ScaleDB

ScaleDB Technical Overview

From a Single Database Instance to a Database Cluster in the Cloud

Dec. 12, 2013

Introduction

ScaleDB transforms a single database instance (such as MySQL or MariaDB) into a grid 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.

A ScaleDB platform extends MySQL and MariaDB into a clustered database that is ideally suited for both online transaction processing (OLTP) and data warehouse (DW) applications. ScaleDB uses a shared-data architecture, delivering high-performance and high-availability across a wide range of applications running either on commodity servers or in the cloud. ScaleDB is tuned to support large data sets. It is a disk-based database and storage platform that is deployed as a cluster of nodes connected by a network. It operates at multiple tiers - At the database tier, multiple database operate over shared data. The data is managed at the storage tier by a collection of storage nodes. All database operations are synchronized by a distributed lock manager.

ScaleDB operates on commodity hardware or on the cloud. It enables low-cost hardware to deliver the highest levels of scalability and reliability. It is simple to install and manage and provides a lower Total Cost of Ownership (TCO) for large and complex systems.

Overview

ScaleDB transparently transforms a single database instance into a clustered shared-data database. ScaleDB manages 2 tiers – a database tier and a storage tier. Both tiers are elastic – database resources and storage resources can be added dynamically. The system provides high-availability as there is no single point of failure. ScaleDB provides simplicity and flexibility resulting in reduced design, development, maintenance and administration costs allowing businesses to adapt to changing needs and efficiently support increased data volumes and increase in the number of users.

Main features:

- A clustered database using a shared-data technology—multiple database instances share the same physical data.
- Elastic database tier where database instances can be dynamically added providing dynamic scalability.
- Elastic storage where storage resources can be dynamically added to address large and growing data volumes.
- Providing ACID properties which guarantee that the database transactions are processed reliably.
- Can leverage the cloud or commodity hardware (and advanced hardware) for both – the database nodes and the storage nodes.
- No single point of failure.
- When datasets grow or the number of concurrent users increases, ScaleDB eliminates the need to partition or shard the data.

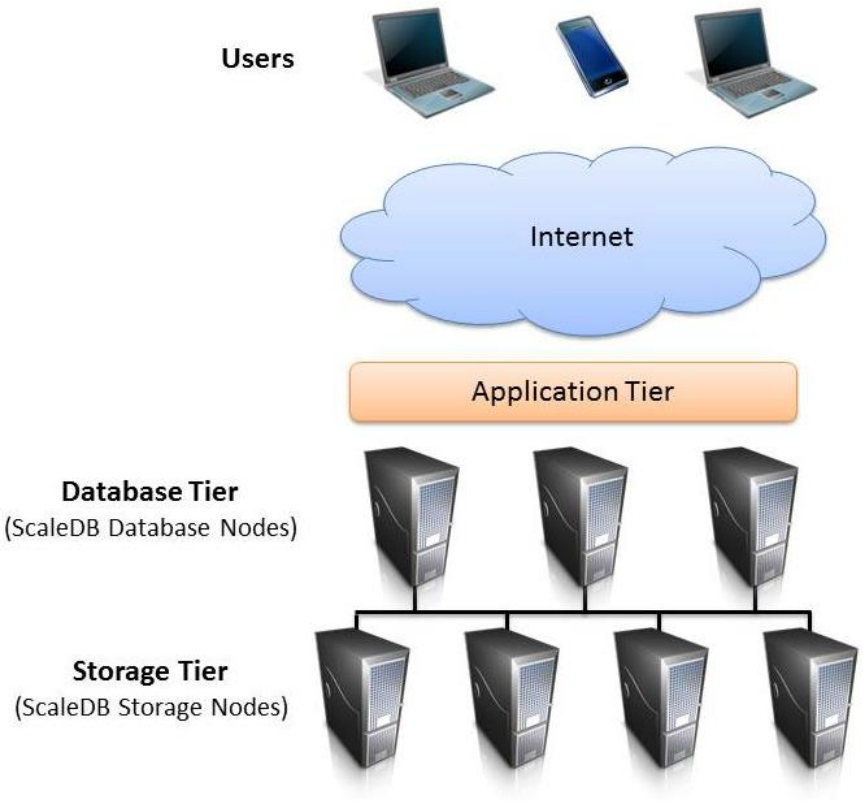
The ScaleDB architecture provides “out of the box” cloud compatibility for new and existing applications. It provides a complete “end to end” database and storage solution that transparently transforms your database to a high end database and storage machine.

The ScaleDB Architecture

The ScaleDB architecture is based on the following components:

- A native API supporting low level database calls providing transactional storage engine functionality. ScaleDB functionality is enabled through the API. In the case of MySQL and MariaDB, an interface layer maps the database calls to the ScaleDB API.
- A database tier of multiple database nodes connected to a network. Each database nodes has a shared read/write access to the physical data that is managed in the storage tier. In the case of MySQL and MariaDB, each database node includes a MySQL/MariaDB server and a ScaleDB Engine.
- A storage tier of multiple storage nodes connected to the network. Each storage node is managed by a ScaleDB storage instance. The storage nodes maintain the data and provide efficient and scalable integrated caching and persistent storage layer.
- A cluster manager instance connected to the network. The cluster manager is a distributed lock manager that maintains the integrity among processes of different nodes of the cluster.

The diagram below shows a typical deployment. Applications are connected via the internet to a particular database node within a cluster. Each of the database nodes includes a database component that is using ScaleDB as the database engine. The collection of database nodes is considered the database tier. The database nodes are connected to storage nodes via the network. Each of the storage nodes is managed by a ScaleDB storage instance. The collection of storage nodes is considered the storage tier and it provides a shared storage for the database nodes. When a database node needs data which is not available in its local cache, it is retrieved from one of the storage nodes of the cluster.



For MySQL users, each database instance is a MySQL server with ScaleDB database engine. For MariaDB users, each database instance is a MariaDB server with ScaleDB database engine. Non MySQL users may interact directly with the ScaleDB database engine API.

The database nodes and the storage nodes can be virtual or physical machines. A database node in the database tier interacts with nodes of the storage tier when needed data is not available on the local cache of the database node or when data is updated and needs to be available to other database nodes in the cluster.

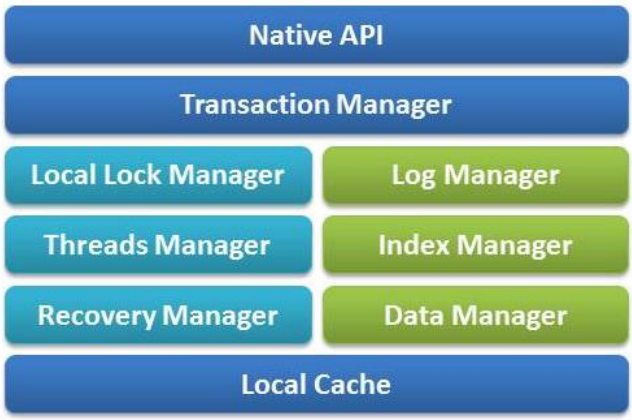
A ScaleDB cluster includes 3 components:

- Multiple database nodes. Each database node is a database instance with ScaleDB as the database engine.
- Multiple ScaleDB storage nodes. The collection of storage nodes forms a global cache and persistency layer.
- A cluster manager. The cluster manager is a ScaleDB management instance (on a dedicated node which can be a virtual or physical machine) in the cluster. The cluster manager resolves conflicts between database nodes in the cluster (the cluster manager is not shown in the diagram above).

The Database Nodes

For MySQL users, each database node is a MySQL server with ScaleDB as the database Engine. The database engine executes the MySQL logical 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 of the cluster. As the data is shared by all the nodes in the cluster, different threads of the same database node as well as threads from different database nodes may operate over the same data at the same time. Conflicts between threads of different database nodes are resolved by the cluster manager.

Below is a diagram of a ScaleDB Database Engine:



ScaleDB Native API receives low level calls to manipulate the data. When a call to manipulate the data is received through the API, it is processed by the ScaleDB database engine. The engine is a multithreaded,

transactional, ACID compliant engine. It supports row level locking and operates in read committed mode.

ScaleDB leverages efficient indexing. It supports 3 types of indexes:

a) Multi Table Indexes – these indexes are general purpose indexes that are compatible to Btree indexes in terms of their functionality. However, these indexes are not sensitive to key length, have a small footprint and efficiently process join operations. Multi Table indexes are unique by being able to efficiently support relational and object views of the data and are used to maintain data integrity by supporting foreign key and referential-integrity constraints.
b) Hash Based Indexes – these indexes are not general purpose indexes – these are non-unique indexes, they support point lookup but not range scans.
c) Time Indexes – these indexes are used for efficient scans of large data sets based on the time of the insert.

ScaleDB database engine maintains, on a local cache of each database node, data and index blocks which are frequently used. If the needed block is not available in the local cache, it is requested from the storage node that maintains the block and placed in the local cache. Conflicting processes of different threads are synchronized by the local lock manager and conflicting processes of different threads of different nodes are synchronized by the cluster manager.

Scalability of a Clustered Database

The main 2 competing methods to scale the database tier are shared-nothing and shared disc. The majority of the database systems are using the shared-nothing approach. SQL Server and MySQL without ScaleDB are examples of shared-nothing databases. Oracle RAC, the IBM IMS (the mainframe database) and MySQL with ScaleDB are examples of databases using the shared-data approach.

In a shared-nothing approach scalability is achieved by partitioning the data. When data volume or the number of users increases, the data is partitioned to multiple database servers. When a user queries the database, the query needs to be routed to the server that maintains the data.

In a shared-data approach scalability is achieved by adding instances of database servers to a cluster (and there is no need to partition or shard the data). With the shared-data approach many databases operate over the same data and every server adds additional processing power to the cluster.

The advantages of each approach are as follows:

If partitioning can be done well, shared-nothing would be very efficient as each database machine needs to be tuned to perform a subset of the potential database tasks. However, good partitioning is hard and many times impossible to achieve: Partitioning needs to break the data such that we get an even distribution of the data among the database nodes. Then, 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 database calls are distributed over a given set of data, but also how the distribution of the calls would change over time and how would the data grow over time. In addition, many queries require data from multiple tables and therefore, the partitioning of the data not only does not help but it adds complexity as a query needs to be executed on multiple servers. For example, to allow scalability, the Customers Table was partitioned to multiple servers. However, many queries are satisfied by joining the customer data with the catalog data. These types of queries raise the question of where would the Catalog Table be placed – on a separate server or multiplied on each of the Customers Servers? If the catalog is multiplied on each server it needs to be updated on each server and

be constantly synchronized with other servers therefore making the system more complex to update and manage. If the catalog is stored once on a separate server, than queries that are satisfied by Customers and Catalog data needs to be executed on 2 servers which add significant overhead and complexity to the system. In addition, many databases include tenths, 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 database, multiple servers can access the same physical data and therefore partitioning is not required. The challenge in a shared-data 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. And as the nodes do not have shared memory, the synchronization task is not trivial. If multiple nodes require the same resources (such as a row or a data block), it adds complexity over a non-cluster system, however, the shared-data approach assumes that although conflicting requests between nodes may trigger contention, most processes do not initiate contention between nodes and the overall computing power of the cluster yields better throughput than a single database.

ScaleDB manage conflicts at 2 levels – at the node level using a local lock manager and at a cluster level by a global lock manager (sometimes referred to as distributed lock manager). The ScaleDB software component that manages the synchronization process is referred as the Cluster Manager.

High-Availability of a Shared-Data Database

With shared-nothing, data is assigned to particular database machine. If the database breaks, 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, the copy is available. This setup is referred to as a Master and Slave setup. With MySQL, the master writes events to a log file which updates the slave. However, Master and Slave setup is complicated to maintain – when data is updated, both the Master and the Slave needs to be updated. The need to update 2 databases may create complexities; for example: one database accepts the update while the second rejects the update. This approach may have performance penalties as the master needs to wait for the update in the slave. With shared-data there is no need to maintain slaves – as multiple nodes share the same physical data, if a particular node fails, one of the surviving nodes will undo the uncommitted data of the failed node and the queries are routed to anyone of the surviving database nodes.

The Cluster Manager

ScaleDB cluster manager is a distributed lock manager that synchronizes lock requests of different 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 node whereas the cluster manager resolves conflicts at a cluster level. ScaleDB cluster manager is using sophisticated algorithms to make the synchronization process efficient. In addition, to minimize the dependency on the cluster manager, the database nodes are capable to determine independently if a particular lock conflicts with other nodes. If a database node decides that a particular lock is not conflicting with a different process in the cluster, the database node processes the lock without dependency with the cluster manager. However, if a conflict occurs, the cluster manager resolves the conflict. ScaleDB processes are such that the ACID properties are maintained and only committed data is returned to the queries.

High-Availability of the Cluster Manager

A ScaleDB deployment is configured with 2 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. This promises no "split brain" scenario where some nodes are connected to the first cluster manager and some are connected to the standby. When all nodes are connected to the standby cluster manager, the standby become the operational cluster manage. This approach allows continual service in case of a failure of the cluster manager. This process is transparent to the user applications.

The Storage Nodes

ScaleDB storage tier is a collection of storage nodes (configured with the ScaleDB storage software) that manages the data. The storage nodes are connected by a network to the database nodes. Data is transferred using a ScaleDB protocol and therefore there is no need for Network File System (NFS). It is completely transparent to the applications and highly fault-tolerant. It designed to be deployed on the cloud or low-cost hardware; it provides high throughput access to the database nodes in the cluster and is suitable for applications that have large data sets.

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 with data storage capacity. Individual machines 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). Each pair is considered a volume. A file can be made of several blocks, and each block is stored twice: on one of the storage nodes and on its mirror. The target machine 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. Storage Nodes can always be added to an existing cluster. The IO calls are partitioned among the storage nodes and leverage the cache and CPU of all the storage nodes concurrently.

When a database node needs data which is not available on its local cache, it identifies the volume that maintains the block containing the data and sends a message to the one of the storage nodes (main or mirror) 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 thread 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 storage medium.

Storage Scalability

ScaleDB storage tier is a distributed storage platform. The data is stored in disk based blocks that are stripped over one or more volumes. A volume may contain multiple nodes (such as a main and a mirror) and each block is placed on all the nodes of the volume. This data in the storage tier is available to all the nodes in the cluster - each of the database nodes is capable to locate the storage node that manages the needed data and request or update the data. 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 block is transferred from the database node to the storage nodes of the particular volume that is responsible to manage the particular block. When a mirror node is present on the particular volume, the data is send to both – the main mode and its mirror.

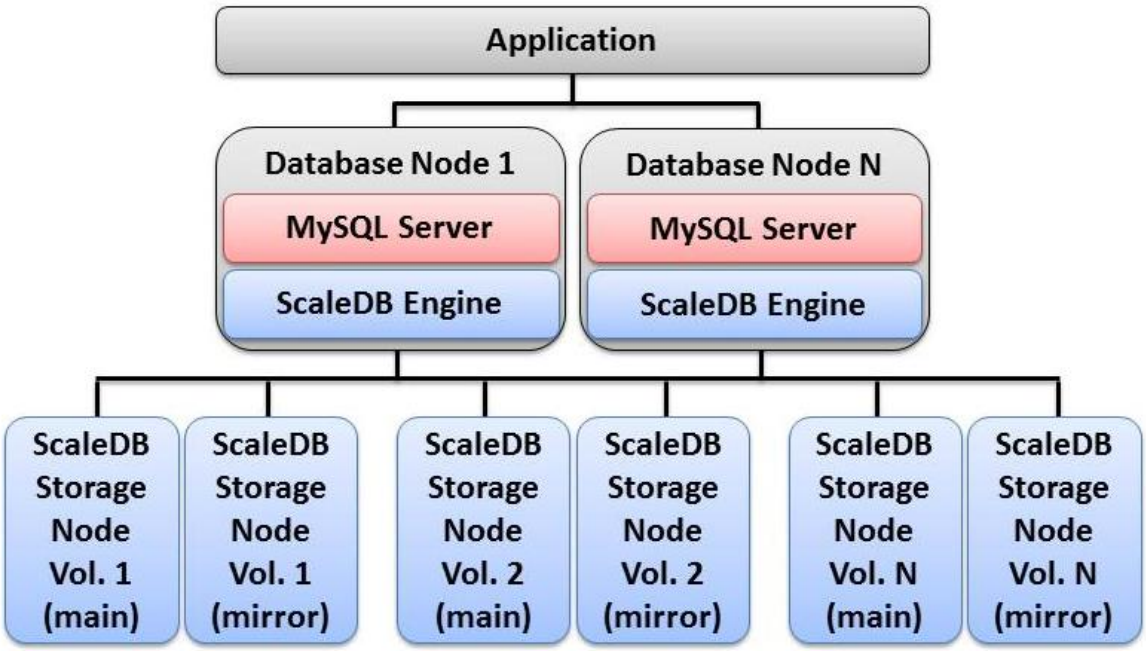
Storage scalability is achieved by adding volumes whereas each volume manages a portion of the data. To avoid hotspots, data blocks are randomly distributed over the different volumes and therefore IO requests are evenly distributed among the storage nodes of the cluster. To Scale the storage tier, more volumes (and therefore more storage nodes) are added. This setup leverages the available cache, CPU and disks on each added storage node. As storage nodes operate without dependencies, a ScaleDB cluster scales by adding volumes to the cluster.

High-Availability of the Storage Layer

When data is written by one of the database nodes in the cluster, the data is send to one volume and therefore to the storage node that manage the data and its mirror. When a database node needs to be acknowledged for a write, it will wait for an acknowledgement from both – the main and the mirror. Therefore, if either the main or the mirror fails, the system continues to run without downtime. By maintaining two copies of data, and maintaining physical separation between the copies of the data, all types of hardware outages, including machine and disk failure will not cause downtime.

ScaleDB Cluster Deployment

The diagram below shows a typical ScaleDB deployment:



A ScaleDB cluster has 3 components:

1) Multiple data volumes make the storage tier. The data is organized in blocks which are randomly distributed over several volumes. With N volumes, the data is striped to N parts and each part is managed by a different volume. A volume contains one or more nodes whereas all the nodes in a volume contain the same data. Data from the storage tier is retrieved by a message to the storage node (or its mirror) in the volume that manages the data. Updates are done by a message to the storage node and its mirror in the volume that needs to be updated.

2) Multiple instances of database nodes that make the database tier. 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 instance receives requests to define data or manipulate data, it transfers the requests to the ScaleDB engine. The ScaleDB engine processes the requests. If data that is needed to satisfy a particular request is not available at the ScaleDB local cache (in the database node), ScaleDB will send a request to retrieve the data from one storage node of the particular volume that manages the needed data. If an update needs to be persistent, a message with the updated data is sent to the main storage node and its mirror (if available) of the volume that needs to be updated.

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 distributed 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 is built without a single point of failure. If a component breaks, the system continues to operate: If a database node breaks, a different database node will undo the uncommitted data of the failed node and queries are routed to one of the surviving database nodes. 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 system will use the mirrored storage node.

This approach provides improved fault resilience over a single database node. In the event of a failure, clustering ensures high-availability to the users and access to mission critical data is not lost.

Database Operations in the Cloud

The combination of a clustered database and storage provides a powerful database solution allowing dynamic scalability and high-availability 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.

On the cloud, database and storage instances are easily configured to replace a single database instance with a scalable grid of instances that provide database services. The grid is transparent to the applications which continue processing as if they operate with a single database instance.

Leveraging ScaleDB to database deployments in the cloud, companies will accomplish the following:

- Build an elastic and high performance database cluster from cloud resources.
- Handle change and growth in data and users in incremental and dynamic steps.
- Consolidate multiple database deployments to common infrastructure on the cloud.
- Deliver data availability for large mission critical applications at low cost.
- Transform a single database instance to a grid of database and storage instances, on the cloud.

These characteristics provide “out of the box” scalability and high-availability transparently to the applications transforming a database instance to a dynamic grid of database and storage nodes that operates, on the cloud, as a scalable database and storage machine.