follows:

*Set ScaleDB as the default storage engine*

It is recommended to set the default storage engine to ScaleDB (or else specify ScaleDB on the table create statements).

```
default-storage-engine=ScaleDB```

*Set isolation level to Read Committed*

It is required to set isolation level to READ_COMMITTED

```
transaction-isolation = READ_COMMITTED```

*Disable Query Cache*

It is required to disable Query Cache as different database nodes in the cluster may modify the data.

```
query_cache_size=0```

The ScaleDB configuration files:

The main configuration file that determines the setup of the nodes in the cluster is called the Public Configuration File. This file needs to be available on the storage nodes of Volume 1.

This file is retrieved by all the nodes of the cluster on startup. If a particular setup needs to be changed for a specific node, this setup is defined in a Private Configuration File, available only to the specific node.

When the cluster starts, the public configuration file (i.e. `scaledb_global.cnf`) becomes available to every connected node. Then, the node updates the needed parameters with the definitions on its private file - for each particular node that maintains a private configuration file, the information in the private configuration file overwrites the parameters in the public configuration file.

A cluster starts with initialization of the storage nodes. The configuration of CAS1 – main and CAS1 – mirror is available in the public configuration file. For the other storage nodes, their configuration is available in their private configuration file. Then, the cluster manager nodes join the cluster. The private configuration file of the cluster manager nodes includes the port and ip of CAS1 (main and
mirror). Therefore, the cluster manager nodes are able to receive the public configuration and join the cluster. Once the cluster managers are connected, database nodes can join. Each database node connects to the CAS1, receives the public configuration file and joins the cluster.

**Configuration files and connecting nodes to the cluster:**

For each database node, node-specific settings can be specified in the private configuration file (for example $HOME/mycluster/mysql_data/scaledb.cnf)

The private configuration file of a database node (i.e. scaledb.cnf) needs to show the IP and port of CAS1 such that the public configuration file can be retrieved. Other parameters specified in the private configuration file will overwrite the parameters in the public file.

The following is an example of a configuration file for a database node:

```plaintext
scaledb_cas_config_ips = 127.0.0.1
scaledb_cas_config_ports = 14000
```

These define the IP and port of CAS1 – main and allow the database node to receive the public configuration file.

In addition, the private configuration file of the database node in this example includes the following:

```plaintext
scaledb_buffer_size_index = 5M
scaledb_buffer_size_data = 15M
scaledb_buffer_size_blob = 15M
scaledb_hash_lookup_size = 20M
```

These definitions overwrite the definitions for the cache size defined in the public configuration file.

For the cluster manager, the private configuration file (for example $HOME/mycluster/slm.cnf) is specified in the command line and needs to include the following:

The IP and port that is used to connect to the cluster manager:

```plaintext
slm_ip = 127.0.0.1
slm_port = 45001
```

The IP and port of CAS1 – main:

```plaintext
scaledb_cas_config_ips = 127.0.0.1
scaledb_cas_config_ports = 14000
```

Note: The cluster manager is also called star lock manager (SLM).
### Starting a Cluster

The following is a step-by-step guide to starting a ScaleDB cluster. The example configuration files are for a cluster with one volume with primary and mirror storage nodes, two SLMs, and two database nodes.

#### Download and Install the ScaleDB Distribution

Download ScaleDB distribution package (ScaleDB Cluster for MariaDB 5.5.30) from www.scaledb.com/supportsuite and copy it to the home directory of each cluster node (CAS, SLM, and database nodes). Then unpack it in your installation directory on each node. For this example, we will create and use $HOME/mycluster:

```bash
mkdir $HOME/mycluster
tar xzf $HOME/scaledb-latest-mariadb-5.5.30.tar.gz -C $HOME/mycluster/
cd $HOME/mycluster
ln -sf scaledb-latest-mariadb-5.5.30 scaledb
```

# This creates a symbolic link called “scaledb” for convenience

#### Start Cache Accelerator Servers (CAS)

Each CAS instance is initiated with a script that references the CAS configuration file, which specifies the location of the public configuration files, as well as an IP and port used for connection.

Create cache accelerator server (CAS) configuration files. Duplicate CAS configuration file on every CAS machine. These should be identical.

Example: $HOME/mycluster/cas.cnf

Note: Replace all instances of /home/bob with your home directory and write in IP addresses.

```bash
# # Make sure data and logs are highly available #
scaledb_data_directory = /home/bob/mycluster/scaledb_data
scaledb_db_directory test = /home/bob/mycluster/scaledb_data/test
scaledb_log_directory = /home/bob/mycluster/scaledb_logs

# # The more memory the better, but over-allocation causes system to hang #
scaledb_buffer_size_index = 10M
scaledb_buffer_size_data = 30M
scaledb_buffer_size_blob = 30M
scaledb_cas_server_ips = <primary CAS IP>
scaledb_cas_server_ports = 13306
scaledb_cas_mirror_ips = <primary mirror CAS IP>
scaledb_cas_mirror_ports = 13306
```
Run the following script on the machine running the primary CAS:

```bash
bash $HOME/mycluster/scaledb/scripts/cas_run -f $HOME/mycluster/cas.cnf
```

On the machine running the mirror CAS, include the `–m` option:

```bash
bash $HOME/mycluster/scaledb/scripts/cas_run -m -f $HOME/mycluster/cas.cnf
```

When a storage node is initiated, it waits for connections from database nodes and the cluster manager node.

When a database node or a cluster manager node is connected to the cluster, it requests the public configuration file from CAS1. The private configuration file ($HOME/mycluster/scaledb_plugin.cnf) provides the information necessary to connect to CAS1.

If CAS fails to start, make sure:
- the port specified in cas.cnf is open
- data and log directories exist with read and write permissions
- index and data buffers are not bigger than available RAM

### Start Cluster Manager (also known as Star Lock Manager or SLM)

Create cluster manager/star lock manager (SLM) configuration file for each cluster manager. Make sure to write in the IP addresses of your nodes.

**Example (SLM 1):** $HOME/mycluster/slm.cnf

```
slm_ip = <SLM 1 IP>
slm_port=43306
scaledb_cas_config_ips = <primary CAS IP>,<primary mirror CAS IP>
scaledb_cas_config_ports = 13306,13306
```

**Example (SLM 2):** $HOME/mycluster/slm.cnf

```
slm_ip = <SLM 2 IP>
slm_port=43306
scaledb_cas_config_ips = <primary CAS IP>,<primary mirror CAS IP>
scaledb_cas_config_ports = 13306,13306
```

Note: The only difference between SLM configuration files is the slm_ip.
Each cluster manager instance is initiated with a script that references slm.cnf, which specifies the location of the private configuration files. The private configuration file provides the IP and port of the first storage node (CAS1):

```
bash $HOME/mycluster/scaledb/scripts/slm_run -f $HOME/mycluster/slm.cnf -v
```

**Start Database Node**

Database nodes are initiated with MySQL and ScaleDB as the storage engine. The private configuration file of the database node needs to include the IP and port of the first storage node (CAS1).

1. Create MySQL configuration file on each MySQL database node. This configuration file should be identical on each of the nodes.

   Example: $HOME/mycluster/my.cnf

   Note: Replace all instances of /home/bob with your home directory.

```
# # $HOME/mycluster/my.cnf
#
[mysqld]
plugin_dir=/home/bob/mycluster/scaledb/x86_64
transaction-isolation = READ-COMMITTED
query_cache_size=0
datadir=/home/bob/mycluster/mysql_data
basedir=/home/bob/mycluster/mysql
```

2. Create ScaleDB plugin configuration file on each database node and write cluster connection information. This configuration file should be identical on each of the nodes. Make sure to write in the IP addresses of the nodes your primary CAS and mirror CAS are running on.

   Example: $HOME/mycluster/scaledb_plugin.cnf

```
scaledb_cas_config_ips = <primary CAS IP>,<primary mirror CAS IP>
scaledb_cas_config_ports = 13306,13306
```

3. Download MariaDB distribution:

   Go to MariaDB website: [https://downloads.mariadb.org/mariadb/5.5.30/](https://downloads.mariadb.org/mariadb/5.5.30/)
   and download mariadb-5.5.30-linux-x86_64.tar.gz

   Move MariaDB distribution into your installation directory ($HOME/mycluster in this example)
4. Extract distribution into $HOME/mycluster:

```
cd $HOME/mycluster
tar xzf $HOME/mycluster/mariadb-5.5.30-linux-x86_64.tar.gz
ln -sf $HOME/mycluster/mariadb-5.5.30-linux-x86_64 $HOME/mycluster/mysql
# Symbolic link for convenience
ln -sf $HOME/mycluster/scaledb/x86_64 $HOME/mycluster/scaledb/lib
```

5. Initialize MySQL data directory:

```
cd $HOME/mycluster/mysql
scripts/mysql_install_db --defaults-file=$HOME/mycluster/my.cnf
```

This is a script that creates MySQL system tables.

6. Verify mysql server runs successfully:

```
$HOME/mycluster/mysql/bin/mysqld --defaults-file=$HOME/mycluster/my.cnf --console --gdb
```

It should return no errors.

Exit mysql server with "Ctr-C"

7. Link scaledb.cnf to MySQL data directory, so plug-in can find it:

```
ln -s $HOME/mycluster/scaledb_plugin.cnf   $HOME/mycluster/mysql_data/scaledb.cnf
```

8. Start MySQL server:

```
export LD_LIBRARY_PATH=$HOME/mycluster/scaledb/lib
cd $HOME/mycluster/mysql/
bin/mysqld_safe --defaults-file=$HOME/mycluster/my.cnf  >&/dev/null &
```

9. Load ScaleDB Interface Plugin (ha_scaledb.so), which pulls ScaleDB Engine Library (libscaledb.so):

```
$HOME/mycluster/mysql/bin/mysql -uroot -e "INSTALL PLUGIN scaledb SONAME 'ha_scaledb.so'"
```

This activates ScaleDB as a storage engine for MySQL. This step may a minute.

10. Connect to MySQL:

```
$HOME/mycluster/mysql/bin/mysql -uroot test
```

Note: Elevated privileges are not required to run any of the cluster processes.

In order to make ScaleDB the default storage engine, add the following line to my.cnf on each database node:

```
default-storage-engine=ScaleDB
```

Note: This change will not take effect until MySQL server restarts.
Testing the cluster status

To verify ScaleDB cluster is working, connect to MySQL on each node and execute following statements:

Node1 mysql:
create table test.x (n int primary key) engine=scaledb;
Expect: No errors on execution

Node2 mysql:
show create table test.x;
insert into test.x values (1),(2);

Node3 mysql:
show create table test.x;
insert into test.x values (3),(4);

Expect: Table definition shown correctly and no errors on table insert

Node1 mysql:
select * from test.x;
Expect: 4 records shown.

Connection parameters

The final configuration for each node is composed of the parameters in the public file, which are added, for each specific node, to the parameters in its private file. Whenever a value needs to be different on a particular node, it will be specified for the particular node on the private file.

For example, the default cache size for all nodes is in the public file, and specific nodes may be configured with a different cache size in their private files.

The storage nodes of Volume 1 store the public configuration file, and do not have private configuration files. Other storage nodes have private configuration files.


**Examples of Cluster Configurations:**

Below is an example of connection information for a cluster with one Storage Node, Cluster Manager and any number of Database Nodes, where
192.168.10.1 is IP address of the machine running the CAS and
192.168.10.2 is IP address of the machine running the Cluster Manager (SLM):

Storage Node (cas.cnf):
```
scaledb_cas_server_ips       = 192.168.10.1.
scaledb_cas_server_ports    = 13000
```

Cluster Manager Node (slm.cnf):
```
slm_ip                       = 192.168.10.2
slm_port                     = 44001
scaledb_cas_server_ips       = 192.168.10.1
scaledb_cas_server_ports    = 13000
```

Database Nodes (scaledb.cnf):
```
scaledb_cas_server_ips       = 192.168.10.1
scaledb_cas_server_ports    = 13000
scaledb_cluster_port        = 3306
```

Below is an example of connection information of a cluster with any number of Database Nodes, two Cluster Managers (primary and standby), one storage node with a mirrored node, where:
192.168.10.1 is the IP address of the machine running the primary Storage Node,
192.168.10.10 is the IP address of the machine running the mirror Storage Node,
192.168.10.2 is the IP address of the machine running the primary Cluster Manager and
192.168.10.20 is the IP address of the machine running the standby Cluster Manager:

Primary and Mirror Storage Node (cas.cnf):
```
scaledb_cas_server_ips       = 192.168.10.1
scaledb_cas_server_ports    = 13000
scaledb_cas_mirror_ips      = 192.168.10.10
scaledb_cas_mirror_ports    = 13001
```

Primary and stand-by Manager Node (slm.cnf):
```
slm_ip                       = 192.168.10.2
slm_port                     = 44001
scaledb_cas_server_ips       = 192.168.10.1
scaledb_cas_server_ports    = 13000
scaledb_cas_mirror_ips      = 192.168.10.10
scaledb_cas_mirror_ports    = 13001
```

Database Nodes (scaledb.cnf):
```
scaledb_cas_server_ips       = 192.168.10.1
scaledb_cas_server_ports    = 13000
scaledb_cas_mirror_ips      = 192.168.10.10
```
Shut down

To shut down the cluster, execute the following commands.

1. shutdown all mysql server processes:

   mysqladmin -uroot shutdown

2. kill cluster manager

   killall -9 scaledb_slm

3. Use the Linux nc (Netcat) to communicate with the service port of each of each storage nodes with the following instruction:

   echo DO31 | nc [storage node ip address] [storage node port]

These steps must be performed in the specified order, otherwise CAS will refuse to disconnect from any running MySQL nodes.

Clean-up Procedures for Testing

ScaleDB generates multiple data and index files as well as log files. The log files represent the list of updates done in each node as well as the transactional state of the nodes. Using the log files without the related data and index files (or vice versa) can lead to unexpected results. During system tests, the user may want to restart the system with empty files. In that case, the user can issue a truncate command. Alternatively, the user can kill all cluster processes and delete data files from the file system. For fresh restart - the following directories, whose locations are specified in my.cnf, need to be cleaned:

- **ScaleDB data directories**
  Delete all ".registry" files from ScaleDB data directories ($HOME/mycluster/scaledb_data) on the CAS nodes.

- **ScaleDB database directories**
  Delete all files in ScaleDB database directories ($HOME/mycluster/scaledb_data/test) on the CAS nodes.

- **ScaleDB log directories**
  Delete all files in ScaleDB log directories ($HOME/mycluster/scaledb_logs) on the CAS nodes.

- **MySQL data directories**
MySQL data directories specified in my.cnf datadir on the database nodes.

When removing data from the mysql_data directory, make sure not to delete configuration files and the mysql directory where MySQL stores its administrative information.

**Troubleshooting**

*Plugin doesn’t load:*

Verify that ha_scaledb.so is in the MySQL plugin directory ($HOME/mycluster/scaledb/lib/)
Verify that there is a link to the ScaleDB configuration file in the MySQL data directory ($HOME/mycluster/mysql_data/scaledb.cnf) (See step 5)

*Cluster doesn’t start:*

Make sure the connection information (IP Addresses and Ports) between the Database nodes, the Cluster Manager and the Storage nodes (CAS) are correct.

*Cluster doesn’t start after failure or improper shutdown:*

The new Cluster Manager waits to receive connections from all the nodes that were connected before the failure. In order to start the cluster, start all the nodes that were connected prior to the failure.

*A database node does not identify a table created on a different node:*

Select some data from the table. This DML will trigger a table discovery process that will make the table available to the node (the frm file will be retrieved from the shared storage and be available to the MySQL server instance).

*Drop database fails because of constraint violations in the database tables:*

When MySQL drops database, it does not honor child-parent relation in that database tables. ScaleDB does not support removing parent table prior to removing child. However workaround can be done with simple shell script that deletes all tables from database until no tables left and then drops database:

```bash
while [ ! -z ""mysql -NBA -uroot mydb -e 'show tables'" ];do
  for table in `mysql -NBA -uroot mydb -e 'show tables'`
    do mysql -uroot mydb -e "drop table $table" >&/dev/null
  done
done
mysql -uroot -e 'drop database mydb'
```

*Drop table or alter table returns an ambiguous ‘unknown table’ error:*

If you drop a table using the mysql client and the table is either, currently in use by another mysql node, or the table has previously been dropped by another node then you might receive an
'unknown table' error. In this case you should issue a SHOW WARNINGS command to find the reason the operation failed. Alternatively you can start mysql using the --show-warnings option.

```sql
mysql> drop table t9;
ERROR 1051 (42S02): Unknown table 't9'
mysql> show warnings;
```

<table>
<thead>
<tr>
<th>Level</th>
<th>Code</th>
<th>Message</th>
</tr>
</thead>
<tbody>
<tr>
<td>Error</td>
<td>149</td>
<td>Deadlock found when trying to get lock; try restarting transaction</td>
</tr>
<tr>
<td>Error</td>
<td>1051</td>
<td>Unknown table 't9'</td>
</tr>
</tbody>
</table>

2 rows in set (5.75 sec)

You cannot drop or alter a table on one node if it is open on another node. You will receive a 'Deadlock found trying to get lock ... error'.

To avoid this error you need to close the table. This can be done by closing the client that has the table open (e.g. exit the mysql client), or you can issue a FLUSH TABLES command against the mysql node that currently has the table open.

```
#ON node 1.
mysql> create table t7 (a int);
Query OK, 0 rows affected (0.19 sec)
mysql> select * from t7;
Empty set (0.04 sec)

#ON node 2, alter will fail because node 1 has table t7 open
mysql> alter table t7 add b int;
ERROR 1213 (40001): Deadlock found when trying to get lock; try restarting transaction

#ON node 1.
mysql> flush tables;
Query OK, 0 rows affected (0.01 sec)

#ON node 2, alter will now work.
mysql> alter table t7 add b int;
Query OK, 0 rows affected (0.22 sec)
```
Performance Tuning

Many factors impact performance. The list below provides information on setup and parameters that may have significant impacts on performance.

Locality

Locality means separating your database requests such that similar database requests are routed to the same database node. By sending similar requests to the same node, you are taking advantage of the “location” of the data to impact and improve performance:
(1) If similar request are executed at the same node it is more likely that the data is available on the local cache of the node.
(2) It reduces the cluster messaging and data shipping between nodes.

Sending and receiving messages and data between nodes in the cluster are expensive processes. Operations running inside a single server are measured in nanoseconds, whereas network messages may run in the milliseconds. Sending similar queries to the same node eliminates the overheads and may have substantial impact on performance.

We recommend that you setup the cluster, if possible, such that it would increase locality of data and reduce contention between nodes.

For example, if you have a database comprised of 2 nodes that manages information for 10 branches of a particular company: If queries for branches 1 to 5 could be sent to server A, and queries for branches 6 to 10 could be sent to server B, performance would be significantly better than randomly sending queries to any server for two main reasons:

a) The needed data is more likely to be found in the local cache of the database node with locality;
b) The dedicated node is more likely to have the permissions to read and update the data without interacting with the cluster manager and eliminating the need to wait for the other node in cases where it already has a lock on the requested data.

By leveraging data locality, you reduce contention between nodes (both trying to lock the same data, resulting in wait states), reduce data shipping (sending data back and forth between nodes), the Cluster Manager can allocate certain locking rights to the specific nodes, reducing messages between the Database nodes and the Cluster Manager, and it improves the odds of getting the data from cache (fast) instead of the disk (slow). This collection of issues can have a significant impact on performance. In fact, with data locality, you can achieve the performance profile of a shared-nothing cluster, while maintaining the advantages of a shared-disk cluster: namely high-availability, no partitioning, nodal elasticity, fast joins, and more. Exploiting locality makes a huge difference in performance in most situations.
Fast Network Interconnects

A cluster database connects different database and storage nodes via network interconnects. Fast network interconnects are essential to achieve high-performance. The cluster interconnects for large databases need a high-bandwidth (preferably 10 Gigabit or more) network.

Network Packet Size

The network packet size is a configuration parameter that determines the amount of memory allocated for each configured network connection. This parameter is hardware/network dependent and may have big impact on performance. It is suggested to try different setups to optimize performance for your application/network.

TCP or UDP

UDP configuration is a better approach when mirrored storage is used. With UDP updates are multi-casted to the appropriate servers whereas TCP will send each update twice. Note that currently Amazon does not support multi-cast (UDP).

CAS Threads

These are the number of threads dedicated on each database node to communicate with the storage nodes. It impacts performance when data is transferred frequently to and from the storage nodes. If the CPU utilization and I/O on the CAS are high, you should try increasing the number of threads used by the CAS.

The following is an example of configuring the database nodes to use 8 threads to communicate with the storage nodes:

```
scaledb_io_threads = 8
```

Note: The default value for the number of IO threads is 4.

Cluster Manager Threads

These are the number of threads dedicated to transferring messages to and from the Cluster Manager. It impacts performance in situations where nodes experience a high level of contention. Increasing the number of threads may help to achieve higher performance in a high contention environment.

The following is an example of configuring the database nodes to use 4 threads to communicate with the cluster manager:
scaledb_sim_threads = 4

Note 1: The default value for the number of sim threads is 2.
Note 2: The number of sim threads needs to be a power of 2.

**Indexing threads**

Some special insert scenarios are able to leverage multiple threads for fast updates of index files. These scenarios include data load and extended inserts (when multiple insert values are specified with a single SQL Insert statement).

The following is an example of configuring the database nodes to leverage up to 4 threads for updates of index files:

scaledb_index_threads = 4

Note: the default value is scaledb_index_threads = 4.

**OS File Handles**

When a storage node opens a file, it allows concurrent IOs by multiple threads. By default, 2 file handles are available for threads per each physical file.

The following is an example of configuring the storage nodes to provide 4 file handles per each physical file:

scaledb_max_file_handles = 4

Note: The default value for the number of file handles per file is 2.

**Backup the ScaleDB Database**

There are multiple methods to backup a ScaleDB database:

**Backup using MySQL dump**

The `mysqldump` client is a utility that performs logical backups, producing a set of SQL statements that can be run to reproduce the original schema objects, table data, or both. It dumps one or more MySQL database for backup or transfer to another SQL server.
Backup by copying the data files after shutdown

If the cluster is not running, (see section shutdown for proper shutdown of a cluster), the data files on all storage nodes needs to be copied. If the system was properly shut, the log files are not required for this backup.

Backup while system is running (hot backup)

There are two modes in which hot backup can be performed:

a) If no writes are being executed, perform a flush on each of the storage nodes and copy the data files of all the storage nodes.

To get a consistent backup, stop the server or lock and flush the relevant tables:

On one database node of the cluster lock the tables to backup (issue a lock for read):

```
LOCK TABLES tbl_list READ;
```

The read lock enables other clients to continue and query the tables while the data files are copied. The lock will succeed only after all running transactions are committed and the updated version of the data and the indexes is stored on the storage nodes.

Make sure that all storage nodes flushed the data before initiating the copy. To flush data on a particular node use the following Linux nc (Netcat) command:

```
echo DO37 | nc [storage node ip address] [storage node port]
```

After the flush, copy the ScaleDB files from all the storage nodes.

b) If data is being written, hot backup can be supported using Linux Logical Volume Manager (LVM).

To get a consistent backup, perform the following operations:

1. On one of the database nodes, lock the tables for read. In order to guarantee data integrity, the database must remain locked until the LVM snapshot is created.

```
LOCK TABLES tbl_list READ;
```

2. Perform a flush of the data to disk on each one of the storage nodes. To flush data on a particular node use the following Linux nc (Netcat) command:

```
echo DO37 | nc [storage node ip address] [storage node port]
```
3. Create the LVM snapshot on each of the storage nodes. This snapshot needs to be large enough to accommodate the changes that will be made to the database while the snapshot is present.

4. Release the read lock so that normal database operation can resume.

```
UNLOCK TABLES
```

At this point, there is a consistent, point-in-time snapshot of the ScaleDB files stored in the LVM snapshot.

5. Backup the scaledb files from each of the storage nodes.

**Note:** Backup using LVM requires the ScaleDB log files as part of the backed up data. A user that would set a database from the LVM backed data without the log files may have inconsistent and corrupted tables data.

---

**Not supported features**

The following describes features which are currently not supported. These features will be supported in later releases.

**The MySQL frm files**

When a table is created, MySQL creates an frm file containing (internal to MySQL) meta data. ScaleDB does not use the frm file. However, ScaleDB keeps a copy of the frm file in the shared storage. When a node joins the cluster, the frm file is not available to the node. Therefore, `show table`, `describe table` requests by a user will not show info for the particular table. However, a DML request for the particular table will trigger a process that will make the frm file available to the new database node. After the first DML over the table, the frm file will be available to the MySQL server and requests such as `show table`, `describe table` would be satisfied by the new MySQL node.

If a table was dropped on a node of the cluster, the frm file (of the dropped table) on different nodes of the cluster are not deleted. They need to be manually deleted. This problem will be resolved at a later version.

**Isolation Levels**

Currently only read committed is supported. Each query executed by a transaction sees only committed data. The read committed transactions use row-level locking and a transaction waits (for
a commit or rollback of a second transaction) if the needed row is updated by an uncommitted concurrent transaction.

**Delete Cascade**

Delete cascade currently not supported. The delete cascade action specifies that when rows containing referenced key values are deleted, all rows in child tables with dependent foreign key values are also deleted. Currently, ScaleDB enforces referential integrity on deletes. For example, if a primary key value is referenced by a value in the foreign key, the referenced primary key value cannot be deleted.

**Self-Referential Integrity Constrains**

Self-referential integrity constraints are not supported. This is a private case of referential integrity constrains where a foreign key references a parent key in the same table. There are no limits on foreign key that references a parent key in a different table.