



Clustered Database Storage Engine

Scaling MySQL in the Cloud

by

Moshe Shadmon, CTO

ScaleDB Inc.

November 12, 2010

Executive Summary

ScaleDB's clustered storage engine is a database engine that seamlessly plugs into MySQL, enabling all MySQL applications—without modification—to inherit scalability, dynamic elasticity and high-availability.

ScaleDB transparently transforms MySQL into a clustered shared-cache database that is uniquely suited for cloud database deployment. With ScaleDB, you can scale your storage and database compute resources independently and in an elastic manner. This enables you to grow or shrink both the compute power and storage resources allocated to the database as dictated by your traffic. This makes ScaleDB ideal for cloud deployments, where a cluster of low-cost virtual machines replaces a single high-cost dedicated server.

Because there is no single point of failure in the ScaleDB cluster, it delivers high-availability. This high-availability is achieved without changing a single line of your application code. By eliminating the added complexity of slaves, replication and database partitioning, the ScaleDB solution simplifies programming, set-up, tuning and maintenance dramatically. This results in a significantly reduced total cost of ownership (TCO) versus vanilla MySQL. 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 increased user counts.

Main Features

- Multiple database nodes in the cluster share access to the entire pool of data in the database, enabling the following:
 - Add new database nodes to scale the database throughput, without interrupting the application;
 - Add new storage as needed, without interrupting the application;
 - If a database node fails, the system redistributes the load to the other nodes without interrupting the application;
 - Since all nodes access the entire database, there is no need for manual partitioning.
- Full ACID-compliance guarantees that all database transactions are processed reliably.
- Delivers a comprehensive solution that includes database, caching and storage—all running on commodity servers—while delivering high-availability for each element.
- By eliminating time-consuming partitioning, replication, slave promotion and the like, deployment is fast, simple and low-cost.

ScaleDB seamlessly extends MySQL applications, moving them from a single server to a cluster configuration without changing a single line of code. ScaleDB handles all of the coordination between database nodes in the cluster, such as locking, creating new tables and the like. As a result, ScaleDB provides a plug-and-play scale-out solution for any MySQL application to run on public and private clouds.

By moving your MySQL application to ScaleDB, it automatically becomes highly-available. Since all nodes in the cluster have full access to the database, the failure of one of the servers simply results in the other servers picking up the slack, instantly and seamlessly, without interrupting your application.

Your MySQL application also inherits elasticity. As the demand on the cluster increases, you can add servers to share the load. As the demand declines, you can repurpose excess servers elsewhere. ScaleDB is virtualization friendly, enabling it to run on virtual machines as well as physical servers.

ScaleDB also provides a complete end-to-end solution for database scale-out. In addition to delivering database clustering, the shared-cache tier—Cache Accelerator Server (CAS)—doubles as a low-cost highly-available storage tier. CAS runs on commodity servers. For high-availability, the data from the database nodes is automatically mirrored to pairs of CAS servers. These CAS servers can be configured to use local storage, cloud storage, or SAN or NAS storage. Operating CAS on commodity servers, configured for local storage, provides an extremely low-cost solution. By automatically mirroring the data, this configuration delivers high-availability with unrivaled price-performance.

The ScaleDB Architecture for the Cloud

The ScaleDB storage engine is a clustered database engine. It enables a collection of servers to operate as a single system. Because it is comprised of redundant servers, which can be added or removed on the fly, the system provides both elasticity and high-availability to MySQL applications.

The ScaleDB architecture is based on the following components:

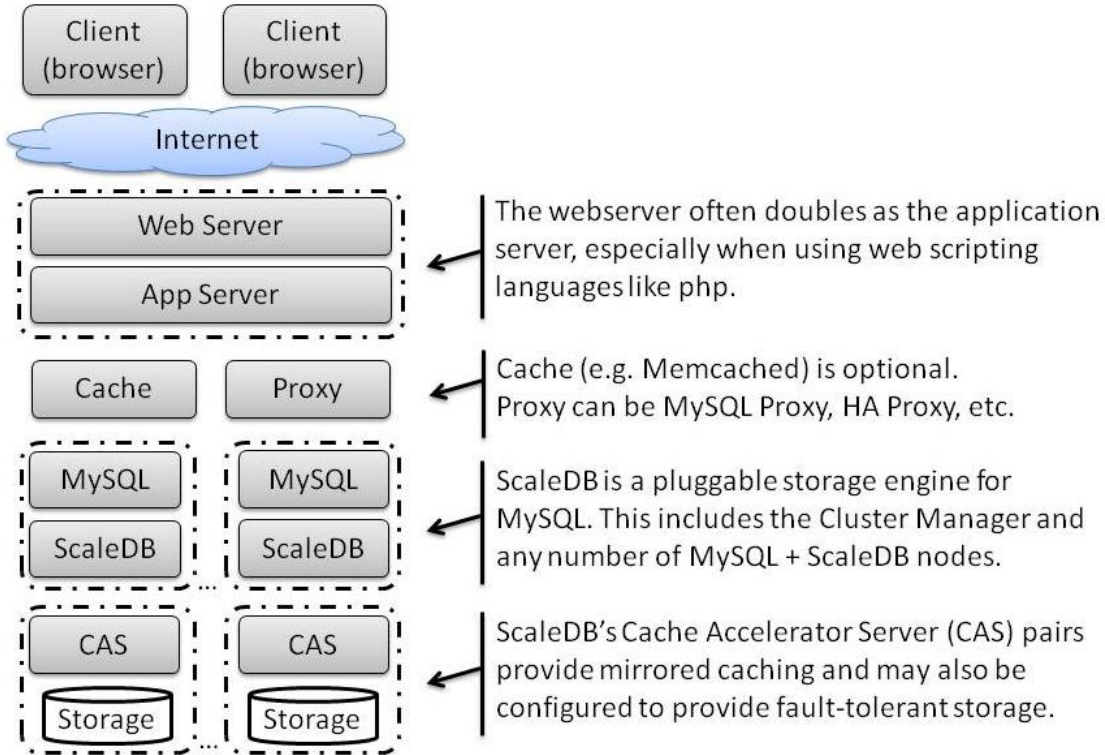
- A cluster of database nodes, all of the nodes share read/write access to the entire database.
- A cluster manager, which, among other things, operates as a distributed lock manager. In this role, it maintains the integrity of the data and the processes for the entire cluster, serving as the traffic cop for the flow of data.
- Pairs of shared cache nodes that provide an efficient and scalable integrated caching layer, and can also serve as low-cost redundant storage for the database cluster.

ScaleDB provides a tightly integrated database and the storage solution. The entire solution is designed to operate on standard or commodity hardware and networks. Instead of requiring you to use costly high-end storage, ScaleDB gives you the freedom to use low-cost PC-based storage. Instead of requiring a complex and expensive general-purpose cluster file system, ScaleDB handles the sharing of the storage via a purpose-built solution that is optimized for performance with database traffic. Instead of requiring high-cost interconnections like Infiniband, ScaleDB runs on any interconnect, from low-cost Ethernet to higher performance interconnect options.

ScaleDB is a general-purpose transactional database. As such it is ideal for online transaction processing (OLTP). It is also an excellent solution for general-purpose online analytical processing (OLAP). However, this is not meant to compete with specialized data warehouses (DW). ScaleDB is ideal for large web applications and consolidation of mixed workloads.

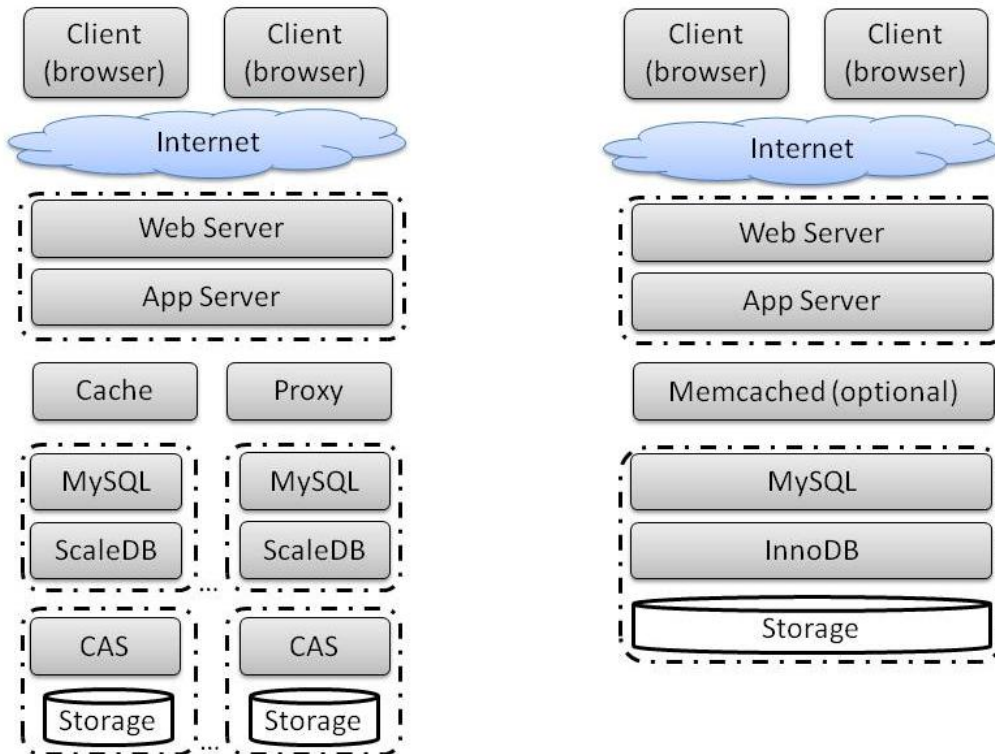
The diagrams below compare a typical single-server MySQL deployment using InnoDB to a typical clustered deployment of shows using ScaleDB.

Typical ScaleDB Configuration



Applications are connected via the internet to a particular database within a cluster of database servers. Each of the database servers is using ScaleDB as the storage engine to share the same physical data. The servers are integrated with storage servers via a network. Each of the storage servers runs the ScaleDB Cache Accelerator. These servers provide the storage for the database nodes and the mechanisms to share the data among the database nodes.

Clustered MySQL + ScaleDB vs. Standalone MySQL + InnoDB



Note: In the clustered diagram with ScaleDB above (1) the cache (e.g. Memcached) is optional; (2) the database tier includes a variable number of database nodes running MySQL + ScaleDB and at least one Cluster Manager; (3) the Cache Accelerator Server (CAS) nodes come in pairs, and in this example they are configured for local mirrored storage of the database.

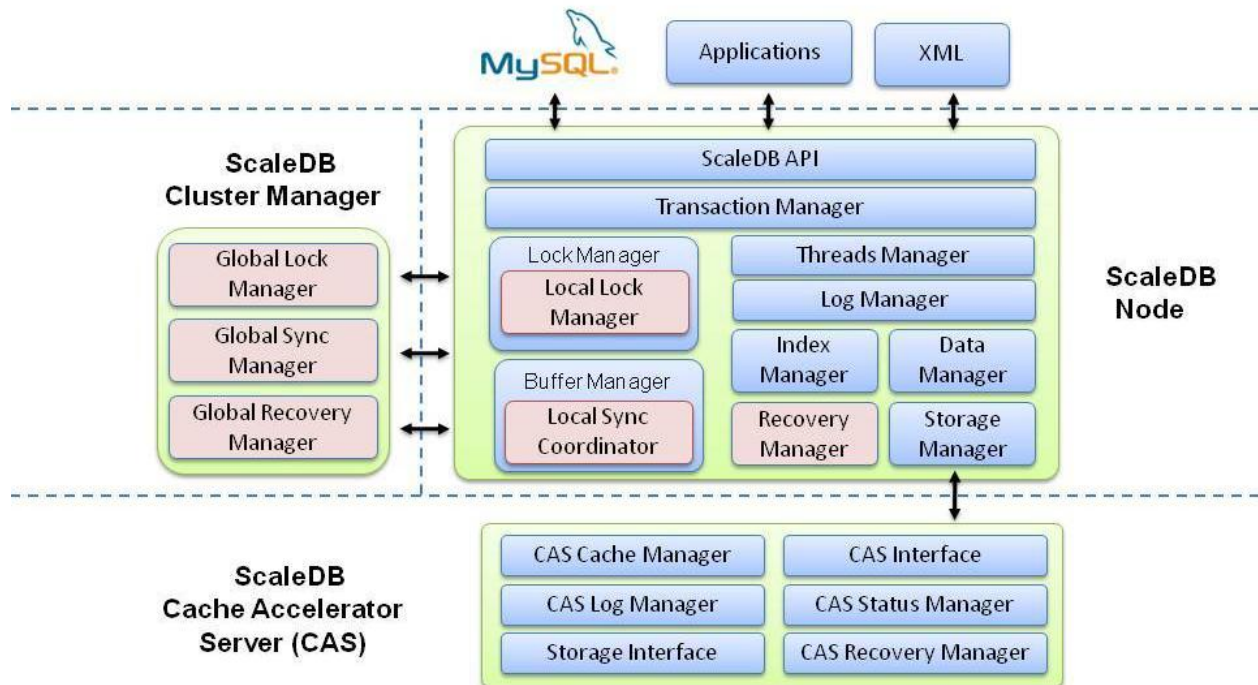
The API and Storage Engine Functionality

MySQL's architecture splits the duties of the database between the MySQL layer and the storage engine layer. MySQL provides the application API, SQL parsing and query optimization, among other things. It then hands off the low-level (read and write) calls, through a standard API, to the storage engine.

InnoDB, which was a separate company from MySQL until 2010 when Oracle officially acquired Sun and MySQL, is the leading single-server storage engine for MySQL. Like InnoDB, ScaleDB interfaces with MySQL via the standard API.

MySQL passes low-level calls via the storage engine API. ScaleDB translates these storage engine calls into transactional ACID-compliant database operations. ScaleDB then updates or retrieves persistent data in a high-performance and highly-available way, while enabling dynamic recovery in case of system failure.

Below is the diagram of the ScaleDB Storage Engine Architecture:



The main components of the ScaleDB cluster are:

1. The database nodes, which combine MySQL and the ScaleDB storage engine
2. The Cluster Manager (and a standby Cluster Manager)
3. The Cache Accelerator Server (CAS) nodes, which can serve as mirrored data storage

The database nodes manage the execution of the SQL statements. They provide the mechanisms for the update and search operations, using advanced and high-performance algorithms. The cluster manager integrates multiple database engine nodes into a cluster, and maintains the integrity of the data across the cluster. Data is updated or retrieved from Cache Accelerator Server nodes. These nodes maintain the data and provide a cache that enables efficient data sharing between the database nodes in the cluster.

The Database Nodes and the Cluster Manager

As a clustered, shared-cache storage engine, the cluster needs something to coordinate the activities of the database nodes, to ensure that they don't conflict with one another; this is the job of the Cluster Manager. The Cluster Manager acts like a symphony conductor, orchestrating the actions of the various musicians.

The Cluster Manager is responsible for coordinating locking. Write locking ensures that when one node is writing data to the database, other nodes aren't attempting to read the data or write conflicting data.

Read locking, ensures that the data, once read, will remain the same until the lock is released. Handling locking, while maintaining superior performance, is at the core of any relational database.

The Cluster Manager also ensures that the various local caches of the database nodes don't get out of sync. For example, the various nodes may read Bob Smith's credit limit at \$25,000. If node #1 changes that credit limit to \$30,000, then the other nodes need to know that the \$25,000 number in their cache is invalid, so that the next time that data is requested they provide the current \$30,000 number.

The Cluster Manager also monitors the status of the various elements of the cluster. If one of these elements (e.g. a database node or a CAS) fails, then the Cluster Manager orchestrates the failover and recovery processes. A standby Cluster Manager automatically takes over in case the Cluster Manager fails.

Shared-Nothing, Shared-Disk and Shared-Cache

The primary database architectures are shared-nothing and shared-disk. Shared-nothing is the most common database architecture. If your database resides in a single database, and high-availability is not a concern, then the shared-nothing architecture is ideal. Shared-nothing can scale-up—moving to a more powerful server—but scaling-out is a much more significant challenge.

The shared-disk architecture also scales-up to larger machines, but the primary advantage of shared-disk is the ability to scale-out. Scale-out refers to running your database on multiple servers or virtual machines. IBM's mainframe databases—IMS and DB2—use a shared-disk architecture. The Oracle Parallel Server (OPS) started as shared-disk, but they later added a shared-cache called Cache Fusion, renaming the product Oracle Real Application Clusters (RAC). By using a shared cache, the data sharing between nodes is much more efficient than sharing data via the disk.

ScaleDB also implemented a shared-cache. Unlike Cache Fusion, which is a peer-based cache sharing system, ScaleDB utilizes a shared-cache tier. Just as Memcached can be used to provide tier-based caching in front of the database to improve performance, ScaleDB's Cache Accelerator Server (CAS) is a tier-based solution that sits in front of the storage (which may be the local disk drive in the CAS). CAS handles the communication between the many database nodes and the storage, eliminating the need for a complex and expensive cluster file system. CAS also greatly accelerates the performance of sharing data between the database nodes. By enabling high-performance, ScaleDB's CAS supports scaling out to a large number of database nodes.

Scalability

Shared-nothing databases scale-out by partitioning data. This is a process that involves splitting the data and running pieces of that data on separate database servers. In MySQL, these separate databases have no knowledge of each other. As such, the developer must write the routing code in the application, so it knows which server to interface with for each piece of data. Furthermore, if you want data that spans

multiple partitions—such as a join, an aggregate, or a 2-phase commit—you must build all of the cross database logic into the application itself. Finally, if you want to change the way in which the data is partitioned, you must shut down the application, repartition the data, update the routing code in the application and then restart the application and database again.

Partitioning results in a static configuration, meaning it cannot change on the fly. As a result, when data volume or the number of users increases, exceeding the current partitioning scheme, the data must be manually repartitioned.

Shared-disk and shared-cache databases present the entire database to all nodes on the cluster. These shared database architectures enable any node to process complex joins, aggregates and 2-phase commits, using the efficiency of the database, without having to code that in the application. The developer also doesn't worry about partitioning, routing code, replication and much more. As a result, shared database architectures reduce the application development effort considerably.

Shared databases—both shared-disk and shared-cache—enable scalability by simply adding more database servers to the cluster. Each of the nodes adds additional processing power to provide the needed scalability. This is done on-the-fly, meaning that the application is not interrupted during this process.

The advantages of the database architectures are as follows:

Shared-Nothing: If the database can be efficiently partitioned, shared-nothing can achieve excellent performance. Shared-nothing can also scale to address very large numbers of users, since the processing is effectively split into small independent pieces. The challenges arise when you need to process operations that span these independent pieces such as joins, aggregates and 2-phase commits. Another challenge presented by shared-nothing databases is the data partitioning scheme; sometimes there is no ideal partitioning scheme.

Ideally, partitioning should split the data such that you get an even distribution of the data and the database calls among the database nodes. This can be a complex task, to determine the distribution of database calls today, how they will change at different times of the day, month and year, and how they will evolve in the future. In addition, the partitioning scheme must accommodate growth of the data over time.

When partitioning, usage patterns must also be considered. Will any database calls require interaction with more than a single server? If so, this introduces inefficiencies. For example, if the Customers Table is partitioned across multiple servers, as is typical in sharded environments, you will run into problems if you needed to join the customer data with the catalog data. Where would the catalog data reside? Do you have a single catalog table on one of the servers or do you duplicate it to all of the customer servers. If you duplicate it, then you get better performance, but you have the headache of synchronizing the various copies, which increases the management overhead. If the Catalog is stored once on a separate server, then queries that are satisfied by Customers and Catalog data must be executed across two or more servers, adding to the overhead and complexity of the system. This challenge is compounded

when there are hundreds of thousands of tables that are all interrelated. The richer the interrelation between tables, the harder it is to partition the data. In many cases, there simply is no good way to partition your data. Furthermore, the rich interlinking between tables can increase as the application matures over time. This means that a database that could be partitioned at one point in its evolution, may not be able to be partitioned in the future.

Shared-Disk: The problems described above are eliminated with a shared-disk database, since each node in the cluster has access to the entire database. A shared-disk solution must 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 that row, the cluster needs to synchronize between the requests. In a single server database, a local lock manager manages conflicting requests from different threads, such that the integrity is maintained (between conflicting requests within a single server). ScaleDB manages conflicts at 2 levels – at the node level, using a local lock manager, and at the cluster level, by a global lock manager (sometimes referred to as distributed lock manager). 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-disk approach assumes that, although conflicting requests between nodes may trigger contention, most of the processes do not initiate contention between nodes and the overall computing power of the cluster yields better throughput than a single database.

High-Availability of a Clustered Database:

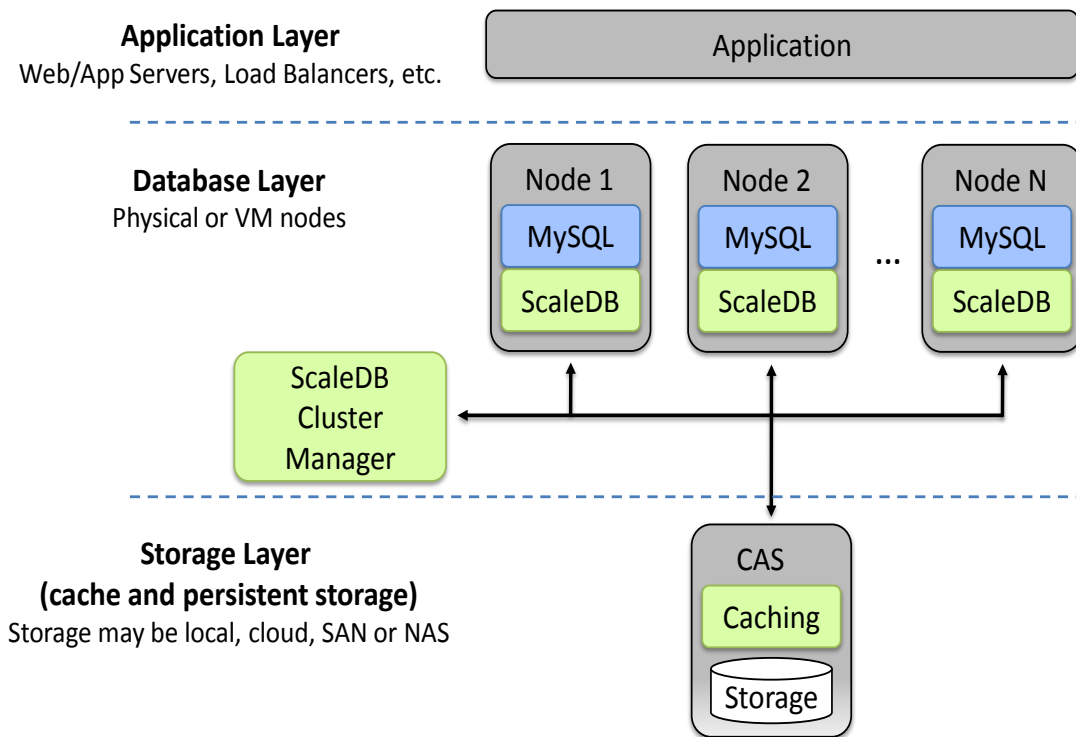
With shared-nothing, data is assigned to a particular database machine. If a server fails, the system is not available, the application stops and the data may be lost. Therefore, to achieve high-availability (HA) in a shared-nothing database, the database is duplicated so that when a server fails, the duplicated database is available. This approach is referred to as a Master and Slave setup, where each partitioned database has a slave database that maintains the same data as the master. However, Master and Slave setup is complicated to maintain. When data is updated, both the Master and the Slave must be updated. The need to update two databases may create complexities. For example, one database may accept an update while the second database rejects that same update. In addition, it creates performance issues. For example, updates on the master may be done concurrently, yet they may need to be serialized on the slave because of different race conditions on the slave.

Shared-disk databases eliminate the need for slaves. Because all shared-disk nodes have access to the entire database, the shared-disk database is impervious to failure of individual nodes. If a node fails, a surviving node will undo any uncommitted transactions from that failed node, and then those transactions, and all future transactions, are processed by the remaining nodes. This process does not require human intervention, nor does it result in an interruption of the application.

ScaleDB has no single point of failure. If a node fails, the other nodes pick-up the load as described above. If the Cluster Manager fails, a standby Cluster Manager identifies the failure and assumes this function.

The drawing below shows a typical ScaleDB deployment of database nodes in a cluster and a Cluster Manager.

ScaleDB: Scalable cluster of database nodes



Using CAS to Provide Low-Cost Mirrored Storage

ScaleDB leverages the Cache Accelerator Server (CAS) to provide high-performance caching for sharing data between the nodes. All of the database nodes have access to the CAS, and the CAS acts as a shared-cache between the nodes. By leveraging a shared cache, instead of sharing via the disk, the performance of the entire cluster is improved considerably. The data that is sent to the CAS is mirrored to a pair of CAS, thereby providing redundancy one would only expect from high-end storage.

CAS runs on commodity servers. A popular configuration is to use the local storage—either the hard-disk or flash memory—to store the entire database. This eliminates the need for a high-end and high-cost shared storage device such as a SAN or NAS. And, since that data is mirrored, the storage is redundant and highly-available.

CAS can be configured to maximize data persistence or to maximize performance. As such, depending on the state of the transaction and the type of data written, a disk flush may complete when the CAS acknowledges that the data has been stored in the caches of two unique CAS; or it may only commit once the data has actually been flushed to the disk on two CAS. This is determined by the user configuration.

Storage Scalability

You have the option to use a single CAS, a mirrored pair of CAS for redundancy, or several mirrored pairs of CAS to provide both redundancy and throughput. ScaledB automatically leverages additional CAS pairs, utilizing their cache, CPU and disks.

ScaledB provides multiple configurable mechanisms to operate the CAS nodes and address potential CAS server failure:

- a) **Cached Mode** – In Cached Mode, user updates are written in the cache layer of a CAS and the CAS flushes that data to disk outside of the transaction. For HA, updates are sent to a mirrored pair of CAS. The database can continue running on a single CAS, without interruption, in case the other one fails.
- b) **Logged Mode** – In this mode, when a transaction is persisted, the data is written to a log file that allows recovery in case of a failure. Writing log files is a sequential process, and therefore much faster than writing the individual data elements. Writing the data would then be done outside of the transaction.
- c) **Hot Backup Mode** – In this mode, CAS dynamically builds a replica of a storage node while the database system is running. Hot Backup Mode can be activated with either Cached Mode or Logged Mode.

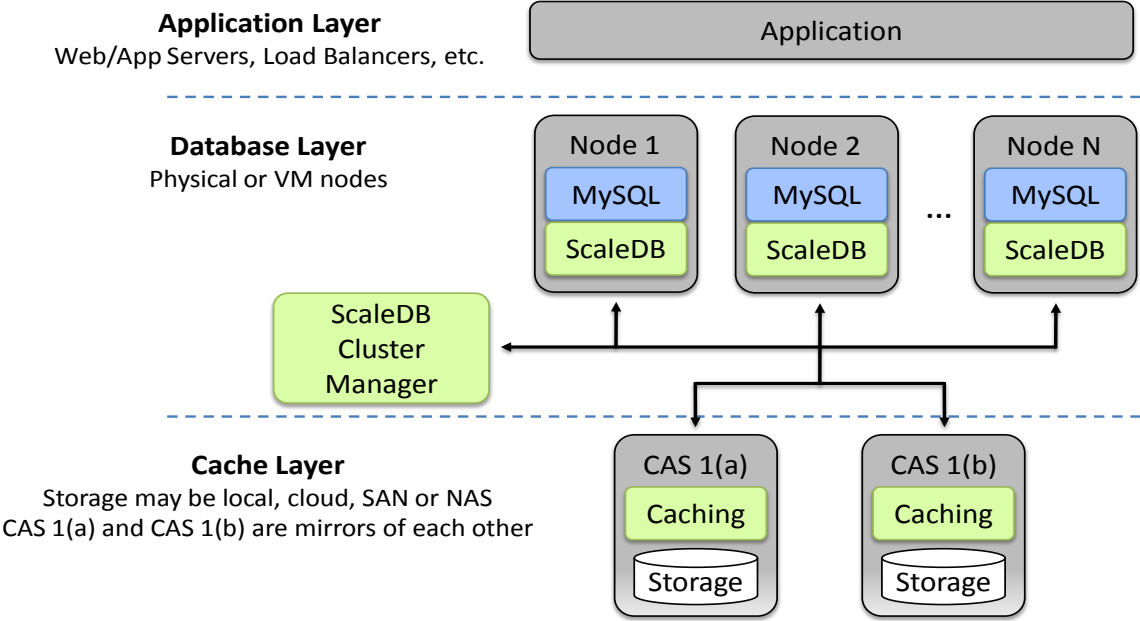
The CAS nodes can run any type of hardware. However, you might choose to use large amounts of RAM or solid state disks (SSD or Flash) to improve performance. Additional RAM increases the amount of data that can be cached, increasing the performance in all modes. Faster storage, like SSD, will improve the performance of actions outside of the cache, especially when running in Logged Mode.

CAS eliminates the need for a Clustered File System. CAS handles the multiplexing of the data to the database nodes itself, in a manner that is optimized for a clustered database. Eliminating the need for a clustered file system, which is used by other shared-disk databases, reduces your overhead, cost and maintenance. If you are using more than one pair of CAS nodes, the database nodes will automatically direct communication to the appropriate CAS pair.

CAS automatically tracks access patterns to determine when and how to optimize cache performance. Because this optimization is automated, it does not require manual tuning.

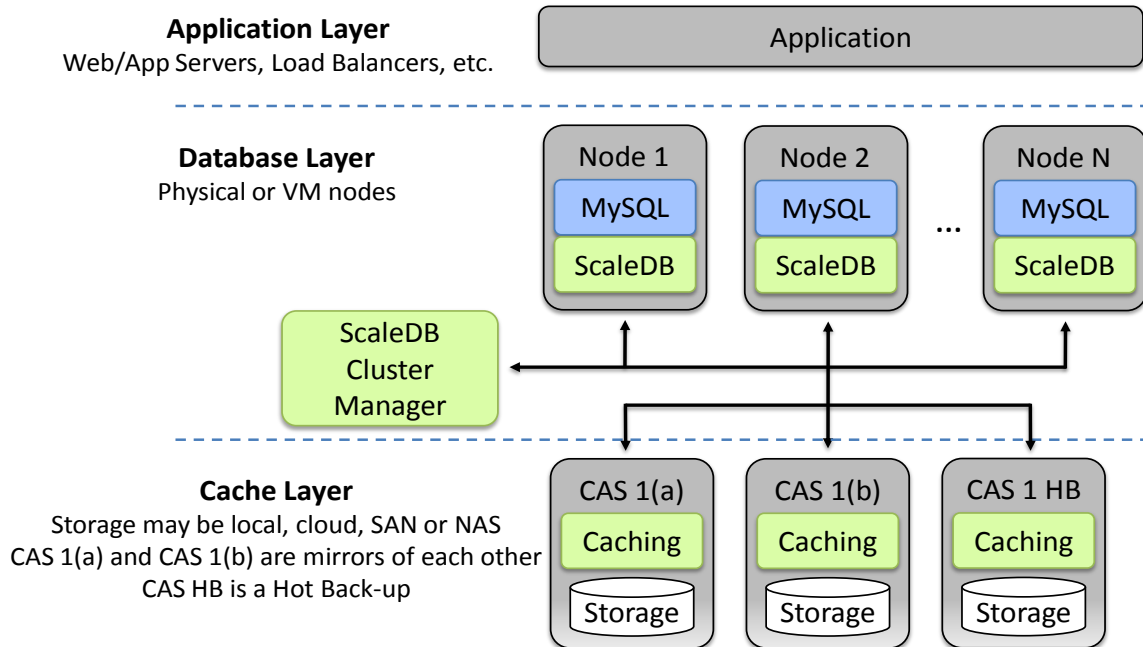
The diagram below shows a ScaleDB DBMS cluster with a mirrored pair of CAS servers. Every time a database node writes data to CAS, that data is mirrored to both CAS; both (a) and (b). This provides redundancy.

ScaleDB: Mirrored Storage



The diagram below shows a ScaleDB DBMS cluster with CAS configured with a hot backup node. This requires that CAS servers be in triples instead of pairs. The third CAS is the hot backup (HB) node. When using the hot backup node, this node is unavailable to satisfy queries by the database nodes, it is solely dedicated to the hot backup function. The hot backup CAS does, however, write the same information as the CAS nodes.

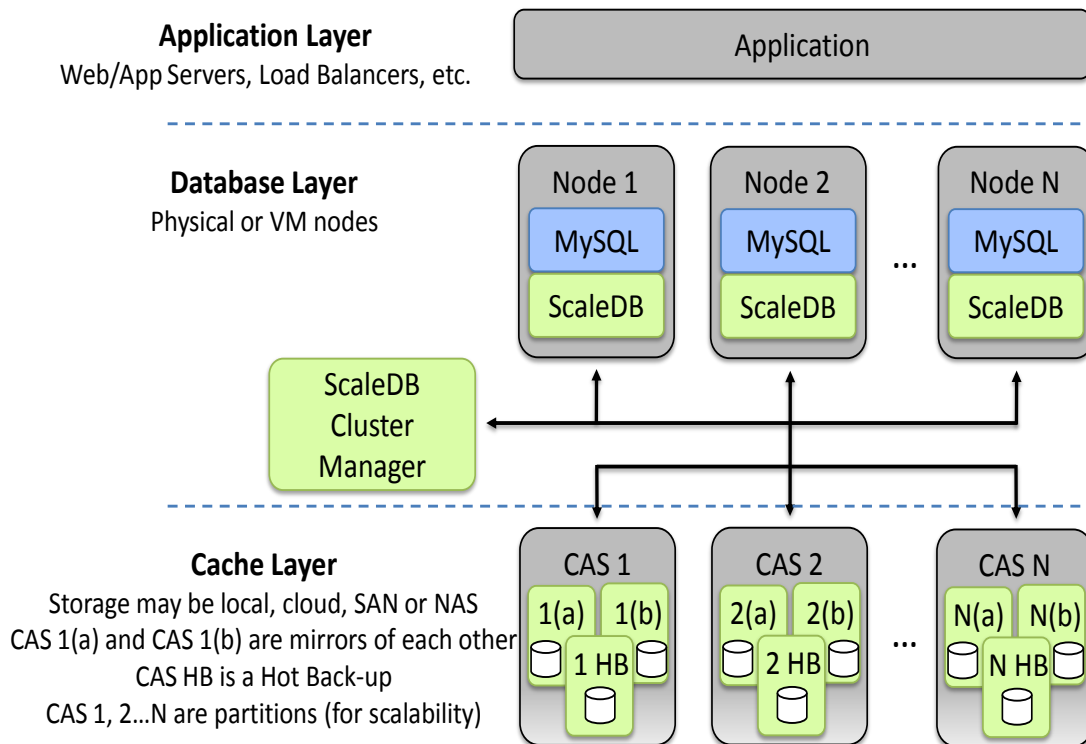
ScaleDB: Mirrored Storage & Hot Back-up



Hot backup provides a continuously updated or current copy of the paired CAS. If you are using more than one pair of CAS, each pair would require a hot backup CAS as well, so that you have a hot backup of the entire database. The hot backup process is handled at the storage interface level, without any downtime or database locking, as is required by other databases.

The diagram below shows a ScaleDB cluster using N number of CAS triples. Each triple includes a pair of active CAS and one hot backup CAS. The database is dynamically split across these CAS triples, but unlike a shared-nothing partition, each database node in the cluster has full access to the entire database. The CAS triples are not associated with selected servers, as with a shared-nothing database.

ScaleDB: Partitioned Storage with Mirrors & Hot Back-up



Businesses today increasingly need to leverage a unified database platform to enable the deployment and consolidation of all applications onto one common infrastructure. Whether the application workload is OLTP, OLAP, or a mixed workload, a common infrastructure delivers the efficiencies and reusability the datacenter needs. ScaleDB provides an integrated database, caching and storage solution with a common high-performance infrastructure, using low-cost commodity servers.

Database Operations in the Cloud

ScaleDB's complete database cluster solution—encompassing database computation, caching and storage—delivers dynamic elasticity, which is the core of cloud computing. Many cloud vendors have gone down the path of providing NoSQL solutions in an effort to deliver a dynamically elastic data store.

These NoSQL solutions are more appropriately named NoACID solutions since they abandon the ACID properties of relational databases. ScaleDB delivers a fully ACID database cluster that plugs into MySQL and *also* delivers dynamic elasticity and high-availability. The combination of a clustered database and storage provides a powerful database solution, allowing dynamic scalability and HA of the database tier in the cloud:

Capabilities:

- a) Database nodes can be added and removed dynamically providing scalability without the need to partition the data.
- b) The storage can scale by dynamically splitting the data at the CAS tier, while each database node still retains access to the entirety of the data. This is the polar opposite of shared-nothing databases, which partition at the database layer, so that each database node only accesses the portion of the data it is associated with; a process that is not dynamic.
- c) The system is built such that there is no single point of failure:
 - 1. If a database node fails – a different node will undo uncommitted data and the application continues running on the surviving nodes.
 - 2. If the cluster manager fails – a standby cluster manager takes over to manage the cluster.
 - 3. If a storage node fails – a mirrored storage node is available.
- d) Both the database nodes and the storage nodes can run on commodity hardware.
- e) Scaling throughput is as simple as dynamically adding/removing database nodes.
- f) Scaling data size is as simple as adding/removing CAS nodes.

When a cloud application running on ScaleDB requires more computing power, more database nodes are dynamically initiated, and they are then removed when workload slacks off. This process is done automatically by monitoring performance and loads.

These characteristics provide “out of the box” cloud-friendly elastic scalability and high-availability. All of this is completely transparent to the applications. ScaleDB has transformed the database tier in the cloud into an automated, elastic and scalable tier that simply runs MySQL applications, while marshalling the compute power it needs on the fly.