Developing Applications with Cross Data Center Replication (XDCR)
Contents

Introduction .................................................................................................................................. 3
Some Terminology First! ............................................................................................................. 3
XDCR Overview........................................................................................................................ 3
Benefits ...................................................................................................................................... 4
Topologies ................................................................................................................................ 6
XDCR Architecture .................................................................................................................. 7
Consistency, Availability and Partition Tolerance with XDCR........................................... 7
Developing Applications with XDCR ..................................................................................... 9
Failover with XDCR................................................................................................................ 9
Latency with XDCR.................................................................................................................. 10
Consistency Considerations with XDCR .............................................................................. 10
Other Programmability Considerations .............................................................................. 13
Conclusion............................................................................................................................... 13
Additional References............................................................................................................ 13
Introduction

Mission critical applications need a database that is always on. While intra-cluster replication helps with server failures, giving you high availability, your system still remains at risk from catastrophic failures – loss of a data center, power loss, natural disaster, etc. You may also need a way to improve the response time for users when they are globally distributed, by replicating information closer to them. Other mission critical applications may need separation of operational workloads from your analytics workloads. Cross data center replication (XDCR) in Couchbase Server answers these requirements.

In this paper, we will look at how XDCR works and what considerations are important for developers who are building applications that work with multiple Couchbase Server clusters connected with XDCR technology.

Some Terminology First!

Before we get into specifics of XDCR, let us briefly recap the various availability and consistency terms and metrics.

Let's start with basic metrics used for enumerating failover characteristics.

- **Recovery Time Objective (RTO):** Refers to how fast the application gets back to a fully working state after the failure.

- **Recovery Point Objective (RPO):** Refers to quantity of data loss expected after the application gets back to a full working state after the failure.

Data consistency, availability, fault tolerance and generally the CAP theorem will be at the heart of most things we discuss. For these discussions, the two data consistency models that are relevant to replicated architectures are key to understand.

- **Immediate Consistency with Synchronous Replication**
  Describes systems that only acknowledge mutations to the application after replicated copies confirm execution of the mutation. In other words, it describes $RPO = 0$ after failover, coupled with higher RTO and higher latencies for reads and writes.

- **Eventual Consistency with Asynchronous Replication**
  Describes systems that acknowledge mutations immediately to the application as soon as a single copy confirms execution of the mutation. In other words, it implies $RPO > 0$ after failover coupled with lower RTOs.
XDCR Overview

XDCR is a high performance asynchronous data replication system. XDCR provides eventual consistency of data between two clusters of Couchbase Servers. XDCR complements the local replication mechanism that is used to maintain replicas across nodes of a single cluster. Local replication works well under lower latencies, providing high availability under local failures. However, XDCR can be deployed across sites with high latencies and provides disaster recovery and high availability under cluster wide failures. XDCR provides the ability to read or write to any cluster and provides a simple conflict resolution mechanism to resolve write conflicts.

XDCR is a perfect fit for a number of problems, but the following list details the top scenarios where XDCR shines:

- **High Availability and Disaster Recovery with XDCR:** Mission critical systems that need protection against geographical disasters can set up XDCR in a unidirectional form to create a hot backup system or in a bidirectional form to create a load balanced set of clusters to service their workloads. If some of the clusters in the XDCR setup experience cluster-wide disasters such as power loss or total connectivity loss, remaining clusters can continue to serve the application workload.

- **Data Locality with XDCR:** Network latency around the world can be high and many global applications bring their data closer to their users to provide a responsive and fluid experience. XDCR can allow replication of data effectively between continents and eliminate the need for applications to reach across continents for their interactive experiences.

- **Load Separation with XDCR:** Today, many applications require predictable low latencies from their data processing platforms. Variances in latencies can create complications. Separating low latency workloads from heavy analytics processing can prevent these complications. XDCR can provide the ability to set up this isolation by moving data between clusters. Many customers set up a separate analytics environment with XDCR and fine-tune their indexing and performance to meet specific needs of the workloads.
Benefits

XDCR comes with a great set of benefits, the most important of which are highlighted here:

- **High Throughput:** Couchbase splits all data in a bucket into 1024 vbuckets. When moving data, vbuckets in source and destination pair up and independently move data without any centralized funnel. This architecture allows very high throughput and lower latency for data movement between clusters both for the initial synchronization as well as the continuous streaming of data.

- **Powerful Topology Support:** XDCR provides a great basic building block: unidirectional replication. Unidirectional replications can be utilized to construct more complex topologies including circular/peer-to-peer replication topologies, cascading/chained topologies, and well known topologies like hub-and-spoke for data aggregation scenarios. We will talk about topologies in detail in the next section.

- **Simplified Administration:** XDCR provides a greatly simplified setup and administration interface. XDCR replication requires a cluster reference and a replication setup per bucket that points to the source and destination buckets.

XDCR also requires no intervention under cluster topology changes. Source and destination clusters can have different hardware profiles, node counts, bucket settings or view definitions and freely add/remove nodes. XDCR also support replication between different versions of Couchbase Server to allow rolling upgrade of clusters tied together with XDCR. Even if there are individual node failures, transient or long term network issues, XDCR resumes and restarts automatically.

XDCR also provides simplified secure communication setup. With Couchbase Server 2.5 or later, administrators can use secure communication for XDCR between Couchbase clusters.

You can find additional information on XDCR use cases and benefits on our [online documentation](#).
Topologies

XDCR comes with powerful topology support. You can set up complicated circular and cascading replication topologies from star to mesh to full-circular topologies. All of these topologies are derived from a simple building block: unidirectional replication. Some of the topologies are illustrated in the picture below.

- **Unidirectional Replication**: Unidirectional replication is one-way replication, where data from the source cluster gets replicated to the destination cluster. Unidirectional topology is used typically where you need to create an offsite standby cluster or in cases where you need an offsite read-only cluster for lower latency reads or read-heavy traffic. Another common use case for unidirectional XDCR is load separation. With this topology, you can replicate data with XDCR and index the destination cluster to optimize your analytics workload and isolate it from your online data processing on the source cluster.

- **Bidirectional Replication**: Allows two clusters to replicate data to each other in a circular topology. Setting up bidirectional replication in Couchbase Server involves setting up two unidirectional replication links between the clusters involved. Bidirectional replication is useful when you want to load balance your entire read and write workload across two clusters. Some applications distribute their workload randomly and use multi-master writes and can tolerate write conflicts. The majority of applications control workload for specific sets of data to reduce/eliminate overlapping writes and avoid conflicts in most cases.

Using these building blocks, you can set up infinite number of topologies between clusters. The figure below provides some examples of such complex topologies XDCR supports.
XDCR architecture is inherently flexible, high throughput and high performance. A few properties are key to achieving these.

XDCR decentralizes replication for throughput and performance: Couchbase Server buckets are split across physical partitions called *vbuckets* as they are distributed across the nodes of the cluster. Based on the placement, each node ends up with a number of active and replica vbuckets. XDCR uses change detection on active vbuckets as changes are persisted to disk. XDCR workers pick up the changes for the given active vbucket on the node and communicate the changes to the remote node that maintains the active vbucket at the destination cluster. The architecture does not require any centralized coordination for replication of each vbucket to provide the best latency and throughput characteristics.

Each node maintains a number of workers to round robin among active vbuckets that can be configured through the “XDCR Max Replications per Bucket” setting.

XDCR also maintains checkpoints to create a resume point in case a failure happens on the source or destination nodes. When replication resumes after a communication failure, the last checkpoint serves as the resume point instead of retransmitting all data from scratch. The checkpoint interval can also be configured through the “XDCR Checkpoint Interval” setting.

XDCR workers communicate in batches and batching characteristics can be tuned by change count or by the size of the batch through the “XDCR Batch Count” and “XDCR Batch Size” settings.

XDCR replicates changes in 2 modes: Pessimistic and Optimistic. Pessimistic replication sends only the keys in the batch while replicating. If the keys are accepted by the destination, values are replicated with another round trip. In optimistic mode, keys and values are shipped together in a single batch. The behavior is configurable based on your workload through the “XDCR Optimistic Replication Threshold” setting.
You can find detailed discussion on XDCR architecture in the [online documentation](#). There is also a detailed discussion of setting up and administrating XDCR on the [online documentation](#).

## Consistency, Availability and Partition Tolerance with XDCR

CAP (consistency, availability and partition tolerance) Theorem is well known among database professionals and can act as a good high level scheme for guiding your application model. In a sentence, XDCR provides an available and partition loss tolerant system for applications. Let's detail what that means a little more with a table comparing single cluster setup as well as various XDCR topologies.

As you examine the table, It's important to note that terms in the CAP theorem – consistency, availability and partition tolerance – can have different interpretations based on fault domains (what types of faults we talk about). So use the table below as a starting point, but for best results continue to keep various fault details and system behavior details in mind.

<table>
<thead>
<tr>
<th>Deployment Topology</th>
<th>Fault Domain Protection</th>
<th>CAP Balance (CAP = Consistency, Availability, Partition Tolerance)</th>
<th>Comments</th>
</tr>
</thead>
<tbody>
<tr>
<td>Single Couchbase Server Cluster</td>
<td>Node level failure domain (example, HW failures, communication failures between nodes)</td>
<td>Can be configured CP and can be tuned to be Available through auto failover or with replica reads and more.</td>
<td>Couchbase Server allows reading active or replica vbuckets thus can be tuned with auto failover to give you write availability after a short failover timeout.</td>
</tr>
<tr>
<td>Couchbase Server Clusters with Unidirectional XDCR for HA/DR</td>
<td>Node and cluster-wide failures (examples: DC failures caused by natural disasters)</td>
<td>AP across clusters with unidirectional XDCR with protection against cluster wide or node failures. Same CAP balance for within each cluster for node failures.</td>
<td>Unidirectional with passive computational capacity at a second site/data center. Destination cluster can be used for eventually consistent reads during steady state and can be promoted to read/write when source cluster fails.</td>
</tr>
<tr>
<td>Couchbase Server clusters with bidirectional XDCR for HA/DR</td>
<td>Node and cluster-wide failures (examples: DC failures caused by natural disasters)</td>
<td>AP across clusters with bidirectional XDCR with protection against cluster wide or node failures.</td>
<td>Bidirectional with active/split computational capacity across sites/data-centers. Destination cluster can be use for eventually consistent reads and writes during steady state. You can experience write conflicts if the same key is mutated at both clusters. Many customers minimize the write conflicts by segmenting write traffic to non-overlapping key ranges across source and destination clusters.</td>
</tr>
</tbody>
</table>
Developing Applications with XDCR

Couchbase Server guards against local failures such as hardware problems or OS crashes on a given node through intra-cluster replication. In case of a failure, automatic or manual failover of the node promotes replica vbuckets to active vbuckets on remaining nodes. Many mission critical apps however require protection against cluster wide failures. XDCR protects against these failures by allowing replication of data between clusters across long distances. There are a number of considerations for applications when targeting a single cluster vs multiple clusters connected through XDCR. The top three considerations are:

- Failover behavior with a single vs. multiple clusters
- Latency of replication within a single vs. multiple clusters
- Consistency model considerations with a single vs. multiple clusters

In this section we will go through details of these differences and best practices for developing applications that target multiple clusters.

Let’s first start with a high level discussion on consistency, availability and partition tolerance around XDCR.

Failover with XDCR

As mentioned above, Couchbase Server provides auto and manual failover capabilities within a single cluster. However with XDCR, under cluster wide failures, applications are responsible for managing failover with XDCR and typically, applications do this based on server error messages and operation timeouts. Once the failure is detected, most applications implement a retry period coupled with a failover path. Failover happens by switching the connection to another cluster. RTO can depend on how aggressively the applications detect the failure with retries and timeouts. With XDCR, applications have full control over how this happens. However, as XDCR replicates asynchronously, RPO can be as long as the XDCR latency.

How aggressively applications fail over may also increase the chances of “false” failovers. For example, clusters experiencing network hiccups or transient resource contents may cause applications to timeout but quickly after may come back to life and continue to drain the remaining mutations in their queues. The following timeline demonstrates the lost update with premature failover:

- At T1, the application adds a document key doc#1 to the East Cluster. The value gets a sequence/revision ID of 0 (rev0).
- At T2, this mutation is replicated to the West Cluster through XDCR.
- At T3 and T4, two additional updates happen on doc#1. Revision value is incremented to 2 with the updates.
- At T5, the changes have not been replicated to the West Cluster but the application times out. After the application fails over to the West Cluster, it does an additional update. However, since the West Cluster only has rev0, the update gets a revision ID of 1.
- At this point, XDCR catches up, and at T7 replicates the doc#1 (rev2). After a premature failover from East Cluster to West Cluster, East Cluster comes back to life and causes a lost update made at T6, because its revision ID is lower than the revision ID replicated by XDCR.
Obviously, under a true failure of the East Cluster, you won't receive any updates from XDCR thus don't experience the lost update. Some applications can gracefully handle the issue and others choose to expand the timeout period to above milliseconds to avoid these situations.

**Latency with XDCR**

Both intra-cluster replication and XDCR receive mutations with some latency from the active vbuck. Under steady state, latency in intra-cluster replication is much lower than XDCR since nodes of a cluster are located in closer proximity than a separate cluster. Replication latency in XDCR is dependent on a number of factors:

- Topology of the cluster
- Rate of mutations on the source
- The network latency between 2 clusters
- XDCR settings
- and more

Applications working with XDCR have to be aware of the increased latency especially during failover. Due to increased latency, RPO may be larger in XDCR failovers compared to failovers within the cluster.

**Consistency Considerations with XDCR**

- **Eventual Consistency with XDCR**

  The XDCR consistency model can be characterized as eventually consistent under regular operation. Even though it is extremely rare, there are conditions when eventual consistency may be compromised.

  - **Tombstone cleanup**: When data is deleted or expired on the source cluster, the deletion operation is maintained through a tombstone to allow replication of the deletion to the destination cluster with XDCR. However, tombstone cleanup can be configured to run faster than the replication latency, which means the tombstone can be cleaned up before XDCR is able to detect and replicate these delete and expiry operations. The default setting for tombstone cleanup is set to 3 days, which means tombstones will be maintained for 3 days. Dialing this value too low or experiencing extended failures in XDCR that prevent replication between clusters over the setting for this value can cause consistency to be compromised between source and destination cluster. The effect would be a set of
missed deletes and expiries on the destination cluster.

- **Read and Write Conflicts with XDCR**

  With intra-cluster replication in Couchbase Server, there is a designated active vbucket and a set of replica vbuckets. All reads and writes by default happen on the active vbucket. Applications can specify flags to read secondary replicas if they can deal with reading out-of-date information, but today writing to secondary replicas is not possible.

  However, unlike intra-cluster replication, applications with XDCR have the option to read or write to any cluster. Let’s take a look at specific topologies to understand the flavors of data inconsistencies an application has to deal with.

- **Unidirectional XDCR**

  Typically applications using unidirectional topology perform writes to the source cluster but may perform reads on both the source and the destination clusters. That means, under this topology, applications don’t have to deal with conflicting writes under steady state. How about unsteady state, like the behavior during failover? Well, if the application fails over write operations from the source cluster to the destination cluster and doesn’t drain all mutations from the source to destination cluster, the application may have to deal with write conflicts.

  What if you want to read from the destination cluster? That is certainly possible and fairly common. Reading from the destination cluster is not much different in nature from reading secondary replicas with intra-cluster replication. Applications reading from the destination cluster have to be able to tolerate reading out of date information, which means your write may not immediately be available on the destination cluster. If the application expects to be able to read its own write, you should perform reads on the cluster you performed your write on.

- **Bidirectional XDCR**

  Typically applications using bidirectional topology perform all their reads and writes to both clusters. However, we find two distinct traffic models for writes with bidirectional XDCR:

  - **Active/Active clusters without overlapping reads or writes**: Such applications use a load balancing technique to route parts of their traffic to one of the active clusters/data centers. The routing guarantees that a given unique key in the bucket always reliably gets accessed in one of the clusters as long as the environment does not experience a failure (a.k.a routing with stickiness). Both active clusters receive non-overlapping read/write operations across the active clusters. This traffic model has the same experience for the read and write conflicts as the unidirectional topology above.

  - **Active/Active clusters with overlapping read or writes**: Such applications also use a load balancing technique but subsequent requests of the same key may be routed to another active cluster at any time (a.k.a routing with no stickiness). You need to be aware that there will be lost updates and write conflicts if simultaneous writes happen within the window of XDCR latency. You may also not be able to read your own write if you switch between clusters, if reads happen within the window of XDCR latency. In this topology it is critical to understand the behavior of conflict resolution to ensure tolerance to these conditions exist in the application. In the next section we will detail how conflicts are resolved in XDCR.
Resolving Conflicts: “Most updates wins” vs. “Last update wins”

With Bidirectional replication, conflict resolution is also important to understand. XDCR conflict resolution uses revision IDs as the first field to resolve conflicts between two writes across clusters. Revision IDs are maintained per key and are incremented with every update to the key. Revision IDs keep track of number of mutations to a key thus XDCR conflict resolution can be best characterized as “the most updates wins”.

For example, in the following case imagine an East and a West cluster setup with bidirectional replication. An applications updates the doc#1 in the East Cluster at time T1. XDCR replicates data over to the West Cluster at T2. Another application makes three additional updates to doc#1; but ahead of replication back to East Cluster, at time T6, a third application updates the East Cluster. However, since only revision ID 0 is present in the East Cluster, the update increments revision ID to 1. Once XDCR replication catches up, the last value written at time T6 is lost, since the revision ID 3 > 1.

Lost updates can also happen across update and delete operations. Update conflicts based on updates and deletes are also resolved with the same mechanics. For example, if an application updates a given key in the East Cluster three times in a row and before XDCR replication kicks in, and on the West Cluster a delete operation is performed on the same key, the East Cluster with three updates wins the XDCR exchange and data is resurrected in the West Cluster.

How about keys that get deleted and recreated?

At this point, you may be worried that the above scenario may cause you a ton of headache if you are in the habit of deleting data and reusing the same keys to recreate these keys. Worry not. XDCR protects you against deletes and recreates of keys by maintaining the deleted item revision number and ensuring the revision numbers do not to revert back to 0 under a recreate of the same key.
There are additional fields to help resolve conflicts in cases where revision IDs are the same. You can find details of the conflict resolution in the online documentation.

Other Programmability Considerations

Couchbase API provides a number of capabilities across SDKs. It’s worth mentioning a few of the behavioral differences in using the capabilities against a single cluster vs. multiple clusters tied with XDCR.

**CAS Token:** CAS (check-and-set) token provides optimistic concurrency. CAS token is maintained per key and with every mutation to a value, CAS token is updated. Couchbase SDK provides capabilities to get CAS token and use CAS for ensuring no other operation is interleaved between the get and the set. You can use CAS to detect changes from updates other than XDCR as XDCR replicates the CAS value associated with each key. That means, with incoming XDCR mutations the CAS value is updated to the CAS of the original mutation on the source cluster and not a new independent CAS value.

**Locking Items:** Couchbase Server also supports pessimistic concurrency through a lock operation. However locking operation is only valid for locking keys within the cluster. Locks do not extend to the same key values on other clusters connected with XDCR.

**Expiring Items:** Expiry works as expected with XDCR, so applications don’t have to be concerned with this flag. The metadata on expiry is replicated with the keys. If an item expires before it is replicated through XDCR, the deletion of the item is still replicated. It is important to note that expiry is based on UTC time of the node (UTC represents a coordinated universal time that is time-zone independent). However local clocks across nodes and cluster may have skews. If a node is running ahead, it may expire values first before other copies of the same data with the same expiration time.

Conclusion

Couchbase Server XDCR provides a great complementary addition to intra-cluster replication to protect your application against additional failure domains. To take full advantage of these capabilities, developers need to understand how consistency, availability and fault tolerance with XDCR are provided to ensure it delivers the full benefits for your applications.

Additional References

- **XDCR Overview:** Extend your data tier with XDCR [http://info.couchbase.com/whitepaper-cross-datacenter-replication.html](http://info.couchbase.com/whitepaper-cross-datacenter-replication.html)

- **XDCR Online Documentation:** [http://docs.couchbase.com/admin/admin/XDCR/xdcr-intro.html](http://docs.couchbase.com/admin/admin/XDCR/xdcr-intro.html)

- **Administrating XDCR:** [http://docs.couchbase.com/admin/admin/UI/ui-xdcr-panel.html](http://docs.couchbase.com/admin/admin/UI/ui-xdcr-panel.html)
About Couchbase

Couchbase provides the world’s most complete, most scalable and best performing NoSQL database. Couchbase Server is designed from a simple yet bold vision: build the first and best, general-purpose NoSQL database. That goal has resulted in an industry leading solution that includes a shared nothing architecture, a single node-type, a built in caching layer, true auto-sharding and the world’s first NoSQL mobile offering. Couchbase Mobile, announced in 2013 is a complete NoSQL mobile solution comprised of Couchbase Server, Couchbase Sync Gateway and Couchbase Lite – a lightweight NoSQL database designed for the device. Couchbase counts many of the worlds biggest brands as its customers, including Amadeus, Bally’s, Beats Music, Cisco, Comcast, Concur, Disney, Orbitz, Rakuten / Viber, Sky, Tencent, Walmart and Verizon, as well as hundreds of other household names worldwide. Couchbase is headquartered in Silicon Valley, and is funded by Accel Partners, Adams Street Partners, Ignition Partners, Mayfield Fund, and North Bridge Venture Partners.