Oracle NoSQL Database

Failover

12c Release 1

(Library Version DRAFT)

Legal Notice

3/27/2014

Table of Contents

Introduction

Oracle NoSQL Database is a data storage product which offers enormous scalability and performance benefits. As importantly, Oracle NoSQL Database also offers excellent *availability* mechanisms. These mechanisms are designed to provide your applications access to data contained in the store in the event of localized hardware and network failures.

This document describes the mechanisms Oracle NoSQL Database uses to ensure your data remains available. The various failover algorithms employed by Oracle NoSQL Database are described here. In addition, this document describes application design patterns you can use to best make use of Oracle NoSQL Database's availability mechanisms. In some cases, there are tradeoffs between ensuring data is highly available, and achieving the highest performance possible. This documents also explores these tradeoffs.

The intended audiences for this document are System Architects or Engineers who want to understand the concepts and issues surrounding data availability when using Oracle NoSQL Database. In addition, software engineers who are responsible for writing code that interacts with an Oracle NoSQL Database store should also read this document.

This document assumes you have read and are familiar with the contents of the *Oracle NoSQL Database Getting Started with the Tables API* guide or the *Oracle NoSQL Database Getting Started with the Key/Value API* guide. If you have not read one of those manuals, you should do so before continuing with this document. In particular, you should understand the concepts described in these sections:

- KVStore Architectural Overview

  This section introduces terms and concepts you need to know before reading this document.

- Durability

  This section includes concepts that lead to issues surrounding write availability.

- Consistency

  This section includes concepts that lead to issues surrounding read availability.

Replication Overview

Fundamentally, Oracle NoSQL Database ensures data durability and availability using a single-master replication strategy. That is, a single machine is used to perform write operations, and then to broadcast those operations to multiple read-only replicas.

As mentioned in the KVStore Architectural Overview found in *Oracle NoSQL Database Getting Started with the Tables API*, a shard is a collection of Replication Nodes with a single master and multiple replicas. Your store contains multiple shards, and your data is spread evenly across all the shards in use by your store.

When you perform a write operation in your store, that write operation is performed on the master Replication Node in use by the shard which contains your data. The master performs

this write according to whatever durability guarantees are in place at the time. If you set the durability guarantee strong enough, the master will require the participation of some or all of the replicas in the shard in order to perform the write.

Also, if the master should become unavailable for any reason (such as due to a network or hardware outage), the replicas in primary zones will hold an election to decide which of the remaining nodes should take over as the master. The node with the most up-to-date data will win the election.

The election is decided based on a simple majority vote. This means that a majority of the nodes in the shard in primary zones must be available to participate in the election in order for a new master to be selected.

Loss of a Read-Only Replica Node

The most trivial failover case is if a replica is lost due to a problem with the machine upon which it is running. This loss can be due to something as common as a hard drive failure.

In this case, the only shard that is affected is the one using the replica. By default, the effect on the shard is one of reduced read throughput *capacity*. The shard itself is capable of continuing normal operations. However, it has lost a single Replication Node so its *capacity* to service read requests is lessened by whatever read throughput a single host machine offers your store. Whether you will notice this reduction in read throughput capacity depends on how heavy of a read load your shard is experiencing. The shard could easily have a low enough read load that you will never notice any kind of performance hit due to the loss of the replica.

Note that the previous paragraph assumes a single host machine contains one and only one Replication Node. If you have configured your store such that multiple Replication Nodes run on a single machine, then the throughput capacity implications are multiplied accordingly. It is likely that the loss of a machine running multiple Replication Nodes will affect the throughput capacity of more than one shard because it is unlikely that all the Replication Nodes on that machine will belong to the same shard. Again, whether you notice any actual performance hits due to the loss of the Storage Node depends on how heavy of a read load the individual affected shards are experiencing.

In this scenario, with one exception the shard will continue servicing write requests, and may be able to do so with no changes to its write throughput capacity. The master itself is not affected, so it can continue performing writes and replicating them to the remaining replicas in the shard. There may, however, be reduced write throughput capacity if:

- there is such a heavy read load on the shard that the loss of one replica saturates the remaining replica(s); and
- the master requires an acknowledgement before finishing a write commit.

In this scenario, write performance capacity can be reduced either because the master is continually waiting for the replica to acknowledge commits, or because the master itself is expending resources responding to read requests. In either case, you may see degraded write throughput, but by how much depends on how heavy the read/write load actually is on

the shard. Again, it is very possible that you will never notice an actual reduction in write throughput just because the write load on the shard is low.

In addition, the loss of a single read-only replica can cause all write operations at that shard to fail with a `DurabilityException` exception. This will happen if you are using a durability guarantee that requires acknowledgements from all replicas in the shard in primary zones. In this event, writes at that shard will fail until either that replica is brought back online, or a less strict durability guarantee is put into use.

Durability guarantees requiring acknowledgements from all replicas in primary zones offer you the strongest data durability possible (by making absolutely certain that your writes are replicated to every machine in a shard). But at the same time, they open the potential for the loss of write capabilities for an entire shard due to the failure of a single piece of hardware. Consequently, you need to balance your durability requirements against your availability requirements, and configure your store and related code accordingly.

Loss of a Read/Write Master

If you lose a host machine containing a shard's master, then for a brief moment (that may not even be noticeable to your client code), that shard will not be capable of responding to write requests. Note that only the shard containing the master is affected by this outage; all other shards will continue to perform as normal.

In this case, the shard's replicas in primary zones will quickly notice the master is missing and call for an election. Typically this will occur within a few tens of milliseconds of losing the master.

The election will then be conducted, and the replica in a primary zone with the most up-to-date set of data will be elected master. To be elected master requires a simple majority vote from the other machines in the shard hosting nodes in primary zones. (This simple majority requirement has implications if many machines are lost from your store, so remember it.)

Once a new master is elected, the shard will continue operations, albeit with its read throughput capacity reduced by one machine. As is the case with the loss of a single replica (see the previous section), all write operations can continue so long as your durability guarantee does not require acknowledgements from all replicas in primary zones.

Note that your client code will not notice the missing master if the new master is elected and services the write request within the timeout value in use for the write operation. Nevertheless, you should write your production code to guard against timeout problems. What you should do in the event of a timeout is a policy that you must decide for your code. Options include immediately retrying the write operation, waiting a short time and then retrying the write operation, or abandoning the write operation entirely.

Network Partitions

A network partition occurs when a piece of networking gear (such as a router) fails in such a way as to divide a shard into two separate, non-communicating networks. How the store responds to an event such as this depends on how the shard's Replication Nodes are divided by the network partition.

In the simplest case, a single Replication Node is isolated from the rest of the shard. If the Replication Node is a read-only replica, the shard will continue operating as normal, albeit with the reduced read throughput capacity caused by the loss of a single machine. See Loss of a Read-Only Replica Node (page 4) for more details.

If the single Replication Node is a master, then the shard will handle that in the same way as it would when losing a master: the shard will hold an election to select a new master and then continue operating as normal. See Loss of a Read/Write Master (page 5) for more information.

Things become more complicated when the network partition divides the shard into two or more groups of machines. In this scenario, you will have at least one *minority node partition*. That is, a partition containing less than a majority of the Replications Nodes in the shard. There might also be a *majority node partition* — a partition with the majority of nodes in the shard — but this is not necessarily a given, especially if the partition creates more than two sets of Replication Nodes.

How failover is handled in this scenario depends on whether a majority node partition exists and if the master exists in that partition. There are also issues having to do with the durability and consistency policies in use at the time of the partition.

Master is in the Majority Node Partition

Suppose the shard is divided into two partitions. Partition A contains a simple majority of the Replication Nodes in primary zones, including the master. Partition B has the remaining nodes.

- Partition A will continue to service read and write requests as normal, albeit with a reduced read throughput caused by the loss of however many Replication Nodes are in Partition B. The caveat is that the durability policy in use at the time might prevent writes if there are not enough replicas in Partition A from primary zones to satisfy the durability policy. So long as the durability policy requires a simple majority of replicas, or less, then the shard will be able to service write requests.

- Partition B will continue to service read requests as normal, albeit with increasingly stale data. Partition B might cease to service read requests, depending on the consistency guarantee in place. If a version-based consistency is in use, then Partition B will probably encounter `ConsistencyException` exceptions soon after the partition occurs due to its inability to obtain version tokens from the master. Likewise, if a time-based consistency policy is in use, then `ConsistencyException` exceptions will be seen as soon as the replica lags too far behind the master (from which it is no longer receiving write updates). Note that by default, no consistency guarantee is required to service read requests. So unless you explicitly create and use a consistency policy, Partition B will continue to service read requests through the entire network outage.

  Partition B will attempt to elect a new master, but will be unable to do so because it does not contain the simple majority of Replication Nodes required to hold an election.

Further, if the partition is such that your client code can reach Partition A but not Partition B, then the shard will continue to service read and write requests as normal, albeit with a reduced read capacity.

However, if the partition is such that your client code can read Partition B but not Partition A, then the shard will not be able to service write requests at all. This is because Partition A contains the master, and Partition B does not include enough Replication Nodes to elect a new master.

Master is in the Minority Node Partition

Suppose the shard is divided into two partitions. Partition A contains a simple majority of the Replication Nodes from primary zones, but NOT the master. Partition B has the remaining nodes, including the master.

Assuming both partitions are network accessible by your client code, then:

- Partition A will notice that it no longer has a master. Because Partition A has at least a simple majority of the Replication Nodes in primary zones, it will be able to elect a new master. It will do this quickly, and the shard will then continue operations as normal.

  Whether Partition A can service write requests will be determined by the durability policy in use. So long as the durability policy requires a simple majority of replicas, or less, then the shard will be able to service write requests.

- Partition B will continue to operate as normal, believing that it has a valid master. However, the only way Partition B can service write requests is if the durability policy in use requires no participation from the shard's replicas. If a majority of nodes in primary zones must acknowledge the write operation, or if all nodes in primary zones must acknowledge, then the partitions will not be able to service writes because not enough nodes will be available to satisfy the durability policy.

  If durability NONE is in use, then for the period of time that it takes to resolve the network partition, the shard will operate with two masters. When the partition is resolved, the shard will recognize the problem and correct it. Because Partition A held a valid election, writes performed there will be kept. **Any writes performed in Partition B will be discarded.** The old master in Partition B will be demoted to a simple replica, and the replicas in Partition B will all be synced up with the new master.

  ### Note

  Because of the potential for loss of data in this scenario, Oracle **strongly** recommends that you do NOT use durability NONE. The only time you should use that durability setting is if you want to absolutely maximize write throughput, and you absolutely do not care if you lose the data.

Further, if the partition is such that your client code can reach Partition A but not Partition B, then the shard will continue to service read and write requests as normal, albeit with a reduced read capacity, but only after an election is held.

However, if the partition is such that your client code can read Partition B but not Partition A, then the shard will not be able to service write requests at all, unless you use the weakest durability policy available. This is because Partition B does not include enough Replication Nodes to satisfy anything other than the weakest available durability policy.

No Majority Node Partition

Suppose the shard is divided into multiple partitions such that no partition contains a majority of the Replication Nodes in the shard. In this case, the shard's partitions can service read requests, so long as the consistency policy in use for the read supports it. If the read requires tight consistency with the master, and the master is not available to ensure the consistency can be met, then the read will fail.

The partition containing the master can service write requests only if you are using the weakest available durability policy (that is, a policy which requires acknowledgements from NO replicas). If acknowledgements are required, then there will not be enough replicas to satisfy the durability policy and so no writing can occur.

Once the network partition is resolved, the shard will elect a new master, synchronize all replicas with it, and then continue operations as normal.

Zone Failover

Zones allow you to spread your store across physical installation locations. The different locations can be anything from different physical buildings near each other, to different racks in the same building. The basic idea is that you can guard against large scale infrastructure disruptions, such as power outages or storm damage, by placing the nodes in your store physically as far apart as is possible.

Oracle NoSQL Database provides support for two kinds of zones. *Primary* zones contain nodes which can serve as masters or replicas. Zones are created as primary zones by default. *Secondary* zones contain nodes which can only serve as replicas. Secondary zones can be used to make a copy of the data available at a distant location, or to maintain an extra copy of the data to increase redundancy or read capacity.

Both types of zones require high throughput network connections to transmit the replication data required to keep replicas up-to-date. Failing to provide sufficient network capacity will result in nodes in poorly connected zones falling farther and farther behind. Locations connected by low throughput network connections are not suitable for use with zones.

For primary zones, in addition to network throughput, the network connections with other primary zones should provide for highly reliable and low latency communication. These capabilities make it possible to perform master elections for quick master failovers, and to provide acknowledgments to meet write request timeout requirements. Primary zones are not, therefore, suitable for use with an unreliable or slow wide area network.

For secondary zones, the nodes in the secondary zones do not participate in master elections or acknowledgments. For that reason, the system can tolerate reduced reliability or increased latency for connections between secondary zones and the primary zones. The network connections still need to provide sufficient throughput to support replication, and must provide sufficient reliability that temporary interruptions do not interfere with network throughput.

If you deploy your store across multiple zones, then Oracle NoSQL Database tries to physically place at least one Replication Node from each shard in each zone. Whether Oracle NoSQL

Database can do this is dependent on the number of shards in use in your store, the number of zones, the number of Replication Nodes, and the number of physical machines available in each zone. Still, Oracle NoSQL Database makes a best-effort to spread Replication Nodes across the available zones. Doing so guards against losing entire shards should the zone become unavailable for any reason.

All of the failover descriptions covered earlier in this book apply to zones. Failover works across zones in the same way as it does if all nodes are contained within a single zone. What zones offers you is the ability for your data to remain available in the event of a large outage. However, read and write capability for any given shard is still gated by whether the remaining zone(s) constitute a majority node partition, and the durability and consistency policies in use for your store activities.

Durability Summary

Throughout this document we have described how your durability guarantees affect a shard's write availability in the event of hardware or network failures. In summary:

- A durability guarantee which requires no acknowledgements from the shard's replicas gives you the best possible chance that the shard can continue servicing write requests in the event of an outage. However, this durability guarantee can also result in the shard operating with two masters, which will lead to data loss once the hardware problems are resolved. It is **not** a recommended configuration.
- A durability guarantee which requires a simple majority of replicas in primary zones to acknowledge the write operation guards against the accidental condition of two masters operating at one time. However, it also means that the shard will not be capable of servicing write requests if more than a majority of those replicas are taken offline due to the hardware failure.
- A durability guarantee that requires all replicas in primary zones to acknowledge the write operation guards against any possibility of data loss. However, it means that the shard will lose its ability to service write requests if even one of those replicas is taken offline.

Consistency Summary

In almost all cases, replicas will continue to service read requests so long as the underlying hardware remains functional. In its default configuration, there is nothing that stops a replica from doing this, even if it is the only node left running after some kind of catastrophic failure.

However, it is possible that a replica will stop servicing read requests in the event of a network failure if the consistency policy in use requires either version information, or disallows stale data relative to the master. Whether this happens depends exactly on how your Replication Nodes are partitioned as the result of the failure, and how long it takes to establish a new master. The replica's ability to service the read request will also be gated by the consistency policy in use for that request. If the read requires tight consistency with the master, and the master is not available to ensure the consistency can be met, then the read will fail.