Alachisoft

NCache

Couchbase Server vs. NCache

For .NET & Java Applications

Couchbase Server 3.0 vs. NCache 4.3 SP1

This document compares Couchbase and NCache. Read this comparison to:

- Understand Couchbase and NCache major feature differences.
- See how Couchbase and NCache compare on qualitative aspects such as performance, scalability, high availability, data reliability, and administration.

Table of Content

Disclaimer 1
1 Executive Summary 2
2 Qualitative Differences Explained 5
2.1 Performance and Scalability 5
2.2 Cache Elasticity (High Availability) 7
2.3 Cache Topologies 8
2.4 WAN Replication 10
2.5 Cache Administration 11
2.6 Security & Encryption 13
2.7 Object Caching Features 14
2.8 Managing Data Relationships 15
2.9 Synchronization with Data Sources 16
2.10 Runtime Data Sharing 17
2.11 Cache Search (SQL-Like) 19
2.12 Data Grouping 19
2.13 Read-through, Write-through & Cache Loader 20
2.14 Cache Size Management (Evictions Policies) 22
2.15 ASP.NET & Java Web Sessions Caching 23
2.16 Third Party Integrations 24
3 Conclusion 27

Disclaimer

The comparison provided in this document is for the purpose of helping you get a better understanding of Couchbase versus NCache. Information obtained about Couchbase is from the freely available downloads, documents, and forums.

We did not conduct any scientific benchmarks for performance and scalability of Couchbase so our assessment about it may be different from yours. NCache benchmarks are already published on our website (www.alachisoft.com) for you to see.

Additionally, we have made a conscious effort to be objective, honest, and accurate in our assessments in this document. But, any information about Couchbase could be unintentionally incorrect or missing, and we do not take any responsibility for it.

Instead, we strongly recommend that you do your own comparison of Couchbase with NCache and arrive at your own conclusions. We also encourage you to do performance benchmarks of both Couchbase and NCache in your environment for the same purpose.

1 Executive Summary

This document compares Couchbase with NCache, and contrasts their significant differences. This comparison focuses on all the major areas that a good in-memory distributed cache should provide.

Feature	Couchbase Server	NCache
Performance & Scalability		
- Cache Performance	*Please verify yourself*	Excellent
- Cache Scalability	*Please verify yourself*	Excellent
- Bulk Operations	Partial support	Full support
- Async Operations	Partial support	Full support
- Compression	No support	Full support
- Fast Compact Serialization	No support	Full support
- Indexes	Supported	Full support
- Multiple NIC binding	No support	Full support
Cache Elasticity (High Availability)		
- Dynamic Cache Cluster	Partial support	Full support
- Peer to Peer Architecture	Partial support	Full support
- Connection Failover	Partial support	Full support
- Dynamic Configuration	Supported	Full support
- Multiple Clusters	No support	Full support
- Named Caches	Supported	Full support
- Communication Protocols	TCP, REST	TCP, REST (soon)
- Cluster Specific Events	No support	Full support
Cache Topologies		
- Local Cache	Partial support	Full support
- Client Cache (Near Cache)	No support	Full support
- Mirrored Cache	No support	Full support
- Replicated Cache	No support	Full support
- Partitioned Cache	Supported	Full support
- Partitioned-Replica Cache	Partial support	Full support
- Auto Data Balancing	Partial support	Full support
- Rack Awareness for Replicas	Supported	No support
WAN Replication		
- Active – Passive	Supported	Full support
- Active – Active	Supported	Full support
- One Active – Multiple Passive	Supported	No support (soon)
- 3 or More Active	Supported	No support (soon)
- Conflict Resolution	Partial support	Full support
- De-duplication	Supported	Full support
- Data Security	Supported	Full support
Cache Administration		
- Cache Admin (GUI Tool)	Partial support	Full support
- Cache Monitoring (GUI Tool)	Supported	Full support
- PerfMon Counters	No support	Full support
- JMX Counters	No support	Full support
- Command Line Admin Tools	Supported	Full support
- Admin and Monitoring API	Partial support	Full support

Security & Encryption		
- Active Directory/LDAP Authentication	No support	Full support
- Authorization	Supported	Full support
- Data Encryption	No support	Full support
- Secure Communication	Supported	Full support
Object Caching Features		
- Get, Add, Insert, Remove, Exists, Clear Cache	Supported	Full support
- Expirations	Partial support	Full support
- Lock & Unlock	Supported	Full support
- Item Versioning	Supported	Full support
- Multiple Object Versions	Supported	Full support
- Streaming API	Partial support	Full support
Managing Data Relationships		
- Key Based Dependency	No support	Full support
- Multi-Cache Key Dependency	No support	Full support
Synchronization with Data Sources		
- SqlDependency (SQL Server)	No support	Full support
- OracleDependency (Oracle)	No support	Full support
- DbDependency (OLEDB)	No support	Full support
- File Based Dependency	No support	Full support
- Custom Dependency	No support	Full support
Runtime Data Sharing		
- Item Level Events (onInsert/onRemove)	No support	Full support
- Cache Level Events (Add/Insert/Remove)	No support	Full support
- Custom Events (Fired by Apps)	No support	Full support
- Continuous Query	No support	Full support
Cache Search (SQL-Like)		
- Object Query Language (OQL)	Supported	Full support
- OQL on Tags, Named Tags, & Groups	No support	Full support
- LINQ Queries	No support	Full support
Data Grouping		
- Groups/Subgroups	No support	Full support
- Tags	No support	Full support
- Named Tags	No support	Full support
Read-through, Write-through & Cache Loader		
- Read-through	No support	Full support
- Write-through	No support	Full support
- Write-behind	No support	Full support
- Reload Items with Read-through (Expiration, Db Sync)	No support	Full support
- Cache Startup Loader	Partial support	Full support
Cache Size Management (Evictions Policies)		
- Max Cache Size (in MBs)	Supported	Full support
- Least Recently Used (LRU) Evictions	Partial support	Full support
- Least Frequently Used (LFU) Evictions	Partial support	Full support
- Priority Evictions	No support	Full support
- Do Not Evict Option	No support	Full support

ASP.NET Sessions & Java Web Sessions		
- ASP.NET Sessions	No official support	Full support
- ASP.NET Sessions (Multiple Datacenters)	No support	Full support
- ASP.NET View State Cache	No support	Full support
- ASP.NET Output Cache	No support	Full support
- Java Session Persistence	No official support	Full support
Third Party Integrations		
- NHibernate 2nd Level Cache	No official support	Full support
- Entity Framework 2nd Level Cache	No support	Full support
- Memcached Protocol Server	Supported	Full support
- Memcached Smart Wrapper	No official support	Full support
- Hibernate 2nd Level Cache	No official support	Full support
- Spring Integration	No support	Full support
- JCache API	No support	No support (soon)

2 Qualitative Differences Explained

2.1 Performance and Scalability

Performance is defined as how fast cache operations are performed at a normal transaction load. Scalability is defined as how fast the same cache operations are performed under higher and higher transaction loads. NCache is extremely fast and scalable.

See NCache benchmarks for details.

Feature Area	**Couchbase Server**	**NCache**
Cache Performance	*Please verify this yourself.*	Extremely good. NCache uses its own light-weight socket-level protocol for client/server and server/server communication.
Cache Scalability	*Please verify this yourself.*	Extremely good. NCache provides linear scalability, means as you add more nodes to the cluster your performance increase in a linear fashion.
Bulk Operations	Partial support. Only “Get” and “Update” supported. Supported in .NET. Not supported in Java.	Full support. Bulk Get, Add, Insert, and Remove. This covers most of major cache operations and gives great performance boost. Provides for both .NET and Java.
Async Operations	Partial support. Only supported for Java No support for .NET	Full support. Async add, insert, and remove provided. Async operation returns control to the application and performs the cache operation in the background. Improves application performance greatly. Provides for both .NET and Java.
Compression	No support.	Full support. Specify this along with object size threshold and only items larger than the threshold are compressed. Compressing small objects yields no benefit and actually slows things down. And, you can change

Feature Area	Couchbase Server	NCache
		this config at runtime through "Hot Apply". Can keep both compressed and uncompressed objects in the same cache.
Fast Compact Serialization	No support. Provides JSON serialization only. JSON serialization is slow and also creates much larger serialized objects. As a result, data transfer from app to cache is slower. And, cache storage requirement also increases quite a bit.	Full support. NCache lets you register your classes with the cache through a GUI tool (NCache Manager). Then, NCache generates serialization code and compiles it in-memory when your application connects to the cache. This code is then used to serialize objects and it is almost 10 times faster than regular .NET and Java serialization (especially for larger objects). Provides for both .NET and Java.
Indexes	Partial support. Not available with in-memory only configuration. Requires persistent storage. Index creation is asynchronous that may result in retrieving old data; this is data integrity issue. Requires JavaScript programming on server-side for each View. Only indexes JSON documents. Cannot index binary serialized object. Multiple copies of JSON objects stored with each index thereby increasing storage requirement drastically. Also slows down cache operations due to extra server-side index update operation overhead.	Full support. You can use NCache Manager (GUI tool) to create indexes on any attributes of .NET or Java objects. NCache also creates indexes automatically on Tags, Named Tags, Groups, and Subgroups. Expiration and eviction policies also use indexes. NCache generates data extraction code at connection time, compiles it in-memory, and uses it for all data extraction instead of .NET and Java Reflection. This is much faster.
Multiple NIC binding	No support.	Full support. You can assign two NICs to a cache server. One can be used for clients to talk to the cache server and second for multiple cache servers in the cluster to talk to each other. Improves your data bandwidth scalability greatly. You can also assign a specific NIC for a cache client to use for talking to the cache server.

2.2 Cache Elasticity (High Availability)

Cache elasticity means how flexible the cache is at runtime. Are you able to perform the following operations at runtime without stopping the cache or your application?

1. Add or remove cache servers at runtime without stopping the cache.
2. Make cache config changes without stopping the cache
3. Add or remove web/application servers without stopping the cache
4. Have failover support in case any server goes down (meaning are cache clients are able to continue working seamlessly).

This is an area where Couchbase is not as strong as NCache in general. NCache provides a self-healing dynamic cache clustering that makes NCache highly elastic. Read more about it at dynamic clustering.

Feature Area	**Couchbase Server**	**NCache**
Dynamic Cache Cluster	Partial support (has limitations). Data is not automatically rebalanced when a cache server is added or removed at runtime. It requires a manual intervention.	Full support. NCache is highly dynamic and lets you add or remove cache servers at runtime without any interruption to the cache or your application Data is automatically rebalanced (called state transfer) at runtime without any interruption or performance degradation. NCache clients keep on communicating with cache servers, independent of state of server. Clusters ensure execution of client operations even when data balancing is in process.
Peer to Peer Architecture	Partial support. Peer-to-peer architecture not supported with in-memory store. Only available with persistent storage.	Full support. This means that even for in-memory cache, there is no “master” or “slave” nodes in the cluster. There is a “primary coordinator” node that is the senior most node. And, if it goes down, the next senior most node automatically becomes the primary coordinator.
Connection Failover	Partial support. Failover recovery is not automatic by default. Even with automatic, only one server failover is supported. Any subsequent failures will result in application downtime. And, automatic failover recovery requires a minimum of 3-node cache cluster.	Full support. When a cache server goes down, the NCache clients automatically continue working with other servers in the cluster and no interruption is caused. Data of failed node is automatically redistributed between remaining servers in

Feature Area	Couchbase Server	NCache
	Even automatic failover recovery does not rebalance the data. That requires manual intervention.	cluster (If cluster topology has replicas of servers) Cluster auto-manages itself by rebalancing its data, assigning replicas and even redistributed data is replicated to new assigned node's replica automatically, all at runtime without causing any interruption.
Dynamic Configuration	Supported.	Full support. 'Hot Apply' feature is provided to change cluster configuration at runtime, without any need of restarting the server/cluster. Reconfigurable options include most of cluster configurations including cache size, eviction ratio, etc.
Multiple Clusters	No support. Cannot create multiple clusters on same set of cache servers. Only one cluster is created upon which multiple caches can be created.	Full support. NCache allows you to create multiple cache clusters of different configuration and sizes on same set of cache servers.
Named Caches	Supported.	Full support. NCache allows you to create multiple named caches on the same set of cache servers.
Communication Protocol	Binary TCP Text based API REST API	Binary TCP REST API (coming soon) Memcached Protocol Server
Cluster Specific Events	No support.	Full support. NCache provides .NET and Java events about changes in the cluster like: MemberJoined, MemberLeft, CacheStopped, etc.

2.3 Cache Topologies

Cache Topologies determine data storage and client connection strategy. There are different topologies for different type of uses.

Read more details at NCache caching topologies.

Feature Area	Couchbase Server	NCache
Local Cache	Partial support (with limitation). Only OutProc cache provided. Cannot use it as InProc local cache.	Full support. Both InProc and OutProc. You can use NCache as InProc or OutProc local cache. InProc is much faster but your memory consumption is higher if you have multiple application processes. OutProc is slightly slower but saves you memory consumption because there is only one cache copy per server.
Client Cache (Near Cache)	No support.	Full support. Client Cache is simply a local InProc/OutProc cache on client machine but one that stays connected and synchronized with the distributed cache cluster. This way, the application really benefits from this "closeness" without compromising on data integrity.
Mirrored Cache	No support.	Full support. Mirrored Cache is a 2-node active-passive cache and data mirroring is done asynchronously.
Replicated Cache	No support.	Full support. Replicated Cache is active-active cache where the entire cache is copied to each cache server. Reads are super-fast and writes are done as atomic operations within the cache cluster.
Partitioned Cache	Supported.	Full support. You can create a dynamic Partitioned Cache. All partitions are created and clients are made aware all at runtime. This allows you to add or remove cache servers without any interruption.
Partitioned-Replica Cache	Partial support (with limitation). Replication is not supported for in-memory cache. Only supported with persistence. Partition is replicated asynchronously. Client must specify sync-replication thru API.	Full support. Same as Partitioned Cache and is fully dynamic except there is also a "replica" for each partition kept at another cache server for reliability. Replica is created and data rebalanced

Feature Area	Couchbase Server	NCache
	Data rebalancing is not automatic and must be done manually. This makes things not very dynamic because if one server goes down and before manual rebalancing is done, another server goes down, you lose data.	automatically at runtime. Replication can be configured at cache level as synchronous or asynchronous.
Auto Data Balancing (Partitioned & Partition-Replica)	Partial support. Data balancing is not automatic. And, manual data balancing requires immediate human intervention if a cache server goes down in production.	Full support. Data is automatically rebalanced when you add/remove cache servers from the cluster. Data is also rebalanced automatically when one cache server has a lot more data than other servers. You can configure the threshold of difference for this. You can turn off auto rebalancing in this case and manually do it if you wish.
Rack Awareness for Replicas (Partition-Replica)	Supported. No support with in-memory storage. Only available with persistent storage. Can configure a group of cache servers so replicas are distributed among them for location affinity.	No support

2.4 WAN Replication

WAN replication is an important feature for many customers whose applications are deployed in multiple data centers either for disaster recovery purpose or for load balancing of regional traffic.

The idea behind WAN replication is that it must not slow down the cache in each geographical location due to the high latency of WAN. NCache provides Bridge Topology to handle all of this.

Read more about it at WAN replication.

Feature Area	Couchbase Server	NCache
Active - Passive	Supported. Support uni-directional cluster replication	Full support. Bridge Topology Active-Passive You can create a Bridge between the active and passive sites. The active site submits all updates to the Bridge which then replicates them to the passive site.

Feature Area	Couchbase Server	NCache
Active – Active	Supported. Setting up bidirectional replication requires setting up two uni-directional replication links from one cluster to another.	Full support. Bridge Topology Active-Active You can create a Bridge between two active sites. Both submit their updates to the Bridge which handles conflicts on last update wins rule or a custom conflict resolution handler provided by you. Then, the Bridge ensures that both sites have the same update.
One Active – Multiple Passive	Supported.	No support (but coming soon).
3 or More Active	Supported.	No support (but coming soon).
Conflict Resolution	Partial support (with limitation). Document with most updates "wins" the conflict. This is the only conflict resolution option available. No support for a "custom conflict resolution handler" provided.	Full support. By default, "last update wins" algorithm is used to resolve conflicts. But, you can specify a "custom conflict resolution handler" that is called to resolve conflict by comparison content of both objects.
De-duplication	Supported.	Full support. NCache Bridge optimizes replication queue by de-duplicating items. If the same key is updated multiple times, it only replicates the most recent update.
Data Security	Supported. Uses SSL/TLS for securing the connection.	Full support. Uses VPN between data centers for security. Additionally, can also encrypt data with 3DES and AES algorithms before transportation.

2.5 Cache Administration

Cache administration is a very important aspect of any distributed cache. A good cache should provide the following:

1. GUI based and command line tools for cache administration including cache creation and editing/updates.
2. GUI based tools for monitor the cache activities at runtime.
3. Cache statistics based on PerfMon (since for Windows PerfMon is the standard)

NCache provides powerful support in all these areas.

Read more about it at cache administration and monitoring.

Feature Area	Couchbase Server	NCache
Cache Admin (GUI Tool)	Partial support. Limited features in GUI tool. Mostly through command line tools.	Full support (advanced). NCache Manager is a powerful GUI tool for NCache. It gives you an explorer view and lets you quickly administer the cache including cache creation/editing and many other functions.
Cache Monitoring (GUI Tool)	Supported.	Full support (advanced). NCache Monitor is a powerful GUI tool for NCache. Its lets you monitor NCache cluster wide activity from a single location. It also lets you monitor all of NCache clients from a single location. And, you can incorporate non-NCache PerfMon counters in it for comparison with NCache stats. This real-time comparison is often very important.
PerfMon Counters	No support. Only proprietary counters shown inside the web based monitoring tool. Cannot mix counters with other third-party app counters for correlation analysis.	Full support. NCache provides a rich set of PerfMon counters that can be seen from NCache Manager, NCache Monitor, or any third party tool that supports PerfMon monitoring.
JMX	No support.	Full support. NCache provides a rich set of JMX counters for its Java clients that can be seen from any third party tool that supports JMX monitoring.
Command Line Admin Tools	Supported.	Full support. NCache provides a rich set of command line tools/utilities. You can create a cache, add remote clients to it, add server nodes to it, start/stop the cache, and much more.
Admin and Monitoring API	Partial support. REST API. No .NET or Java events provided for cluster changes at runtime.	Full support. NCache provides .NET and Java API to manage and monitor the caches & client. Using this API you can stop/start the cache, get the statistics of the connected clients or get the health info of the cache cluster.

Feature Area	Couchbase Server	NCache
		This API also includes event notification for cluster changes.

2.6 Security & Encryption

Many applications deal with sensitive data or are mission critical and cannot allow the cache to be open to everybody. Therefore, a good distributed cache provides restricted access based on authentication and authorization to classify people in different groups of users. And, it should also allow data to be encrypted inside the client application process before it travels to the distributed cache.

NCache provides strong support in all of these areas. See security and encryption features for details.

Feature Area	Couchbase Server	NCache
Active Directory/LDAP Authentication	No support. Only text-file based simple authentication.	Full support. You can authenticate users against Active Directory or LDAP. If security is enabled, nobody can use the cache without authentication and authorization.
Authorization	Supported.	Full support. You can authorize users to as either "users" or "admins". Users can only access the cache for read-write operations while "admins" can administer the cache.
Data Encryption	No support.	Full support (3DES, 256AES, ...) You can enable encryption and NCache automatically encrypts all items inside the client process before sending them to the cache. And, this data is kept encrypted while in the cache. And decryption also happens automatically and transparently inside the client process. Currently, 3DES and 256AES encryptions are provided and more are being added.
Secure Communication	Supported. SSL/TLS supported.	Full support. Through VPN. SSL/TLS is usually needed for end-user applications whereas NCache security is between app servers and the caching tier or

Feature Area	Couchbase Server	NCache
		between multiple data centers. And, here VPN is the most suitable way to secure the transport. NCache also encrypts data (as mentioned above) that adds to this in case you don't want to use VPN.

2.7 Object Caching Features

These are the most basic operations without which a distributed cache becomes almost unusable. These by no means cover all the operations a good distributed cache should have.

Feature Area	Couchbase Server	NCache
Get, Add, Insert, Remove, Exists, Clear Cache	Supported.	Full support. NCache provides more variations of these operations and therefore more control to the user.
Expirations	Partial support. Only absolute expiration provided. No sliding expiration is available.	Full support. Absolute and Sliding expirations They are both provided by NCache. Absolute expiration is good for data that is coming from the database and must be expired after a known time because it might become stale. Sliding expiration means expire after a period of inactivity and is good for session and other temporary data that must be removed once it is no longer needed.
Lock & Unlock	Supported.	Full support. NCache provides both of these. Lock is used to exclusively lock a cached item so nobody else can read or write it. This item stays locked until either the lock expires or it is unlocked. NCache also has incorporated "lock/unlock" features in "get" and "insert" calls. "`GetAndLock()`" returns an item locked and "`InsertAndUnlock()`" updates an item and also unlocks it in one call. This speeds up the cache operations.

Feature Area	Couchbase Server	NCache
Item Versioning	Supported. It is called Check and Set (CAS).	Full support. This feature allows NCache clients to ensure that only one client can update an item and all future updates will fail unless they first fetch the latest version and then update.
Multiple Object Versions	Supported. Uses JSON for handling differences of attributes in different object versions.	Full support. NCache allows two different versions of the same object/class to be stored in the cache by different apps. Each app retrieves its own version and the cache keeps a superset.
Streaming API	Partial support. Provides `Append()` but not `GetChunk()`	Full support. For large objects, NCache allows the cache clients to fetch them in "`GetChunk()`" manner and update them in "`AppendChunk()`" manner. With this, NCache clients can stream in or out large objects from the cache.

2.8 Managing Data Relationships

Since most of the data being cached comes from relational databases, it has relationships among various data items. So, a good cache should allow you to specify these relationships in the cache and then keep the data integrity. It should allow you to handle one-to-one, one-to-many, and many-to-many data relationships in the cache automatically without burdening your application with this task.

See more at managing data relationships.

Feature Area	Couchbase Server	NCache
Key Based Dependency	No support.	Full support. NCache provides full support for it. You can specify one cached item A depends on another cached item B which then depends on a third cached item C. Then, if C is ever updated or removed, B is automatically removed from the cache and that triggers the removal of A from the cache as well. And, all of this is done automatically by the cache. With this feature, you can keep track of one-to-one, one-to-many, and many-to-many relationships in the cache and invalidate cached

Feature Area	Couchbase Server	NCache
		items if their related items are updated or removed.
Multi-Cache Key Dependency	No support.	Full support. This is an extension of Key Based Dependency except it allows you to create this dependency across multiple caches.

2.9 Synchronization with Data Sources

Database synchronization is a very important feature for any good distributed cache. Since most data being cached is coming from a relational database, there are always situations where other applications or users might change the data and cause the cached data to become stale.

To handle these situations, a good distributed cache should allow you to specify dependencies between cached items and data in the database. Then, whenever that data in the database changes, the cache becomes aware of it and either invalidates its data or reloads a new copy.

Additionally, a good distributed cache should allow you to synchronize the cache with non-relational data sources since real life is full of those situations as well.

NCache provides a very powerful database synchronization feature. Read more about it at database synchronization.

Feature Area	Couchbase Server	NCache
SqlDependency (SQL Server)	No support.	Full support. NCache provides SqlDependency support for SQL Server. You can associate a cached item with a SQL statement based dataset in SQL Server. Then whenever that dataset changes (addition, updates, or removal), SQL Server sends a .NET event to NCache and NCache invalidates this cached item. This feature allows you to synchronize the cache with SQL Server database. If you have a situation where some applications or users are directly updating data in the database, you can enable this feature to ensure that the cache stays fresh.
OracleDependency (Oracle)	No support.	Full support. NCache provides OracleDependency support for Oracle. It works just like SqlDependency but for Oracle. Whenever data changes in the database, Oracle notifies NCache through

Feature Area	Couchbase Server	NCache
		Oracle event notification. Just like SqlDependency, this feature allows you to synchronize the cache with Oracle database.
DbDependency (OLEDB)	No support.	Full support. NCache provides support for you to synchronize the cache with any OLEDB database. This synchronization is based on polling and although it is not as real-time as event based synchronization. But it is much more efficient because in one poll, NCache can synchronize thousands of cached items instead of receiving thousands of individual events in SqlDependency.
File Based Dependency	No support.	Full support. NCache allows you to specify a dependency on an external file. Then NCache monitors this file for any updates and when that happens, NCache invalidates the corresponding cached item. This allows you to keep the cached item synchronized with a non-relational data source.
Custom Dependency	No support.	Full support. NCache allows you to implement a custom dependency and register your code with the cache cluster. Then, NCache calls your code to monitor some custom data source for any changes. When changes happen, you fire a dependency update within NCache which causes the corresponding cached item to be removed from the cache. This feature is good when you need to synchronize the cached item with a non-relational data source that cannot be captured by a flat file. So, custom dependency handles this case.

2.10 Runtime Data Sharing

Runtime data sharing has become an important use for distributed caches. More and more applications today need to share data with other applications at runtime in an asynchronous fashion.

Previously, relational databases were used to share data among multiple applications but that requires constant polling by the applications wanting to consume data. Then, message queues became popular because of their asynchronous features and their persistence of events. And although message queues are great, they lack performance and scalability requirements of today's applications.

As a result, more and more applications are using in-memory distributed caches for event driven runtime data sharing. This data sharing should be between multiple .NET applications or between .NET and Java applications.

NCache provides very powerful features to facilitate runtime data sharing. They are discussed below and compared with Couchbase. See runtime data sharing for details.

Feature Area	**Couchbase Server**	**NCache**
Item Level Events (onInsert/onRemove)	No support.	Full support (fast). These events are super-fast and scalable due to NCache's light-weight socket-level protocol.
Cache Level Events (Add/Insert/Remove)	No support.	Full support. NCache allows you to register callbacks against cache level add, insert, and update events. Your callback is called when this happens even if your application is remotely connected to the cache.
Custom Events (Fired by Apps)	No support.	Full support. NCache allows your applications to fire custom events into the cache cluster. And, other applications can register to be notified for these events. This feature allows you to coordinate a producer/consumer scenario where after the producer has produced data, it notifies all the consumers to consume it.
Continuous Query	No support.	Full support. NCache provides a powerful Continuous Query (CQ) feature. CQ lets you specify a SQL-like query against which NCache monitors the cache for any additions, updates, or deletes. And, your application is notified whenever this happens. Think of this feature as being equivalent to SqlDependency but for the cache and not the

Feature Area	Couchbase Server	NCache
		database.

2.11 Cache Search (SQL-Like)

Distributed cache is frequently used to cache objects that contain data coming from a relational database. This data may be individual objects or collections that are the result of some database query.

Either way, applications often want to fetch a subset of this data and if they have the ability to search the distributed cache with a SQL-like query language and specify object attributes as part of the criteria, it makes the distributed cache much more useful for them.

NCache provides powerful Object Query Language (OQL) for searching the cache with a SQL-like query. Read more about it at Object Query Language for details.

Feature Area	Couchbase Server	NCache
Object Query Language (OQL)	Supported. Provides N1QL.	Full support. NCache provides a rich Object Query Language (OQL) with which you can search the cache. Your search criteria can now include object attributes (e.g. `cust.city = 'New York'`) and you can also include Tags and Named Tags in the query language. The net benefit is that the cache is no longer a black box that is only accessible through key-value pair.
OQL on Tags, Named Tags, and Groups	No support.	Full support.
LINQ Queries	No support. Open source third party provider available without any official support.	Full support. NCache allows you to search the cache with LINQ queries. LINQ is a popular object querying language in .NET and NCache has implemented a LINQ provider. So, if you're comfortable using LINQ, you can search the cache the same way you would with NCache's own OQL.

2.12 Data Grouping

A distributed cache should be much more than a Hashtable with a (key, value) pair interface. It needs to meet the needs of real life applications that expect to fetch and update data in groups and collections. In a relational database, SQL provides a very powerful way to do all of this.

We've already explained how to search a distributed cache through OQL and LINQ. Now let's discuss Groups, Tags, and Named Tags. These features allow you to keep track of collections of data easily and even modify them.

Feature Area	**Couchbase Server**	**NCache**
Groups/Subgroups	No support.	Full support. NCache provides the ability for you to group cached items in a group-subgroup combination (or just group with no subgroup). You can later fetch or remove all items belonging to a group. You can also fetch just the keys and then only fetch subset of them.
Tags	No support.	Full support. NCache provides a concept called Tags. A Tag is a string that you can assign to one or more cached items. And one cached item can be assigned multiple Tags. Tags are specified at cache level. And, later, you can fetch items belonging to one or more Tags in order to manipulate them. You can also include Tags in Object Query Language or LINQ search as part of the criteria.
Named Tags	No support.	Full support. NCache provides Named Tags feature where you can assign a "key" and "tag" to one or more cached items. And, a single cached item can get multiple Named Tags. Later, you can fetch items belonging to one or more Named Tags. You can also use Named Tags in OQL and LINQ queries as part of the criteria.

2.13 Read-through, Write-through & Cache Loader

Many people use distributed cache as "cache on the side" where they fetch data directly from the database and put it in the cache. Another approach is "cache through" where your application just asks the cache for the data. And, if the data isn't there, the distributed cache gets it from your data source.

The same thing goes for write-through. Write-behind is nothing more than a write-through where the cache is updated immediately and the control returned to the client application. And, then the database or data source is updated asynchronously so the application doesn't have to wait for it.

NCache provides powerful capabilities in this area. See read-through & write-through for details.

Feature Area	Couchbase Server	NCache
Read-through	No support.	Full support. NCache allows you to implement multiple read-through handlers and register with the cache as "named providers". Then, the client can tell NCache to use a specific read-through upon a "cache miss". NCache also allows you to add read-through handlers at runtime without stopping the cache.
Write-through	No support.	Full support. NCache allows you to implement multiple write-through handlers and register with NCache as "named providers". Then, the client can tell NCache which write-through to use when updating the data source. You can also add write-through handlers at runtime without stopping the cache.
Write-behind	No support.	Full support. Write-behind is the same as write-through except writing to the data source is asynchronously done. NCache updates the cache immediately and queues up the database update and a background thread processes it and calls your write-through handler.
Reload Items at Expiration & Database Synchronization	No support.	Full support. If you've implemented a read-through handler, NCache allows you to use it to specify that whenever a cached item expires. Then, instead of removing it from the cache, NCache should call your read-through handler to read a new copy of that object and update the cache with it. You can specify the same when database

Feature Area	Couchbase Server	NCache
		synchronization is enabled and a row in the database is updated and a corresponding cached item would have been removed from the cache but is now reloaded with the help of your read-through.
Cache Startup Loader	Partial support (limited). Provides a "server warmup" feature to load frequently used data from persistent store. User has no control over the data source from where to load the data.	Full support. NCache lets you implement a Cache Loader and register it with the cache cluster. NCache then calls it to prepopulate the cache upon startup. CacheLoader is your code that reads data from your datasource/database.

2.14 Cache Size Management (Evictions Policies)

An in-memory distributed cache always has less storage space than a relational database. So, by design, a distributed cache is supposed to cache a subset of the data which is really the "moving window" of a data set that the applications are currently interested in.

This means that a distributed cache should allow you to specify how much memory it should consume and once it reaches that size, the cache should evict some of the cached items. However, please keep in mind that if you're caching something that does not exist in the database (e.g. ASP.NET Sessions) then you need to do proper capacity planning to ensure that these cached items (sessions in this case) are never evicted from the cache. Instead, they should be "expired" at appropriate time based on their usage.

Feature Area	Couchbase Server	NCache
Max Cache Size (in MBs)	Supported.	Supported.
LRU Evictions (Least Recently Used)	Partial support. Only available for in-memory store.	Full support.
LFU Evictions (Least Frequently Used)	Partial support. Not available for in-memory store. Only available with persistent store and called Not-Frequently-Used (NRU).	Full support.
Priority Evictions	No support.	Full support. NCache also lets you specify a "do not evict" priority for some cached items and then they are not evicted.

Feature Area	Couchbase Server	NCache
Do Not Evict Option	No support.	Full support. NCache lets you specify "do not evict" option for the entire cache. Then, nothing is evicted even when cache is full. Instead, the client applications receive an error stating that the cache is full when they try to add data to the cache.

2.15 ASP.NET & Java Web Sessions Caching

ASP.NET applications need three things from a good distributed cache. And, they are ASP.NET Session State storage, ASP.NET View State caching, and ASP.NET Output Cache.

ASP.NET Session State store must allow session replication in order to ensure that no session is lost even if a cache server goes down. And, it must be fast and scalable so it is a better option than InProc, StateServer, and SqlServer options that Microsoft provides out of the box. NCache has implemented a powerful ASP.NET Session State provider.
See ASP.NET Session State for details.

ASP.NET View State caching allows you to cache heavy View State on the web server so it is not sent as "hidden field" to the user browser for a round-trip. Instead, only a "key" is sent. This makes the payload much lighter, speeds up ASP.NET response time, and also reduces bandwidth pressure and cost for you. NCache provides a feature-rich View State cache.
See ASP.NET View State for details.

Third is ASP.NET Output Cache. For .NET 4.0, Microsoft has changed the ASP.NET Output Cache architecture and now allows third-party distributed caches to be plug-in. ASP.NET Output Cache saves the output of an ASP.NET page so the page doesn't have to execute next time. And, you can either cache the entire page or portions of the page. NCache has implemented a provider for ASP.NET Output Cache.

Feature Area	Couchbase Server	NCache
ASP.NET Sessions	No official support. An Open Source provider available without any official support.	Full support (advanced). NCache has implemented an ASP.NET Session State Provider (SSP) for .NET 2.0+. You can use it without any code changes. Just change web.config. NCache also provides flexible session locking options to handle robots flooding an ASP.NET website. NCache provides intelligent session replication and is much faster than any database storage for sessions.
ASP.NET Sessions (Multi-Site)	No support.	Full support. NCache allows you to share sessions across multiple data centers.

Feature Area	Couchbase Server	NCache
		This serves situations where you don't want to replicate all sessions to each data center but want the ability to overflow traffic from one data center to another without losing your ASP.NET sessions. The session moves from one data center to the next as the user moves.
ASP.NET View State Cache	No support.	Full support (advanced). Yes. NCache has an ASP.NET View State caching module. Use it without any code changes. Just modify config file. Here are some advanced features supported by NCache: - Group-level policy - Associate pages to groups - Link View State to sessions - Max View State count per user - More…
ASP.NET Output Cache	No support. Open Source provider available without any official support.	Supported. NCache has an ASP.NET Output Cache provider implemented. It allows you to cache ASP.NET page output in a distributed cache and share it in a web farm.
Java Session Persistence	No support. Open Source provider available without any official support.	Full support. NCache has implemented a Java Servlet Session Provider (Java Servlet 2.3+). You can use it without any code changes. Just change web.xml NCache provides intelligent session replication and is much faster than any database storage for sessions.

2.16 Third Party Integrations

Memcached is an open-source in-memory distributed caching solution which helps speed up web applications by taking pressure off the database. Memcached is used by many of the internet's biggest websites and has been merged with other technologies.
NCache implements Memcached protocol to enable users with existing Memcached implementations to easily migrate to NCache. No code change required for this.
See Memcached Wrapper for details.

NHibernate is a very powerful and popular object-relational mapping engine. And, fortunately, it also has a second level cache provider architecture that allows you to plug-in a third-party cache without making any code changes to the NHibernate application. NCache has implemented this NHibernate second level cache provider.
See NHibernate second level cache for details.

Similarly, Entity Framework from Microsoft is also a very popular object-relational mapping engine. And, although Entity Framework doesn't have nice second level cache provider architecture like NHibernate, NCache has nonetheless implemented a second level cache for Entity Framework.
See Entity Framework second level cache for details.

Feature Area	**Couchbase Server**	**NCache**
NHibernate 2nd Level Cache	No official support. Open Source provider available without any official support.	Full support. NCache provides a NHibernate 2nd Level Cache provider that you can plug-in through web.config or app.config changes. NCache has also implemented database synchronization feature in this so you can specify which classes should be synchronized with the database. NCache lets you specify SqlDependency or DbDependency for this.
Entity Framework 2nd Level Cache	No support.	Full support. Custom ADO.NET Provider NCache has implemented a behind-the-scene second level cache for Entity Framework. You can plug-in NCache to your EF application, run it in analysis mode, and quickly see all the queries being used by it. Then, you can decide which queries should be cached and which ones skipped. You can also specify which queries should be synchronized with the database through SqlDependency.
Memcached Protocol Server	Supported.	Full support. NCache has implemented Memcached protocol fully. This means you can plug-in NCache as a distributed cache as a replacement of Memcached. Two ways are offered to use Memcached applications with NCache. Memcached Pug-In: All the popular Open Source .NET Memcached client libraries have been implemented for NCache. Memcached Gateway: Using this you can store

Feature Area	Couchbase Server	NCache
		your application data from any application that use the Memcached.
Memcached Smart Wrapper	No support.	Full support. NCache has implemented the popular .NET and Java Memcached client libraries which in-turn calls NCache. This allows you to plug-in Memcached client library to your application without any code change or recompilation.
Hibernate 2nd Level Cache	No official support. Open Source provider available without any official support.	Full support. NCache provides Hibernate 2nd Level Cache provider that you can plug-in to your Java app without any code changes. NCache has also implemented database synchronization feature in this so you can specify which classes should be synchronized with the database. NCache lets you specify OracleDependency or DbDependency for this.
Spring Integration	No support. An open source integration is provided but without any official support.	Full support.
JCache API	No support.	No support (but coming soon).

3 Conclusion

As you can see in a very detailed fashion, we have outlined all of NCache features and all the corresponding Couchbase features or a lack thereof. We hope this document helps you get a better understanding of Couchbase versus NCache.

Please note that the true cost of ownership for a distributed cache is not just the price of it. It is the cost to your business. The most important thing for many customers is that they cannot afford unscheduled downtime (especially during peak hours). And, this is where an elastic cache like NCache truly shines.

Additionally, all those caching features that NCache provides are intended to give you total control over the cache and allow you to cache all types of data and not just simple data.

Please read more about NCache and also feel free to download a fully working 60-day trial of NCache from:

- NCache details
- Download