TOSHIBA
Leading Innovation

NoSQL Database Architectural Comparison

June 20, 2017
Revision 0.07

GridDB™

Table of Contents

Executive Summary

This white paper compares and contrasts Toshiba's GridDB database to Cassandra, MongoDB, Riak, and Couchbase. Topics covered include the logical and physical cluster topology, how each database handles consistency, replication, and failover as well as the individual storage engine and caching mechanisms that are used. Finally, the Client APIs of the reviewed databases are showcased, demonstrating how developers may build applications.

Introduction

The term NoSQL (or Not Only SQL) became prominent in the late 2000s as the amount of data collected and used by popular web services began to increase exponentially, requiring a solution that could scale better than SQL databases with their tabular storage engines and relational queries.

As a whole, NoSQL databases tend to scale out and favor high reliability, but this is not always the case as databases such as RocksDB (not evaluated here) are meant for use in a single instance.

Cassandra

Cassandra was inspired Amazon's Dynamo paper and was initially developed by Facebook with its first release in 2008. It is written in Java and many companies currently contribute to it as a top-level Apache project with the most notable being Datastax.

MongoDB

10gen began developing MongoDB in 2007 as part of another project before open sourcing it in 2009. 10gen is now known as MongoDB, Inc. and offers commercial support for MongoDB.

Riak

Riak is also based on the principals of Amazon's Dynamo and is written in Erlang and was initially released by Basho Technologies in 2009. Basho offers supported versions of Riak that have additional features.

Couchbase

Couchbase is the merger of Membase (first released in 2010) and CouchDB (first release in 2005) projects and their respective companies in 2011 with the first release of the combined product in 2012. It uses C/C++, Erlang, and Go for different components. CouchDB has continued as a separate project.

GridDB

Toshiba started GridDB development in 2011 with its first commercial release coming in 2013; it was then open-sourced in 2016. It is written in C++ and has language bindings for C/C++, Java, and Python. It has also been integrated with other open source projects such as MapReduce and KairosDB.

Data Type

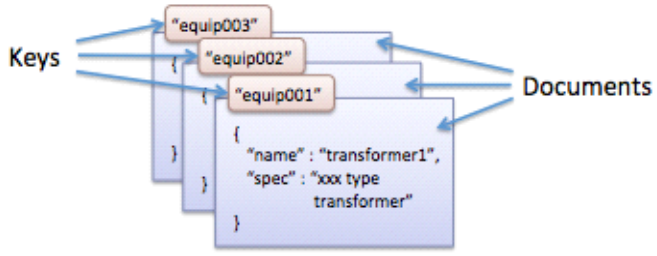
With traditional RDBMS databases, data is stored in a table with a predefined structure which can then be queried using any of the fields. NoSQL databases however do not all share the same structure, different data databases have different data models.

Cassandra

Cassandra uses a key-column data schema that is similar to a RDBMS where one or more columns make up the key. The rows in a Cassandra table can be queried by any value but the keys determine where and how rows are replicated.

MongoDB

MongoDB is a key-document database that stores individual documents in a JSON-like format called BSON. Individual documents can be queried with a key, field values or they can be grouped together in a collection which is analogous to a table in a RDBMS.



Riak

Riak is a key-value database. The value can be a simple literal or it can be a more complex user-defined structure. Riak does not understand any part of the value and thus only the key may be used to query the database. Keys can be separated across different namespaces, these virtual keyspaces are referred to as buckets.

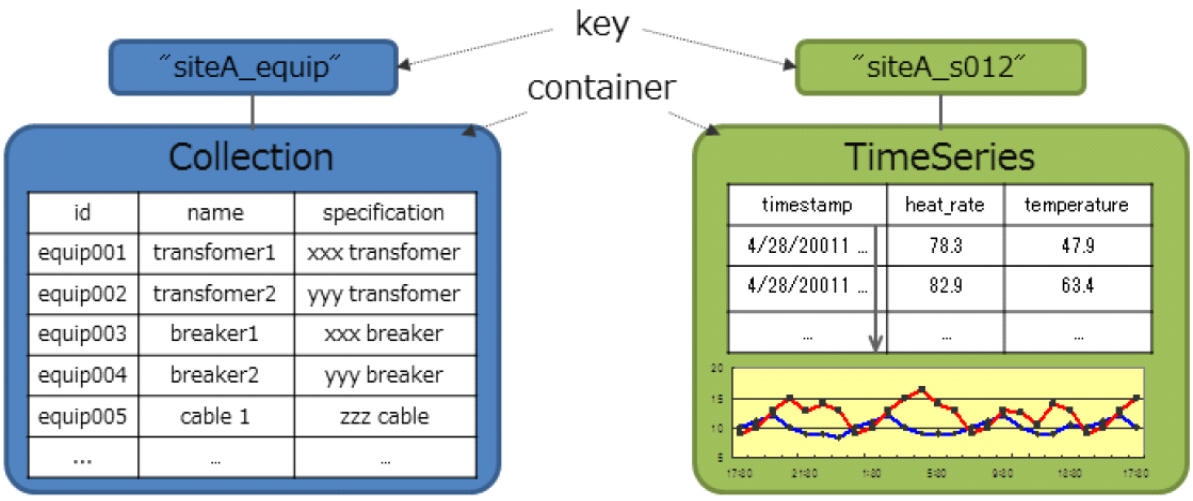
Riak supports using a TimeSeries key type but this requires a different installation and changes the data model to being tabular, where different tables can have the same time key but different values assigned.

Couchbase

Couchbase supports both key-value and key-document databases. The databases key space can be separated by using buckets. By setting a flag, the value can be serialized using UTF characters, raw bytes, Python's native pickle format, or with a user-defined transcoder. Like MongoDB, documents are stored using JSON.

GridDB

GridDB is a key-container database. The key can be either any specified user value or a timestamp. Each container can be specified via a key and then can be further queried like a traditional RDBMS.



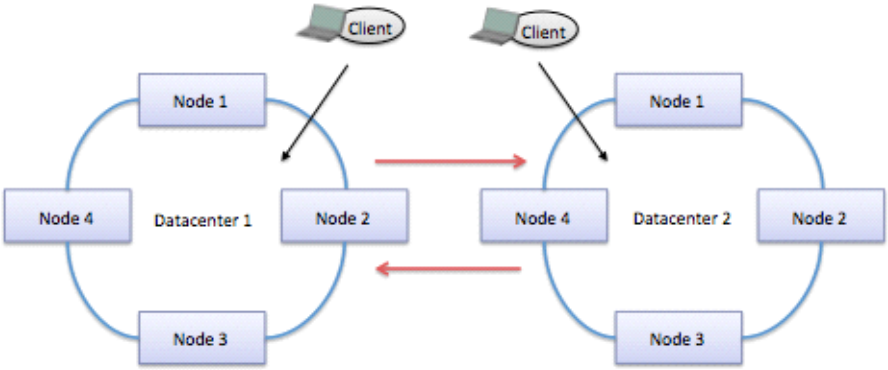
GridDB supports both regular Collections and TimeSeries containers. Collections can use any value as a key while TimeSeries Containers use a time value that allow for specialized handling within the application amongst other features. Unlike Riak, GridDB supports both Collections and TimeSeries Containers in one installation making storing and accessing meta-information about a TimeSeries significantly less onerous than having to switch anf manage between different APIs and connections.

Cluster Topology

Distributed services have two common models, master/slave and peer-to-peer. Typically, master/slave topologies have a single point of failure while with a peer-to-peer cluster, all nodes are equivalent and one or more can fail without affecting the others.

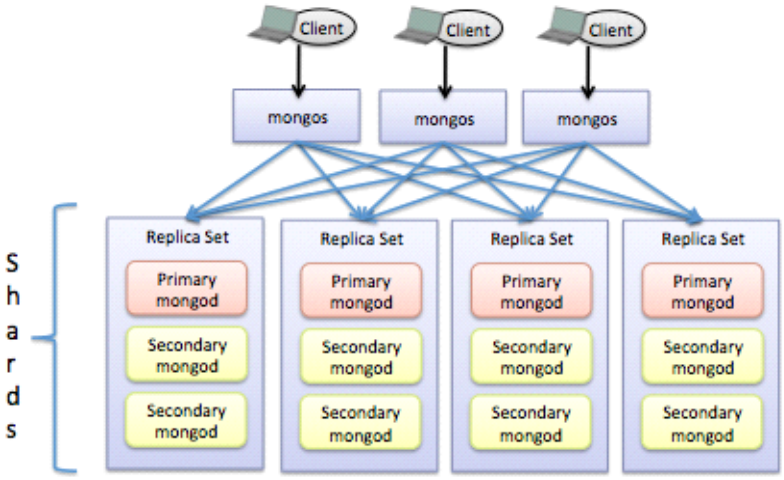
Cassandra

Cassandra has a peer-to-peer ring based architecture that can be deployed across datacenters. A Cassandra installation can be logically divided into racks and the specified snitches within the cluster that determine the best node and rack for replicas to be stored.



MongoDB

MongoDB has a hierarchal architecture built out of one or more query routers named **mongos** and then one or more shards that run **mongod**, which can be built using replica sets. Each replica set has a primary node and then multiple secondary nodes.



Riak

Riak has a peer-to-peer master-less architecture with multiple data center support available in the commercial version. With multiple data center support, one data center will act as the primary and the others will synchronize with it.

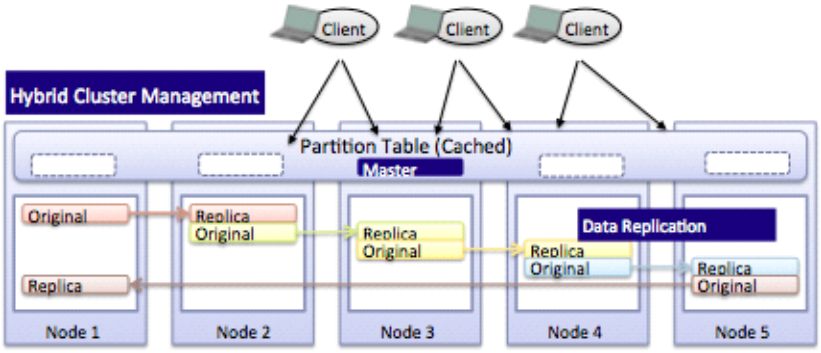
Couchbase

Couchbase has a peer-to-peer architecture where each node contains a data, cluster manager, index and query service. With Couchbase's multiple data center support, updates can flow from one data center to others or bilaterally with conflicts typically being resolved by each cluster being the owner for a certain set of partitions.

GridDB

GridDB has a hybrid master/slave architecture. All nodes in the cluster contain partitioning data required to organize the cluster but only one node acts as the master. In the case of a master failure, a bully election is held and another node is quickly promoted to being the master. This hybrid architecture provides both the high performance of a master/slave architecture and high reliability but without a single point of failure.

Having any node being able to take over for a failed master prevents potential problems, such as a risky single-point of failure, like in MongoDB where there are only a small number of replica mongos instances.



Consistency Model

With distributed databases, individual designs must trade off between consistency, availability, and partition tolerance. This theorem was first described by Eric Brewer in TODO and is known as the CAP theorem.

Consistency (C) is defined by the database always returning the latest update or an error if it can't, with Availability (A) defined as the database always returning a result, even if it's not the latest, and Partition-tolerance (P) means the database will continue to operate despite partial node or network failures between nodes.

As partition tolerance is a must with distributed databases leaving the choice between Consistency and Availability. Consistency can be implemented two ways: eventual or immediate.

Eventual consistency allows different nodes that may have copies of the same data item to update their own copies and make them available for reading without being concerned that if that data is consistent with the copies data items of different nodes. The premise is that eventually all accesses to that data item will return the same, most recently updated version of that data item.

Immediate consistency involves multiple nodes having to agree on a data item after the item is updated before it can be made available to access. This is to ensure that every access for that data item, for example a row, returns a consistent and up-to-date version. This premise can have the cost of having higher overhead and lower availability whenever an item is updated.

Cassandra

Cassandra is typically an available, partition tolerant (AP) database but has tuneable data consistency parameters. A user can change both write and read consistency levels from

having no consistency to requiring all nodes be consistent with options to enforce consistency between data centers or not.

MongoDB

MongoDB is a consistent, partition tolerant (CP) database by default, but it can be configured to read from the secondary nodes within a replica set allowing for the possibility of older data to be served.

Updates to an individual document are atomic meaning a client will always read either the old version or the new version, but never a half-updated document. By default, when multiple documents are updated in a single query they will be updated individually meaning a client could read some updated documents and some old documents.

Riak

Like Cassandra, which is also based on the ideas presented in Amazon Dynamo White Paper, Riak is eventually consistent, meaning that the default configuration is considered to be an AP database. Consistency can be enforced by setting configuration variables that ensure there is a quorum during a read.

Couchbase

A single Couchbase cluster by default is a consistent, partition tolerant cluster but can tuned to favour availability over consistency. A multi-datacenter Couchbase installation is always eventually consistent.

GridDB

GridDB is the only fully ACID (atomicity, consistency, isolation, and durability)compliant database reviewed in this white paper. The ACID compliance guarantees a consistent, partition tolerant (CP) database. GridDB allows the user to configure the database to allow immediately consistency or eventually consistency. In the case of eventual consistency, if the node that is the partition owner fails and the updates have not yet been propagated to any of its backup nodes, an error will occur on access to that partition.

Partition Scheme

Partitioning data is a core requirement for a multi-node database and the five databases in this whitepaper use similar but varying methodologies to spread data across their nodes.

Cassandra

Cassandra uses the first part of a row's primary key to build the partition key. From the partition key, Cassandra distributes rows amongst nodes in the cluster. This means extra care must be taken when choosing a primary key to ensure data is spread evenly across the cluster.

MongoDB

MongoDB uses a specified shared key to partition a collections documents across the nodes in the cluster. The shared key can be either a single field or compound index (an index made

up of multiple fields in each document). The shared keys are immutable, meaning that you can neither switch to new fields nor update the values of the shard key field(s).

Riak

In Riak KV, partitions are automatically spread across virtual nodes (vNodes) by calculating a hash of the bucket and key. A vNode is a logical division of a cluster's ring size across its nodes.

With Riak TS, rows are partitioned based on a specified partition key of one or more columns in the table.

Couchbase

Couchbase calls its internal partitions vBuckets. Documents are mapped to vBuckets based on the hash of their document ID. Document ID's can be automatically generated or specified by the application but are always unique.

GridDB

A GridDB partition is a logical area that stores whole containers and is not directly visible by a user. A hash algorithm that uses the container key as the seed determines which partition a container is assigned to. An allocation table of the node to each partition is called a partition table.

The master node distributes this partition table to the follower nodes or client libraries. By referring to this partition table, the owner node and backup node belonging to a certain container can be made known.

As GridDB stores whole containers in each partition, accesses within a container are always fast as they do not require internode communication or coordination.

Replication Strategy

Data replication is how distributed databases are able to maintain availability after a node fails by placing a configurable number of copies or replicas of every partition on different nodes.

Cassandra

Cassandra has two different replication strategies that can be set on a per table basis along with the number of replicas. With SimpleStrategy, partition scheme sets the first node the replica is stored on and then replicas are stored on the next node in ring. NetworkTopologyStrategy takes into consideration that nodes in the same rack often fail at the same time and replicas are stored on the first node in the ring that is in a different rack.

MongoDB

MongoDB uses replica sets to store multiple copies of its dataset. Each replica set consists of multiple mongod processes, a primary and one or more secondaries. Each replica set is

manually defined in each node's mongod configuration file and the mongo shell. An end user is able to include nodes from different data centers in each replica set. Typically, a client reads from the primary but MongoDB can be configured to read from the secondaries as well. A client will always write to the primary which will then propagate the update to the secondaries.

Riak

Riak is built around replication and the number of replicas can be set on a per bucket basis. Replicas are assigned to the specified number vNodes that may not actually be on different physical machines. A quorum of a configurable number of vNodes responses to a read request is required for the read to be successful.

Couchbase

Couchbase offers a per-bucket configurable level of replication and every node contains both active data and replica data. Reads and writes are always to the node which has the active data. That is until the case of failure in which case the cluster-wide map of where data should be retrieved from is updated, pointing to the replica data.

GridDB

Replication levels can be set for the entire cluster with GridDB. Each partition has a node that is the owner, enough backup nodes to satisfy the configured replication level, and any catch up nodes that are required. If GridDB is configured for immediate consistency, the partition's owner handles to all client read and write requests and propagates updates to the backup nodes while the replicas may respond to read requests if the database is configured to be eventually consistent.

Catch up nodes are used when there are not enough backup nodes to satisfy the replication level or if there are other problems with the backup nodes, the master node first transmits large memory blocks of its partition data to the catch up node and then transfers update logs until the catch up node is in sync with the partition owner and can be promoted to a backup node.

This strategy is similar to Couchbase and MongoDB, reads and writes are direct to the partition owner rather than requiring a quorum of peer-to-peer nodes.

Failover Method

With distributed databases, failure of a node is a given over the lifetime of the product. How the database handles the both failure and recovery of a node is critical to how it will be adapted to various applications.

Cassandra

Within a Cassandra database, pre-defined gossip nodes track the state of all of the nodes. Gossip nodes collect the state of other nodes both directly to or from other gossip nodes. Rather than use a fixed threshold to mark nodes as down, the gossip nodes use a sliding window that takes into account the operating conditions of the cluster as well as

configurable threshold to account for the differences between operating in the public cloud or a high performance LAN or somewhere in between.

After a failed node comes back online, it will try to replay the hints for rows it owns stored by other nodes during the outage. If the node was down for longer than the configurable hint storage window, a manual node repair must be run to re-replicate the row data written during the downtime to ensure consistency.

MongoDB

MongoDB's replica sets use a simple heartbeat between members every two seconds. If a heartbeat has no response within 10 seconds, the node is marked as being down. If the down node is the primary, an election will be held and one of the secondaries will become the primary and begin accepting writes. In the case of a network partition, the primary will demote itself if it cannot reach the majority of it's nodes while the unreachable nodes will carry out an election and one will be promoted to becoming the primary.

Riak

Like Cassandra, Riak uses gossip protocol, quorums, and hinted hand offs to deal with failures within the cluster.

Couchbase

The Couchbase Cluster Manager monitors the status of all nodes in the cluster and if it's configured to do so will trigger auto-failover. Auto-failure will mark the node as offline and transfer ownership of its owned vBuckets to other nodes. If the node is not re-added, a manual rebalance will need to take place that permanently reassigns vBucket ownership and replicas. If the node is added back to the cluster, recovery is performed manually either using the delta or full recovery method. The delta method incrementally catches the node up to the latest state while full recovery removes the stale data and a full rebalance is then performed.

GridDB

The GridDB master monitors all of the nodes in a GridDB cluster. When a node does not respond to the master's heartbeat the master marks that node as down and assigns the partitions it owns to the other replicas. If after failure, the configured replication level is not met, another node will be assigned the role of CATCH UP and that partition will begin transferring large blocks of data. After the initial high speed memory synchronization is complete, it will be transferred

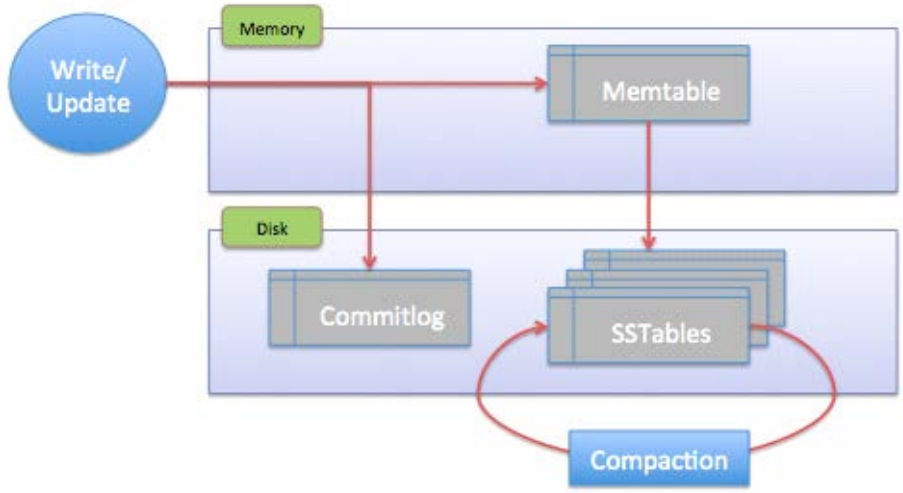
Storage Engine

The details of how each of the different NoSQL databases store their data are different. There are two main themes: append-only transaction log files that require compaction, which involves reducing disk space through old and unused date from the database, used by Cassandra, Riak, and Couchbase and checkpoint writes of data tables that are used by MongoDB and GridDB.

The log based databases present excellent write and update performance while the checkpoint databases can offer consistently better read performance. Note for databases that use compaction, this process causes the system sees a spike in both disk usage and I/O activity which must be planned for as this can affect speed and performance.

Cassandra

Cassandra stores its data in a log merge tree that avoids reads before writes. Updates are initially appended to the commit log on disk and the in-memory memtable. The commit log allows data persistence during sudden failures. When the memtable exceeds the configured size it gets flushed to disk as multiple log structured merge (LSM) tree SSTables (**S**orted **S**tring **T**ables of keys and values sorted by key) per table/partition.



As Cassandra does not update the SSTables in place, they must be compacted. For Cassandra to remove old or deleted, the most recent version of the records will be moved to a new SSTable and then unlinking the old SSTable. During compaction, the system sees a spike in both disk usage and I/O activity which must be planned for as this can affect speed and performance.

MongoDB

MongoDB supports multiple backend storage engines but recommends using the WiredTiger backend for all new installations. WiredTiger can utilize either a B-tree or LSM-tree structure and uses checkpoint commits to flush it's structured memory image to disk. Between checkpoints, WiredTiger writes out a write-ahead log or journal that ensures data persistence between checkpoints.

Riak

Riak has a pluggable storage engine with three default options: Bitcask, LevelDB, and Memory. Bitcask is the default storage engine and is append-only requiring compaction that keeps the entire key space mapping in memory. The LevelDB backend utilizes a fork of Google's LevelDB database using an LSM structure that allows the keyspace to be larger

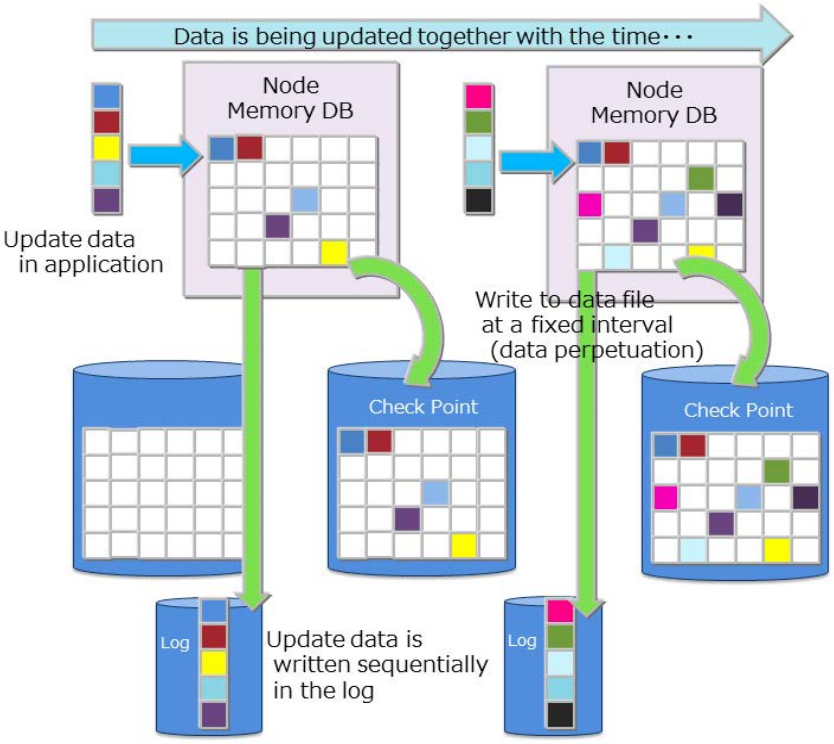
than the memory size. The Memory backend uses in-memory tables and has zero persistence.

Couchbase

Couchbase uses two different storage engines, Couchstore for data and ForrestDB for indices. Both Couchstore and ForrestDB are append-only only storage engines that require compaction although ForrestDB has a circular re-use mode that allows new writes to take space in the log that was freed by old or deleted records.

GridDB

GridDB is intended to keep most of the database in-memory and uses a method similar to MongoDB to persist data. It's internal memory structure is flushed to disk on a checkpoint interval while maintaining a transaction log between checkpoints.



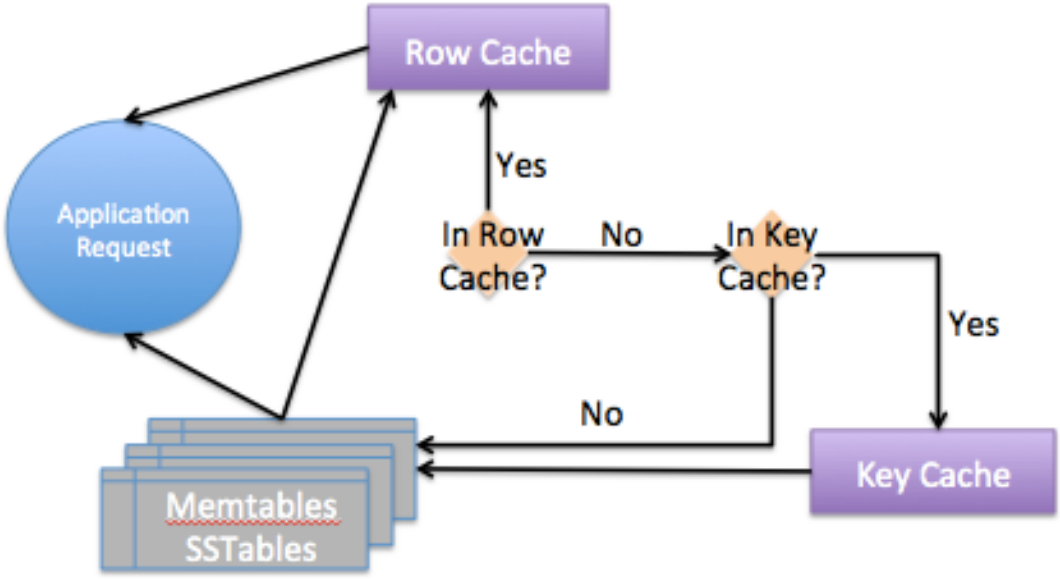
When the database outgrows the configured memory utilization, data not likely to be used soon is freed from memory. This allows more database transactions to occur in-memory, meaning GridDB does not access the disk as frequently thus offering higher performance. Data stored in memory and on disk are both stored using B-trees.

Caching Mechanism

Caching mechanisms are closely related to the databases' storage engines and vary considerably between the discussed databases in this whitepaper with Cassandra, MongoDB, and Riak not using a specified cache system at all by default. GridDB attempts to keep the entire database in memory if capacity allows.

Cassandra

Cassandra has two caches, a row cache and a key cache that are both disabled by default. The row cache is meant to be used by very hot records that are used often. Both caches can be configured for maximum size and record age. The key cache is recommended to be 10% of the heap size or 100MB.



To enable better caching, many Cassandra users add an additional layer and add a tool like *memcached* to their stack to increase performance.

MongoDB

MongoDB will keep working data up to the configured size in memory but does not store the results of queries in memory. By default WiredTiger uses half of the system memory minus 1GB for its cache allowing the filesystem cache for itself and other applications to fill the remainder.

Riak

The caching mechanism used by Riak depends on it's backend storage option. With Bitcask, Riak relies exclusively on the filesystem cache provided by the operating system. The LevelDB backend, on the other hand, has a configurable cache size. As previous Levels are stored in the cache, only one disk read should be required for a query. Of course, the memory backend keeps the entire database in memory making further caching redundant.

With Riak Enterprise, Basho also offers a Cache Proxy service that enables users to integrate Redis into their stack.

Couchbase

Couchbase is built by engineers of *memcached* software and touts a memory-first architecture. Actual row data is kept in a memcached system similar to cache while the Index and Search services store the most popular indices in memory. The Query service doesn't store the actual query results but does cache the processing streams required to calculate the query response.

GridDB

GridDB is an in-memory database with persistence that will store up to the configured amount of data in memory and recommends that most of the system memory is assigned to it. GridDB uses Least Recently Used (LRU) and Data affinity algorithms to determine which records stay in memory and which records remain only in disk. These algorithms help ensure only the most relevant and queried data in the database stay in the cache, which can aid in the overall efficiency of the application that uses GriddB.

The large amount of memory given to the cache along with the optimized cache management functions allow GridDB to offer significantly higher cache hit rates allowing high performance without having to incorporate a second caching system like memcached or Redis.

Client APIs

Cassandra

Applications use Cassandra Query Language (CQL) to interact with a Cassandra database. As Cassandra's data type is the most similar to SQL, it makes sense that CQL is the most similar to SQL having commands such as CREATE, SELECT, INSERT, UPDATE, ALTER, DROP as well as aggregation functions. The biggest difference between SQL and CQL is that CQL does not directly support joins but joins can be completed using Spark.

```
INSERT INTO items (id, number) VALUES ("foo", 25);
INSERT INTO items (id, number) VALUES ("bar", 42);

SELECT * FROM items WHERE number < 30;
```

MongoDB

MongoDB's API differs depending on the programming language used for the application. In most cases either native type dictionaries or parsable JSON strings are provided to insert records. Fetch queries can either be built using a JSON structure or using simple operators such as eq().

Python Insert:

```
db.items.insert_many( [
```

```
        {"id": "foo",
        "number": 25 },
        {“id”: “bar”,
        “number”: 52 }
])
```

Java Query:

```
collection.find(lt("number", 30)));
collection.find(eq("name", "456 Cookies Shop"))
```

Riak

Riak uses a HTTP API at its core allowing users to GET and PUT key-value pairs using curl or slight abstractions via a native language API. Search queries of user-defined indices can be done using Riak’s Solr integration.

```
myBucket.store("foo", 25).execute();
myBucket.store("bar", 52).execute();

Integer foo = myBucket.fetch(“foo”, Integer);
Integer bar = myBucket.fetch(“bar”, Integer);
```

Couchbase

Like Riak, Couchbase uses a HTTP REST API and with simple get/put semantics available to applications.

```
JsonObject foo  = JsonObject.empty().put("number", 25)
JsonObject bar  = JsonObject.empty().put("number", 52)
JsonDocument fooResp = bucket.upsert(JsonDocument.create("foo", foo););
JsonDocument barResp = bucket.upsert(JsonDocument.create("bar", bar));

JsonDocument foo = bucket.get(“foo”);
JsonDocument bar = bucket.get(“foo”);
```

GridDB

GridDB can be queried either through its API, via TQL , or with NewSQL in Advance Edition. TQL is a fairly small subset of SQL that allows developer to perform basic operations including search and aggregation. NewSQL offers a full SQL-92 compliant query language.

```
        Item foo = new Item(“foo”, 25);
        Item bar = new Item(“bar”, 52);
        collection.put(foo);
        collection.put(foo);

        Query<Item> = collection.query(“SELECT * WHERE number < 30”, Item.class);
```

Like Cassandra, GridDB also supports the standard aggregation functions like MIN, MAX, AVERAGE and also includes specialized TimeSeries functions such as TIME_AVG that provides the time-weighted average of a field. TIME_NEXT provides a result that is identical with or just after the specified timestamp or TIME_SAMPLING that returns rows that match the specified interval in the given time frame.

Conclusion

As we've shown, GridDB has many novel features that will make it excel in the applications it was designed for. As no database is perfect for all use cases, it is hoped that the preceding whitepaper was a useful comparison and contrast of the many NoSQL databases. They are not all created equal and have both common and differing features and careful consideration is required before selecting a database for a particular application to ensure that the goals of the