SCALEDB TECHNICAL OVERVIEW

The technology behind ScaleDB's highly-available, elastic scale MySQL clustering solution

Introduction

ScaleDB transforms a single database instance (such as MySQL or MariaDB) to a cluster of database and storage services that provides scalability and high availability features that exceed the capabilities of a single database instance. With ScaleDB, you can scale applications to meet increasing data processing demands without changing the application code. As you add resources such as database nodes or storage nodes, ScaleDB extends the processing powers of these resources beyond the limits of the individual components. ScaleDB lowers the overall cost of ownership more effectively than sharding or partitioning. This is due in great part to the single-system image provided by a cluster of database nodes that operates over shared data. By simply adding nodes to your cluster, insertion rate, processing power and storage all scale, enabling simple and elastic scaling. ScaleDB uses a shared-data architecture, similar to Oracle RAC and IBM's IMS. However, ScaleDB runs on commodity hardware—either on-premise or in the cloud – costing a fraction of these other options. By avoiding sharding, and running as a single logical database, your DBA can manage the largest data sets with ease, therefore, reducing your operational expenses considerably. ScaleDB seamlessly scales your application to handle modern application demands while delivering superior performance and scalability.

Overview

ScaleDB transparently transforms a single database instance of MariaDB or MySQL into a cluster where multiple database instances process shared-data. By using a cluster, scaling is supported by dynamically adding database and storage resources without the need to partition or shard the data. The cluster provides high-availability as there is no single point of failure (SPOF).
Since all database instances access the same dataset, the overall system must guarantee the coordination of data changes on different database instances such that whenever a database instance queries data, it receives the current version – even if another instance modified that data. ScaleDB maintains this functionality by supporting locking at a cluster level and alerting database nodes when data is modified by a different node.

As scaling is done by adding resources to a cluster, ScaleDB provides simplicity, flexibility, reduced development and lower maintenance costs. This approach allows businesses to adapt to changing needs by efficiently and seamlessly supporting increasing data volumes, data velocity, and concurrent users.

Main Features & Benefits:

- High performance – ScaleDB efficiently supports large amounts of data and large numbers of concurrent users.
- No need to shard or partition the data – scaling is done by dynamically adding database nodes and storage nodes to a cluster.
- High Availability (HA) – no single point of failure. No need for master/slave replication.
- Simplicity – simplified design and development as there is no need to partition the data and automatic HA.
- ACID compliance – ScaleDB guarantees that the database transactions are processed reliably.
- Supports row-level locking and Foreign Key constraints.
- ScaleDB is a disk based solution. It arranges data on disk and is designed for CPU and IO efficiency while processing large data volumes.
- ScaleDB plugs into MySQL and MariaDB such that all existing applications and tools work without modification.
- Operates on commodity hardware or virtual machines.

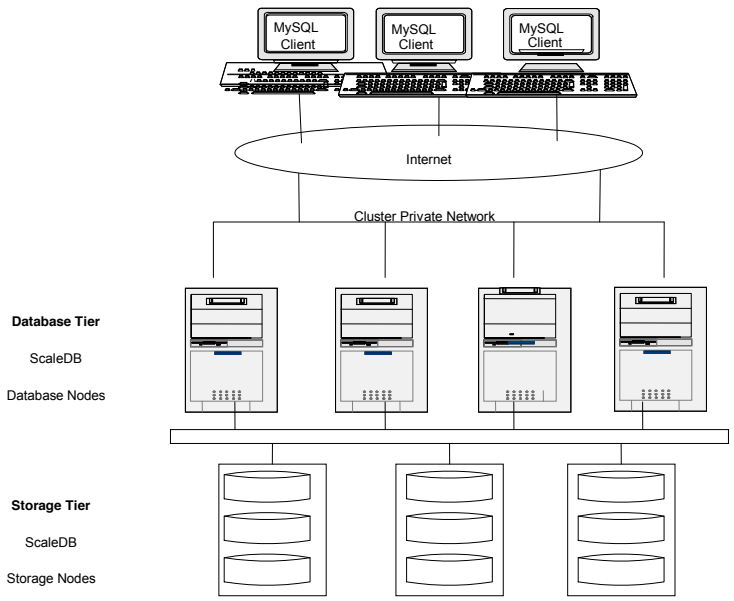
ScaleDB provides a complete "end-to-end" database and storage solution that transparently transforms MySQL/MariaDB into a high-end database. ScaleDB provides a complete and comprehensive platform to support high volume and data intensive applications using low cost commodity hardware or using virtual machines on-premise or in the cloud.

The ScaleDB Architecture

ScaleDB is managing data in 2 tiers, a database tier and a storage tier and is comprised of the following components:

- A collection database nodes connected by a network (the database tier). In the case of MySQL and MariaDB, each database node includes a MySQL/MariaDB server instance that treats ScaleDB as the pluggable storage engine. All the database nodes process shared data that is managed in the storage tier. With ScaleDB, the complete data set is available to each database node and each database node can read and write to the entire data set.
- A storage tier of multiple storage nodes connected to the network (the storage tier). The storage nodes maintain the database data and act as a shared and persistent storage layer for the database nodes.
- A Cluster Manager connected to the network. The Cluster Manager maintains the integrity among processes of different database nodes. As the database data is available to all the database nodes in the cluster, different database nodes may have conflicting processes over the same data at the same time. The Cluster Manager resolves conflicts between different database nodes in the cluster.

The diagram below shows a typical deployment. Applications are connected via the internet to a ScaleDB cluster comprised of multiple database nodes, multiple storage nodes, and a Cluster Manager (not shown in the diagram). The collection of database nodes makes the database tier. The collection of storage nodes makes the storage tier. Each database node includes a MariaDB instance with ScaleDB as the database engine. When a database node needs data, the data is read from the storage tier and when a database node updates data, the updated data is written to the storage tier such that the update is available to the different database nodes in the cluster. The Cluster Manager synchronizes the read and write processes such that only consistent and committed data is available to the database nodes.



Processing Data on a ScaleDB Cluster

As the entire data set, in a ScaleDB cluster, is available to each of the database nodes, every DML (Insert, Update, Delete and Select) can be processed on any database node in the cluster.
With MySQL, DMLs are SQL statements that are parsed by the MySQL instance (on the database node), that trigger a set of low-level calls to process data. These calls are transferred to the ScaleDB instance (on the database node) that processes the request such that data changes are available to all the database nodes in the cluster.
When a table is created by one of the database nodes, it is available to all the database nodes in the cluster. For each table, data can be added, deleted updated and queried by all the database nodes simultaneously while the integrity of the data is maintained at a cluster level – ScaleDB provides the standard ACID-compliant transaction features, along with foreign key support (Declarative Referential Integrity).
ScaleDB supports other databases or applications by offering the DBMS functionality through a native API. With MariaDB, a special process maps the MariaDB storage engine API to the ScaleDB native API.

Database Scalability

Different database architectures have different scale-out profiles. The two primary database architectures are shared-nothing and shared-data. A shared-nothing database, such as MySQL/InnoDB, scales by partitioning the data and splitting that data across multiple independent database servers. Database requests needs to be routed to the specific database that maintains that data. Shared-nothing databases are very rigid; changes in the application, scaling and other changes can force a re-sharding effort, which can be extremely complex and time consuming.

A shared-data database, such as MySQL/ScaleDB, provides a very different method of scaling. In a shared data approach scalability is achieved by adding compute resources and therefore scaling is done without changes to the schema and without the need to move the data. With a ScaleDB cluster, this would mean adding database nodes or storage nodes to the cluster. With the shared data approach, every machine adds additional processing power to the cluster without service interruption or downtime.

Advantages of each approach:

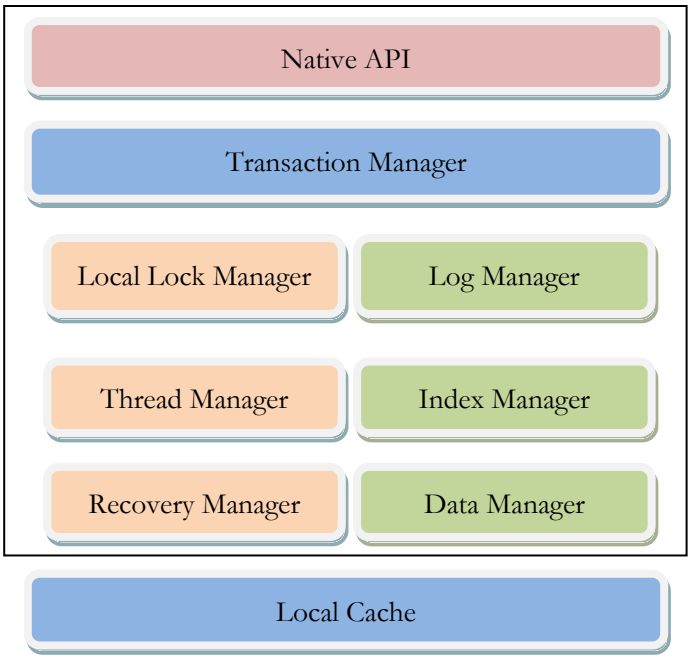
If partitioning is done well, shared-nothing is efficient as each database machine is tuned to perform a subset of database tasks. However, good partitioning is hard and often impossible to achieve. Partitioning breaks the data to achieve even distribution of the data among the database instances. Therefore, it requires assumptions on data growth. In addition, partitioning tries to achieve even distribution of the queries among the different databases. In many cases this is a complicated task and in particular when one tries to estimate not only how queries need to be distributed over a given set of data, but also how the distribution of the queries will change over time. In addition, many queries require data from multiple tables that are not grouped on the same partition. In many cases, partitioning that works well for query A may perform poorly for query B. Many databases include tens, or hundreds or thousands of tables. Tables may be related to other tables resulting in databases that can never be well partitioned.

In a shared disk, the multiple servers access the same physical data and therefore partitioning is not required. The challenge in a shared disk solution is to maintain the integrity between the nodes in the cluster. For example, if node A updates a row while node B, at the same time, needs to retrieve (or update) the same row, the requests of the different nodes need a synchronization mechanism. Since the nodes do not have shared memory, the synchronization task is not trivial. ScaleDB uses patented technology to manage the synchronization efficiently – the ScaleDB software component that manages the synchronization process is referred as the Cluster Manager and is explained below.

The Database Nodes

Each database node leverages MySQL or MariaDB as the DBMS and ScaleDB as the database engine. The database engine executes the MySQL requests to update and retrieve data. Requests for data can be satisfied from the local cache of the database engine. If the data is not available in the local cache, the database engine will retrieve the data from the storage tier. Since all database nodes operate over the entire data set, threads from different database nodes may be operating over the same data at the same time. Conflicts between threads within a single database node are resolved by a local lock manager. Conflicts between requests of different database nodes are resolved by the Cluster Manager.

Below is a diagram of a ScaleDB storage engine on the database node:



The ScaleDB native API receives low-level calls to manipulate the data. Each low-level call is processed by the ScaleDB database engine. The engine is a multithreaded, disk-based, transactional, ACID-compliant engine. It supports row-level locking and operates in read committed mode. ScaleDB leverages efficient indexing. These indexes are not sensitive to key length, have a small footprint and process join operations very efficiently. To maintain data integrity, ScaleDB supports foreign key referential-integrity constraints. Frequently used data is maintained in a local cache on each database node. If the data in a local cache is updated by a different database node in the cluster, a special process invalidates the old copy of the data. When a database node commits, updated data is placed on the storage tier such that it is available to all the database nodes in the cluster.

High-Availability of a Shared-Data Database

With shared-nothing, data is assigned to a particular database server. If the server fails, the system is not available and the data may be lost. Therefore, to achieve HA in a shared-nothing database, the database data is replicated such that if the database breaks, a copy is available. This setup is referred to as a Master and Slave. With MySQL, the master writes events to a log file that updates the slave. However, a Master and

Slave setup is complicated to maintain – when data is updated, both the Master and the Slave need to be updated. The need to update two databases may create complexities. For example, one database accepts the update while the second rejects the update. This approach carries a significant performance penalty as the master needs to wait for the update in the slave. If the shared-nothing database is configured for asynchronous replication, then performance increases at the cost of slave lag and inconsistent data.

With a shared-data approach, HA is transparent and there is no need to maintain slaves – as all database nodes share the same physical data, if a particular database node fails, one of the surviving nodes will undo the uncommitted data of the failed node and the queries are routed to any one of the surviving database nodes.

The Cluster Manager

ScaleDB Cluster Manager is a lock manager that synchronizes lock requests of different database nodes. The Cluster Manager can be considered an extension of the local lock manager within each node – the local lock manager resolves conflicts between threads of the same database node whereas the Cluster Manager resolves conflicts between processes of different database nodes.

In order to enforce coherency of the buffer caches of the nodes in the cluster, the Cluster Manager makes sure that a node acquires a lock at a cluster level before modifying or reading a database block. When a block is modified, it notifies the database nodes in the cluster to invalidate the particular block in their private cache. These processes make the global caches of the different nodes synchronized allowing only one node to modify a block at any single point in time and make all the nodes in the cluster aware of the change. The synchronization protocol in the Cluster Manager tracks the status of resources in the cluster to guarantee the completeness and coherency of the data. Some examples are:

Concurrent Reads on multiple nodes – two nodes need to read the same data block. Reads are done without synchronization as multiple instances can share data blocks for read access without cache coherency conflicts.

Concurrent Reads and Writes on different nodes – a request from a node to read or update a block that is currently modified by another node – the requesting process will be placed on wait for the write process to complete. When the write is completed, the updated block is placed on the storage tier and a buffer invalidate message signals the new process to retrieve an updated copy of the block.

The ScaleDB Cluster Manager uses sophisticated algorithms to make the synchronization process efficient. These algorithms minimize the communications between the database nodes and the Cluster Manager and therefore allow to process data efficiently. This locking approach guarantees that the ACID properties are maintained and queries return only committed and consistent data.

High-Availability of the Cluster Manager

To provide HA, ScaleDB clusters are set with two Cluster Managers. The first is the operational Cluster Manager that operates with the database nodes. The second is in a standby mode. If the operational Cluster Manager fails, the standby Cluster Manager will take over to manage the cluster. When the standby Cluster Manager identifies a failure of the operational Cluster Manager, it will first connect to all the nodes in the cluster. Once all the nodes connect to the standby Cluster Manager, the standby becomes the operational Cluster Manager. This approach enables continual service to the database nodes in the cluster, while maintaining lock status integrity across the cluster. This process is transparent to the applications using the database.

The Storage Nodes

ScaleDB manages the data in block-structured files. Individual files are broken into blocks of a fixed size. These blocks are stored across a cluster of one or more machines. The individual machines that store the data in the cluster are referred to as Storage Nodes. For High Availability, each storage node is mirrored. Therefore, storage nodes are added to a cluster in pairs (main and its mirror). The target machine (and its mirror) for each block is chosen randomly on a block-by-block basis.

This approach has several advantages:

- It supports file sizes far larger than what a single-machine offers.
- The IO calls are partitioned among the storage nodes leveraging resources (CPU, memory, disk drives) that exceed the resources of a single machine.
- Storage Nodes can be added to an existing cluster and therefore the amount of resources supporting the IO calls can be dynamically increased.
- As the data is mirrored, there is no single point of failure.

When a database node needs data that is not available on its local cache, it identifies the storage node that maintains the block containing the data and sends a message to the storage node to retrieve the data. When the data arrives to the database node, it is placed in the cache of the database node and becomes available to the process that requested the data.

The storage nodes keep the data on storage mediums such as disk drives or SSDs. If the RAM available on a storage node is sufficient to keep all the data managed by the particular storage node, a copy of the data will be available in the RAM. Otherwise, frequently requested data blocks will be kept in the cache of the storage node and less frequently used data will be stored on the persistent storage medium.

These nodes make the storage tier and act as a distributed storage system where data is stored in blocks that are placed in one or more storage nodes. The individual storage nodes operate as IO machines: They process requests to retrieve data by locating the data in their cache, if the requested data is not available in their cache; it is read from the storage device, placed in the cache and transferred to the requesting database node. Write requests are performed in a similar way; the updated data is transferred from the database node to the storage node that manages the particular data and replaces the older copy of the data. When a mirror node is present, the data is sent to both – the main node and its mirror. With time Series data queries are pushed from the database nodes to the storage nodes to be executed next to the data. This process delivers performance that is orders of magnitude more efficient than traditional processing. This process is explained in the section “Pushdown Processing” below.

High-Availability of the Storage Layer

The storage tier maintains multiple copies of the data: When a data block is written, it is written to multiple storage nodes in the cluster. To define the setup of the cluster, we use the concept of a volume. A volume is a logical unit that contains data. When a new block is created by a database node, the node randomly picks a volume to manage the data block. This process assigns the data block to a volume and from that point on, every read or update of the block would be managed by the particular volume. With N volumes, every block is available on only one volume. Physically, each volume is supported by one or more storage nodes and a process that writes a block to a volume places the data on all the storage nodes supporting the volume. When a block of data is retrieved, the request is satisfied by one of the storage nodes in the volume.
For example, with 10 volumes and 2 storage nodes per volume, a write of the data to volume 3 will place the data block on the 2 storage nodes that support volume 3.
This approach provides HA – if one of the storage nodes in the volume fails, a different node in the volume continues to serve the IO requests without downtime.

Storage Scalability

To scale the storage tier in a ScaleDB cluster — thereby increasing both data and I/O capacity — the DBA simply adds more storage volumes to the cluster. The cluster automatically recognizes the additional storage volumes and routs IO requests to the new volumes. When new storage volumes are added, new data will be more likely added to the new volumes (rather than to the existing volumes) until data is evenly distributed across all (existing and new) storage volumes. Once data volume parity is achieved, the cluster switches back to random distribution of data. By adding storage volumes to an existing cluster, you expand the pooled cache, disks, storage and processing power of the entire cluster thereby allowing seamlessly scaling of the storage tier.

Push-Down Processing

ScaleDB is using a pushdown mechanism to manage time series data. The Pushdown Mechanism enables certain types of query processing to be done in the storage nodes. With Pushdown technology, the database nodes send query details to the storage nodes. With this information, the storage nodes can take over a large portion of the data-intensive query processing. The Pushdown Mechanism can search storage nodes with added intelligence about the query and send only the relevant bytes, not all the database blocks, to the database nodes.

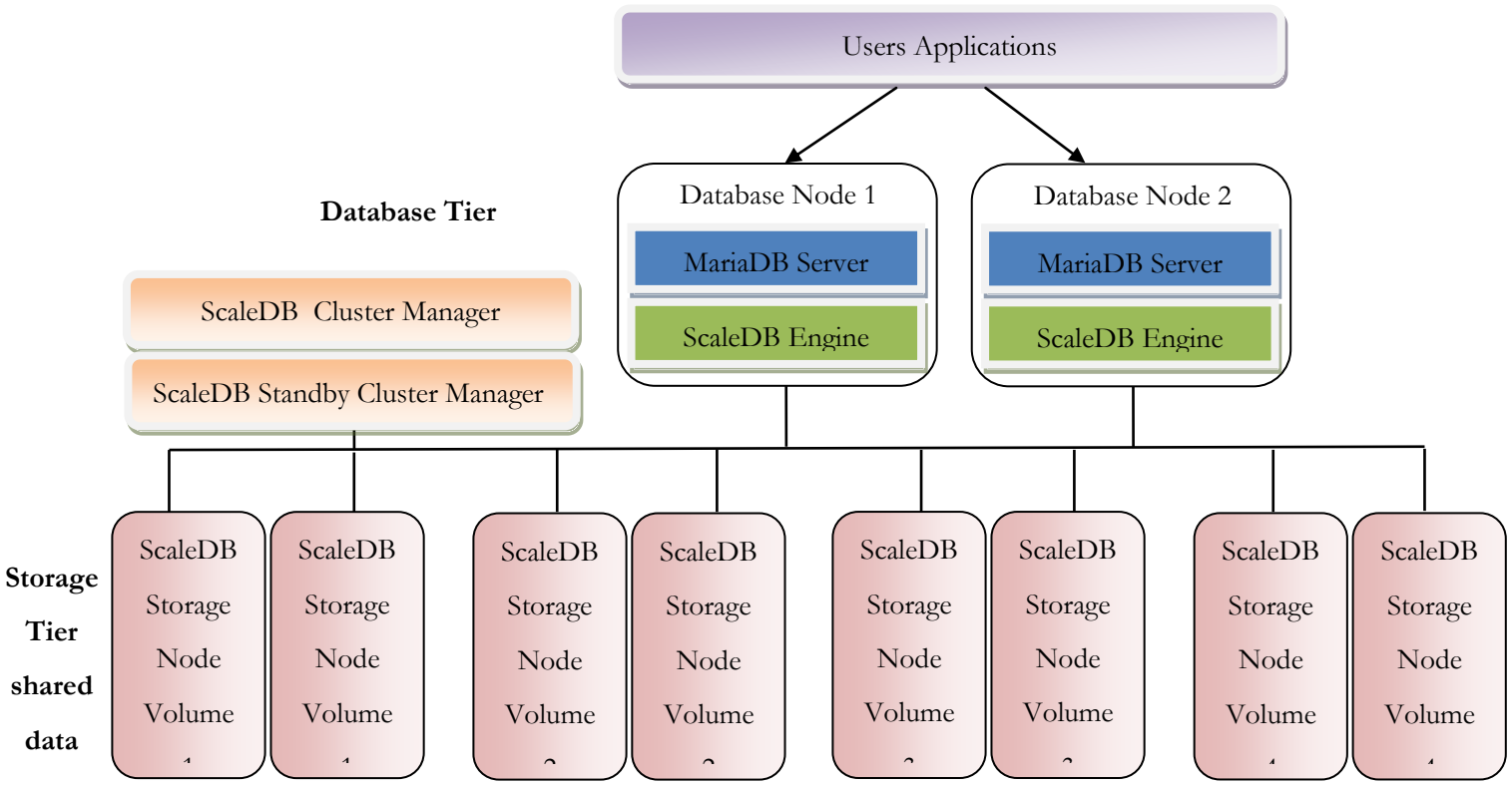
This approach offloads data intensive SQL operations into the storage nodes. By pushing SQL processing to the storage nodes, data filtering and processing occur immediately and in parallel across all storage nodes as data is read from disk. In addition, the storage nodes maintain special indexes and data structures to efficiently add and retrieve time series data allowing millions of inserts per second and extreme query performance.

The performance benefits of this approach are:

1. Parallel processing – Every query is supported by multiple storage nodes. For example, with 10 storage volumes, 10 machines execute the query. Each machine contributes resources (CPU, memory, disk drives) to support the query, each machine only has one tenth of the data to process and all machines work in parallel.
2. Rather than shipping large amounts of data to the database node, the query is executed next to the data and only a small amount of data is passed over the network to the database node.
3. Query performance can increase linearly by adding storage nodes to the cluster.

ScaleDB Cluster Deployment

The diagram below shows a typical ScaleDB deployment. The deployment below includes 2 database nodes, an operational cluster manager, a standby cluster manager and 4 storage volumes with 2 storage nodes supporting each volume.



The cluster diagram above shows the three components of a ScaleDB cluster:

1. The database tier is comprised of multiple database nodes. Each database node is configured with a database instance (such as MySQL or MariaDB). Each database instance interfaces a ScaleDB engine instance. When a database node receives requests to define data or manipulate data, it transfers the requests to the ScaleDB engine that processes the requests in a way that maintains the consistency of the data across the cluster.

2. The storage tier is comprised of multiple storage nodes. The data is organized in blocks which are randomly distributed across the storage volumes. Each volume is comprised of one or more storage nodes. Data from the storage tier is retrieved by a message to a node in the volume that contains the needed data. Updates are done by an update message that is sent to all the storage nodes in the volume that manage the data.

3. Two instances of a Cluster Manager. An operational Cluster Manager and a standby Cluster Manager. If the operational Cluster Manager fails, the standby Cluster Manager becomes operational. The Cluster Manager is a lock manager that resolves conflicts between nodes in the cluster. For example, if two different database nodes in the cluster are trying to update the same resource at the same time, the Cluster Manager will resolve the conflict by allowing the first database node to update the resource and signal the second database node when a second update is possible.

High-Availability of a ScaleDB Cluster

ScaleDB does not have a single point of failure, making it resilient to failure of nodes in the cluster. If a component breaks, the system continues to operate – If a database node fails, a different database node will undo the uncommitted data of the failed node and queries are routed to one of the surviving database node. If the operational Cluster Manager breaks, the standby Cluster Manager identifies the failure and replaces the operational Cluster Manager. If a storage node fails, the data is available on a different storage node in the same volume. This approach provides improved fault resilience. In the event of a system failure, ScaleDB ensures high-availability to the database users and reliable access to mission critical data.

Database Operations in the Cloud

ScaleDB's two-tier shared-data architecture provides a powerful database solution allowing dynamic scalability and High Availability (HA) of the database and storage tiers in the cloud. With ScaleDB you can start with a smaller deployment and increase the size of your database cluster as your business needs grow. Cloud resources are used only when needed and adding and removing resources are dynamic eliminating the need to redesign and shard the database data.

ScaleDB is an easy to deploy solution that delivers the highest levels of database performance with large data sets. ScaleDB delivers outstanding I/O and SQL processing performance for online transaction processing (OLTP), data warehousing (DW) and consolidation of mixed workloads. ScaleDB offers special optimizations for Time Series Data such that a small cluster supports millions of inserts per second and queries evaluating hundreds of millions of rows are performed within few seconds.
On the cloud, database and storage instances are easily configured to replace a single database instance with a scalable cluster of nodes that provide database services.

Using ScaleDB as a cloud based database service, companies will accomplish the following:

- Elastic and high performance database cluster from cloud resources.
- Handle changes and growth in data volumes and users in incremental and dynamic steps.
- Consolidate multiple database deployments to a common infrastructure.
- Deliver data availability for large mission critical applications at low cost.
- Transform a single database instance to a cluster of database and storage instances.

These characteristics provide out-of-the-box scalability and high-availability transparently to the applications transforming a database instance to a dynamic cluster of database and storage nodes in the cloud.

Conclusion

MySQL and MariaDB have achieved widespread adoption because of their ease of use and performance. However, these DBMS are hamstrung by complex and limited scaling, and lack of high-availability. ScaleDB extends these DBMS solutions turning them into a cluster that delivers simple and elastic scaling, high-availability and high-end performance. Because ScaleDB plugs into MySQL and MariaDB, your existing applications and tools simply work without modification. ScaleDB is optimized for commodity hardware and networking, so it is cost-efficient and can be deployed on-premise or in the cloud. If your database requires scalability and high-availability, ScaleDB is the answer.