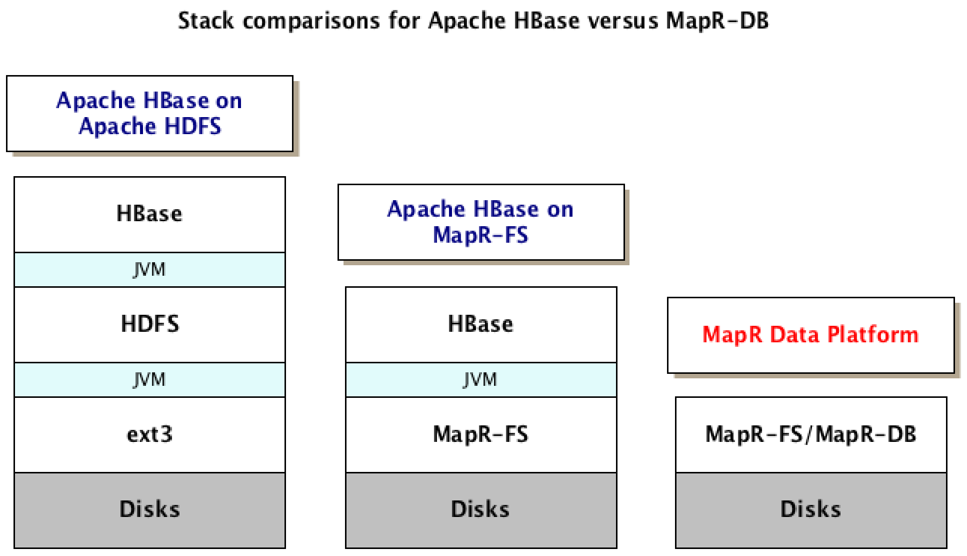
Architecture of MapR-DB and the Resulting Advantages

MapR-DB's architecture gives it a large number of advantages over other NoSQL databases.

MapR-DB operates directly on the file system

MapR-DB tables are implemented directly in the MapR file system (MapR-FS). One of the resulting advantages is that MapR-DB has no layers to pass through when performing operations on data. MapR-DB runs inside of the MFS process, which reads from and writes to disks directly. In contrast, Apache HBase running on the Hadoop file system (HDFS) must communicate with the HDFS process, which in turn must communicate with the ext3 file system, which itself ultimately writes data to disks. The approach taken by MapR-DB eliminates such process hops, duplicate caching, and needless abstractions, with the consequence of optimizing I/O operations on your data.

This advantage in MapR-DB is summarized in the following diagram:



Zero compaction delays

Another advantage is the absence of compaction delays that arise due to I/O storms as logged operations are merged with structures on disk. MapR-DB, like several other NoSQL databases, is a log-based database. Periodically, logged operations must be written to disk. In MapR-DB, tablets (called *regions* in Apache HBase) and smaller structures within them are stored partially as b-trees which together with write-ahead log (WAL) files comprise log-structured-merge trees. Write-ahead logs for the smaller structures within tablets are periodically restructured by rolling merge operations on the b-trees. Because MapR-DB performs these merges at small scales, applications running against MapR-DB see no significant effects on latency while the merges are taking place.

Tablets are stored in containers in MapR-FS

MapR-FS stores data in abstract entities called containers that allow for random write access. Containers reside in storage pools, and each storage pool can store many containers. The default container size is 32GB. For more information about containers, see the section “Containers and the CLDB” in the topic “MapR Data Platform”.

Each tablet of a table, along with its corresponding write-ahead log (WAL) files, b-trees, and other associated structures, is stored in one container. Each container (which can be from 16 to 32 GB in size) can store more than one tablet (which default in size to 4096 MB). The recommended practice is to use the default size for tablets and allow them to be split automatically. Massive tablets can affect synchronization of containers and load balancing across a cluster. Smaller tablets spread data better across more nodes.

There are two important advantages to storing tablets in containers: MapR clusters are extremely scalable and provide exceptional high availability for your data.

Scalability of MapR clusters

The location of containers in a cluster is tracked by that cluster's container location database (CLDB). CLDBs are updated only when a container is moved, a node fails, or as a result of periodic block change reports. The update rate, even for very large clusters, is therefore relatively low. MapR-FS does not have to query the CLDB often, so it can cache container locations for very long times.

Moreover, CLDBs are very small in comparison to Apache Hadoop namenodes. Namenodes track metadata and block information for all files, and they track locations for all blocks in every file. Because blocks are typically 200 MB or less in size, the total number of objects that a namenode tracks is very large. CLDBs, however, track containers, which are much larger objects, so the size of the location information can be 100 to 1000 times smaller than the location information in a namenode. CLDBs also do not track information about tables and files. Therefore, it is practical to store 10s of exabytes in a MapR cluster, regardless of the number of tables and files.

High availability

Containers are replicated to a configurable number of copies, which are distributed to different nodes in the same cluster as the original or master container. A cluster's CLDB determines the order in which the replicas are updated. Together, the replicas form a replication chain that is updated transactionally. When an update is applied to a tablet in the master container (which is at the head of a replication chain), the update is applied serially to the replicas of that container in the chain. The update is complete only when all replicas in the chain are updated.

The result of this architecture is that when a node goes down due to hardware failure, the tablets served by that node are available instantly from one of the other nodes that have the replicated data. In comparison, when a node fails in a busy HBase cluster, it can easily take thirty minutes, if not more, to recover the regions, as the per-RegionServer write-ahead log needs to be replayed in its entirety before other nodes can start serving any of the regions that were being served by the failed RegionServer.

MapR can detect the exact point at which replicas diverge, even at a 2 GB per second update rate. MapR randomly picks any one of the three copies as the new master, rolls back the other surviving replicas to the divergence point, and then rolls forward to converge with the chosen master. MapR can do this on the fly with very little impact on normal operations.

Containers are stored in volumes in MapR-FS

MapR provides volumes as a way to organize data and manage cluster performance. A volume is a logical unit that allows you to apply policies to a set of files, directories, and tables. Volumes are used to enforce disk usage limits, set replication levels, define snapshots and mirrors, and establish ownership and accountability.

There are several advantages to storing table containers in volumes:

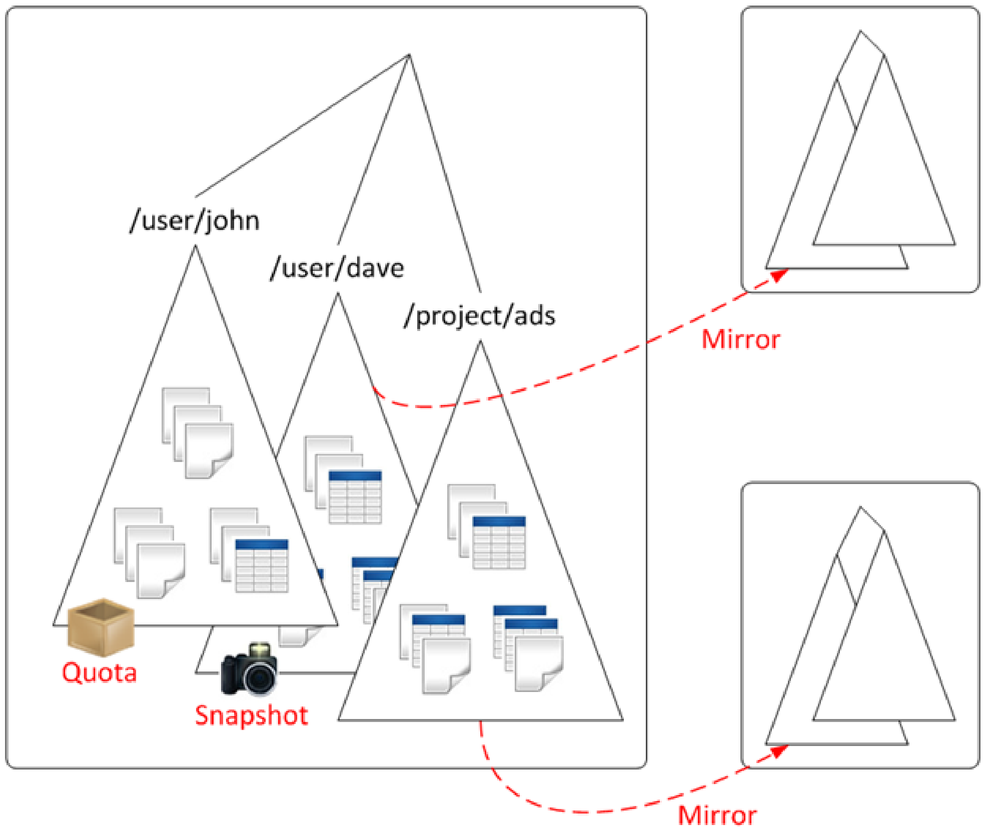
Multi-tenancy

You can restrict a volume to a subset of a cluster's nodes. By doing this, you can isolate sensitive data or applications, and even use heterogeneous hardware in the cluster for specific workloads.

For example, you can use data placement to keep personally identifiable information on nodes that have encrypted drives, or to keep MapR-DB tables on nodes that have SSDs. You can also isolate work environments for different database users or applications and place MapR-DB tables on specific hardware for better performance or load isolation

Isolation of work environments for different database users or applications lets you set policies, quotas, and access privileges at for specific users and volumes. You can run multiple jobs with different requirements without conflict.

As an example, the diagram below depicts a MapR cluster storing table and file data. The cluster has three separate volumes mounted at directories `/user/john`, `/user/dave`, and `/project/ads`. As shown, each directory contains both file data and table data, grouped together logically. Because each of these directories maps to a different volume, data in each directory can have different policy. For example, `/user/john` has a disk-usage quota, while `/user/dave` is on a snapshot schedule. Furthermore, two directories, `/user/john` and `/project/ads` are mirrored to locations outside the cluster, providing read-only access to high-traffic data, including the tables in those volumes.



Example: Restricting table storage with quotas and physical topology

This example creates a table with disk usage quota of 100GB restricted to certain data nodes in the cluster. First, we create a volume named `project-tables-vol`, specifying the quota and restricting storage to nodes in the `/data/rack1` topology, and mounting it in the local namespace. Next, we use the HBase shell to create a new table named `datastore`, specifying a path inside the `project-tables-vol` volume.

```
$ pwd
/mapr/cluster1/user/project

$ls
bin   src

$ maprcli volume create -name project-tables-vol -path /user/project/tables \
    -quota 100G -topology /data/rack1

$ ls
bin   src   tables

$ hbase shell
HBase Shell; enter 'help<RETURN>' for list of supported commands.
Type "exit<RETURN>" to leave the HBase Shell
hbase(main):001:0> create '/user/project/tables/datastore', 'colfamily1'
0 row(s) in 0.5180 seconds
hbase(main):002:0> exit

$ ls -l tables
total 1
lrwxr-xr-x 1 mapr mapr 2 Oct 25 15:20 datastore -> mapr::table::2252.32.16498
```

See Multi-tenancy with MapR for more details.

Capability to create volume snapshots

A volume snapshot captures the state of a volume's directories, MapR-DB tables, and files at an exact point in time. You can use them for:

Rollback from errors

Application errors or inadvertent user errors can mistakenly delete data or modify data in an unexpected way. With volume snapshots, you can rollback your MapR-DB tables to a known, well-defined state.

Hot backups

You can create backups of table data on the fly for auditing or governance compliance.

Model training

Machine-learning frameworks, such as Apache Mahout, can use snapshots to enable a reproducible and auditable model training process. Snapshots allow the training process to work against a preserved image of the training data from a precise moment in time. In most cases, the use of snapshots requires no additional storage and snapshots are taken in less than one second.

Managing real-time data analysis

By using snapshots, query engines such as Apache Drill can produce precise synchronic summaries of data sources subject to constant updates, such as sensor data or social media streams. Using a snapshot of your MapR-DB data for such analyses allows very precise comparisons to be done across multiple ever-changing data sources without the need to stop real-time data ingestion.

See MapR Snapshots for more details.

Replication of volumes with mirroring

Mirroring of volumes lets you automatically replicate differential data in real-time across clusters. You might want to do this to create disaster recovery solutions for databases or to provide read-only access to data from multiple locations. Because MapR-DB does not require RegionServers to be reconstructed, databases can be brought up instantly on the mirrored site if the active site goes down.

Mirroring is a parallel operation, copying data directly from the nodes of one MapR cluster to the nodes in a remote MapR cluster. The contents of the volume are mirrored consistently, even if the files in the volume are being written to or deleted.

MapR captures only data that has changed at the file-block level since the last data transfer. After the data differential is identified, it is then compressed and transferred over the WAN to the recovery site, using very low network bandwidth. Finally, checksums are used to ensure data integrity across the two clusters. There is no performance penalty on the cluster because of mirroring.

See Working with Mirror Volumes for more details.