

**Using NCache
for
ASP.NET Sessions in Web Farms**

June 7, 2010

Table of Content

1	Getting Started	1
1.1	Step 1: Install NCache.....	1
1.2	Step 2: Configure for Multiple Network Cards	1
1.3	Step 3: Configure Firewall TCP Port for NCache Clients	1
1.3.1	Modify client.nconf.....	2
1.3.2	Modify Alachisoft.NCache.Service.exe.config	2
1.4	Step 4: Define a Clustered Cache.....	2
1.5	Step 5: Test the Cluster.....	3
1.6	Step 6: Copy Client.nconf to each client machine	3
1.7	Step 7: Modify web.config for your ASP.NET application.....	4
1.7.1	Web.config changes for Session State Provider (SSP)	4
1.7.2	Web.config changes for HttpModule.....	4
1.8	Step 8: Ensure All Objects are Serializable	5
1.9	Step 9: Run Application and watch sessions being created in NCache	5
2	NCache Session Module Architecture	6
2.1	Control Flow for an Http Request	6
2.2	Control Flow for an Http Response	6
3	Planning Cache Deployment	7
3.1	Caching Topology Recommendation	7
3.2	Memory Usage Requirements	8
3.3	CPU Usage Requirements	8
3.4	Do not use InProc Mode	8
3.5	Auto Start Cache on Reboot	8
4	Multiple Geographical Locations	9
4.1	Step 1: Create a separate cache for each location	9
4.2	Step 2: Modify web.config for each geographical location	9
4.2.1	Web.config changes for Session State Provides (SSP)	10
4.2.2	Web.config changes for NCache section	10
5	Web.config properties for NCache	11
6	Special Considerations for Sessions HttpModule.....	12
6.1	Session.Redirect(url, false)	12
6.2	Session.Abandon()	12
6.3	Using Session in your own custom HttpModules	12
6.4	If you're using Session.Clear()	13
6.5	clearASPSession property	13
7	Troubleshooting: Common Problems Encountered.....	13
7.1	No server available to process request.....	14
7.2	Out of memory	14
7.3	Sessions are being evicted	14
7.4	Items added are not showing up on all cache servers	14

1 Getting Started

1.1 Step 1: Install NCache

You must install NCache on all your web server machines. If you decide to create a separate caching tier, then you must install NCache on all the cache server machines as well. Installing NCache is a simple process. There is a Windows Installer (.msi) file that you need to run.

When you download NCache, an evaluation key is emailed to the email address you provided. Please note that installation requires you to specify the evaluation key (and not the purchased license key). Once NCache is installed, you can use "Activate NCache" to activate your purchased license key.

NOTE: You must install Enterprise Edition in order to use Sessions. Sessions module is not available as part of the Developer Edition.

1.2 Step 2: Configure for Multiple Network Cards

If your cache server (where you plan to host the cache) contains multiple network cards, then you must bind NCache to one of the network cards. If you don't do this, NCache server will automatically pick the first network interface it finds which may not be the one to which your DNS is mapping the hostname or the one you want to use.

In order for you to do that, you'll need to update Alachisoft.NCache.service.exe.config file in NCache\bin\service folder. The line you want to modify in this file is as following:

```
<appSettings>
  ...
  <add key="NCache.BindToIP" value="192.168.1.60" />
  ...
</appSettings>
```

You can either manually update this file or use NCache Manager to update it. Either way, once the file is updated, you must restart NCache service on all the cache servers.

Please note that you must use the same ip-address when creating your clustered cache to which you have bound your cache server. If you want to use a host-name at the time of defining your cache, then please make sure that the host-name resolves to the same ip-address as what you have bound to.

NOTE: Whenever you update Alachisoft.NCache.Service.exe.config, you must do it on all cache servers and then you must restart NCache service afterwards. Otherwise, your changes will not take effect.

1.3 Step 3: Configure Firewall TCP Port for NCache Clients

If you have a separate caching tier and there is a firewall between your client machines (meaning your web servers) and your cache servers, then you must open the following TCP port between them. However, if your clients and cache servers are on the same LAN, then you won't need to do anything.

By default, this port is 9800. If you want to change it, you will need to modify two files:

1.3.1 *Modify client.nconf*

Change the "port" below to specify a different port based on your environment preferences.

```
<configuration>
  <ncache-server port="9800" timeout="90" connection-retries="1"
    retry-interval="0" connection-timeout="5"/>
  ...
</configuration>
```

1.3.2 *Modify Alachisoft.NCache.Service.exe.config*

However, you'll then have to specify the same port in all your cache servers as well:

```
<appSettings>
  ...
  <add key="CacheServer.Port" value="9800" />
  ...
</appSettings>
```

The port value in both files (in client.nconf and in Alachisoft.NCache.Service.exe.config) must match.

NOTE: Whenever you update Alachisoft.NCache.Service.exe.config, you must do it on all cache servers and then you must restart NCache service afterwards. Otherwise, your changes will not take effect.

1.4 Step 4: Define a Clustered Cache

Once NCache has been installed and you've specified all environment related settings, you're now ready to define a clustered cache. However, before you do this, the first thing you have to address is which caching topology to use. You can use Replicated or Partition-Replica topology for ASP.NET session (Partitioned Cache is not suitable because it does not have replication) depending on a number of things. They are each discussed below:

1. Use a Replicated Cache if you have 2 to 3 cache servers
2. Use Partition-Replica Cache if you have 3 or more cache servers

Please note that all cache servers must be either 32-bit or 64-bit but not a mixture of the two. You can mix 32-bit clients with 64-bit cache but the cache servers must all be of the same type.

The second thing you have to decide is whether to host your cache on your web servers itself or whether to have dedicated cache servers. In principle, it is always better to have dedicated cache servers. However, for smaller web farms, it is also okay to keep the cache on your web servers as long as you have sufficient memory and CPU power to run both simultaneously.

In order to create a clustered cache, you must use NCache Manager. Follow these steps to create a cache (in this example, a 2-node replicated cache is created):

1. Run NCache Manager

2. Create a New Project or open an Existing Project
3. Select "Create New Clustered Cache..."
4. Specify a cache name
5. Select cluster type to be "Replicated"
6. Select two servers to add to this cluster. If you're unable to view "Network Neighborhood", you can always type the hostnames or their ip-addresses one by one.
7. Select a Cluster Port. This is a TCP port that the cache servers use to form a cluster. If all cache servers are not on the same LAN, then this port must be open between them.
8. Do not select "Enable overflow capability". You can read about it on online help.
9. Select ".NET Managed Heap" as Storage Type. Then, specify "Maximum size" for this cache. This should be less than total RAM available. If you have enabled evictions, then NCache evicts items when it reaches this size. For Sessions, you should never want to evict active sessions. So, this Maximum Size should be enough to let you store all your sessions without being full.
10. Leave defaults for eviction policy.
11. Leave defaults for Read-Through and Write-Through. Press Finish.

Once this is done, then click on the cluster node in the tree view. This will show properties on the right side. Go to the "Policies" tab and select "Turn off eviction". This ensures that your sessions are never evicted. Instead, when the cache is full (meaning it reaches the Maximum Size), it stops accepting any new sessions.

1.5 Step 5: Test the Cluster

Now that you've created a clustered cache, you need to test it to make sure it is working properly before you use it from within your ASP.NET application. Follow these steps in NCache Manager:

1. Right-click on the clustered cache in tree view and select "Start". This will start cache on all servers you had added to the clustered cache earlier. You'll see the server nodes changing their colors to indicate "running".
2. Right-click on the clustered cache in tree view and select "Statistics". This will display statistics on the right side. Statistics are displayed in Performance Monitor control of Windows.
3. Press "Add" button on the right side to add a few items to the cache. You should notice the "Count" increase on all the server nodes in the statistics window. In case of Replicated Cache, the count will be the same. In case of Partition-Replica, the Count will not be the same but the total of Count on all partitions will be equal to the total cache count.

1.6 Step 6: Copy Client.nconf to each client machine

Let's assume that the clustered cache you've created is called "demoClusteredCache" and it includes two cache server, 192.168.1.60 and 192.168.1.61. Then, the client.nconf on each client machines (meaning your web server machine) should contain the following:

```
<cache id ="democlusteredcache" node-balance="true">  
  <server name="192.168.1.60" priority="1"/>  
  <server name="192.168.1.61" priority="2"/>  
</cache>
```

Please make sure that "node-balance" is "true". This will ensure that all your clients (at connection time) will evenly go to all the servers. The only time you don't want this is if you have geographically separated cache servers and you want certain clients to go to one server and others to the second one.

1.7 Step 7: Modify web.config for your ASP.NET application

The final step before running your ASP.NET application and storing your sessions in NCache is to modify your web.config. For .NET 1.1, NCache provides an HttpModule and for .NET 2.0 or later, NCache provides a Session State Provider (SSP) module to handle sessions. These modules automatically store your sessions in a distributed cache. Below are web.config changes for each path.

1.7.1 Web.config changes for Session State Provider (SSP)

```
<assemblies>
  <add assembly="Alachisoft.NCache.SessionStoreProvider,
    Version=X.X.X.X, Culture=neutral,
    PublicKeyToken=CFF5926ED6A53769" />
</assemblies>

<sessionState cookieless="false" regenerateExpiredSessionId="true"
  mode="Custom" customProvider="NCacheSessionProvider" timeout="20">
  <providers>
    <add name="NCacheSessionProvider"
      type="Alachisoft.NCache.Web.SessionState.NSessionStoreProvider"
      sessionAppId="NCacheTest"
      cacheName="demoClusteredCache"
      writeExceptionsToEventLog="false"
      enableLogs="false" />
  </providers>
</sessionState>
```

Please note the **Version=X.X.X.X** in "assemblies" tag. This version must match the version of NCache you have downloaded. You can check the NCache version by viewing HELP->ABOUT dialog in NCache Manager or run "c:\program files\ncache\bin\tools\listcaches.exe" as it also displays version.

If you decided to set `enableLogs="true"` then you'll have to make sure that "NCache\log-files\SessionStoreProvider" folder gives write permissions to "users" or to your specific ASP.NET user-id.

Similarly, if you decided to set `writeExceptionsToEventLog="true"` then you'll have to make sure that your ASP.NET user-id has the permission to write to the Event Log.

Please note that `timeout="20"` means that your sessions will expire after 20 minutes of inactivity. You can specify whatever value that suits you here.

1.7.2 Web.config changes for HttpModule

```
<appSettings>
  <add key="cacheName" value="demoClusteredCache" />
</appSettings>
```

```

    <add key="exceptionsEnabled" value="true"/>
    <add key="enableLogs" value="false"/>
    <add key="sessionId" value="NCacheTest"/>
</appSettings>

<httpModules>
    <add name="NCacheWebSessionState"
        type="Alachisoft.NCache.Web.SessionState.NSessionStateModule,
            Alachisoft.NCache.SessionState, Version=X.X.X.X, Culture=neutral,
            PublicKeyToken=cff5926ed6a53769"/>
</httpModules>

<sessionState mode="InProc"
    stateConnectionString="tcpip=127.0.0.1:42424"
    sqlConnectionString="data source=127.0.0.1;Trusted_Connection=yes"
    cookieless="false"
    timeout="20"/>

```

Please note the **Version=X.X.X.X** This version must match the version of NCache you have downloaded. You can check the NCache version by viewing HELP->ABOUT dialog in NCache Manager.

If you decided to set `enableLogs="true"` then you'll have to make sure that "NCache\log-files\HttpModule" folder gives write permissions to "users" or to your specific ASP.NET user-id.

Please note that you specify a "sessionState" with an "InProc" mode. And, please also note that `timeout="20"` means that your sessions will expire after 20 minutes of inactivity. You can specify whatever value that suits you here.

1.8 Step 8: Ensure All Objects are Serializable

If you were previously using the InProc mode of ASP.NET Sessions, then the objects you were putting your Session did not need to be Serializable. However, if you were using StateServer or SqlServer modes for Sessions previously, then your objects are already serializable for all of this to work.

In either case, before using NCache for your sessions, you must ensure that all your custom objects that you're putting in Session are serializable. It can be a very simple tag that you need to put on all your class definitions as shown below (in C#):

```

[Serializable]
public class Product
{
    ...
}

```

1.9 Step 9: Run Application and watch sessions being created in NCache

Now, you're ready to run your ASP.NET application and watch its sessions being created in NCache. You can view the "Statistics" (as described above) and see the "Count" variable increase. Please note that each session is stored as one item in the cache. So, the Count indicates how many sessions are stored in the cache. Some of the additional counters that you would find interesting are:

- **Count:** Shows how many sessions in the cache

- **Fetches/sec:** Shows you how many sessions are being read by your application from the cache. Remember, each Http Request results in one "Fetch" and one "Add" or "Update" call to the cache. Make sure you select your cache-id for this counter.
- **Additions/sec:** Shows you how many new sessions are being created per second. Make sure you select your cache-id for this counter.
- **Updates/sec:** Shows you how many existing sessions are being updated per second. Make sure you select your cache-id for this counter.
- **Expirations/sec:** This shows you how many sessions are being expired per second. Make sure you select your cache-id for this counter.
- **Requests/sec:** This shows how many requests of all types one cache server is handling per second. Make sure you select "ncache server" for this and not your cache-id.

You can view the above counters either from the "Statistics" window of NCache Manager or from Performance Monitoring tool in Control Panel. You can also monitor this remotely and log all the results to a file for later review.

2 NCache Session Module Architecture

NCache has implemented a module on top of its core distributed cache to handle sessions. There are two types of implementations. For .NET 1.1, an HttpModule is implemented and for .NET 2.0, a Session State Provider (SSP) is implemented. Below is the control flow for a session module. This is pretty much the same for HttpModule and SSP.

2.1 Control Flow for an Http Request

Here is how a typical flow happens for an SSP implementation:

1. User sends an Http Get or Post from browser to IIS
2. IIS invokes ASP.NET framework to process this request
3. ASP.NET framework generates a SessionId if one is not provided by the user request.
4. ASP.NET framework calls NCache SSP to load session from the distributed cache based on the SessionId. NCache uses the SessionId as the key to lookup session in the cache. If you have specified an AppId in web.config, NCache uses this in combination with SessionId as the cache-key.
5. If session is found in the cache, NCache loads it and populates the regular ASP.NET Session object with the data fetched from the cache. If no session is found, then it is assumed that a new ASP.NET session had been created for the user which is left empty.
6. ASP.NET Framework executes the desired ASP.NET page (.aspx) with the Session object either populated with data from the cache or empty if it is a new session.

2.2 Control Flow for an Http Response

Here is the control flow once your ASP.NET page (.aspx) has been invoked.

1. ASP.NET page uses the Session object by getting or putting data in it. All these get and put operations do not involve any NCache calls and are being performed within your ASP.NET worker process.
2. When ASP.NET page execution is done and a response is about to be sent back to the user, ASP.NET Framework again calls the NCache SSP and asks it to save this session to the cache.
3. If the ASP.NET page has called `Response.Redirect(url)`, the response is terminated and the session data is not saved in the cache because ASP.NET framework does not call NCache Session State Provider. Therefore, if you want your session to be saved to the cache, you must use `Response.Redirect(url, false)` which does not terminate the response.
4. When NCache SSP is called to save the session, NCache takes the SessionId and uses it as a key to update an existing entry in the cache. If no entry exists, NCache adds one to the cache. If the user has specified an AppId in web.config, the AppId is used in combination with SessionId as the cache-key.
5. Once the session is saved in the cache, the response is sent back to the user.
6. The user can issue another Http Request thus repeating the entire process explained here. The only difference is that the next Http Request may go to another web server which will be able to load the same session from the distributed cache based on the SessionId.

3 Planning Cache Deployment

You must plan correctly for deploying your ASP.NET application with NCache in order to ensure it would continue to work smoothly. This involves a number of important things including choosing the right caching topology, having enough memory, and having enough CPU power. Each is explained below.

3.1 Caching Topology Recommendation

There are three possible deployment scenarios based on how small or big your web farm is. Each of them is mentioned below.

1. **2 Server Web Farm:**
 - a. Replicated Cache on your two web servers
2. **3-10 Server Web Farm:**
 - a. 2-node Replicated Cache but on separate dedicated servers
 - b. Ideally, use 64-bit machines as cache servers.
3. **11+ Server Web Farm:**
 - a. Partition-Replica Cache on a separate caching tier.
 - b. Have a 3:1 to 5:1 ratio between web servers and cache servers.
 - c. Ideally, use 64-bit machines as cache servers.

Some of the things you need to consider when planning for your cache cluster are as following:

3.2 Memory Usage Requirements

You need to estimate your peak load in terms of number of sessions per 20 minute period and multiply that by the average session size to come up with your memory requirements. NCache puts around 15% overhead on top of this in terms of memory consumption.

The bottom line is that you never want to run out of memory. And, since NCache stores your sessions in memory, you must have adequate memory available always. On a 32-bit system, the maximum memory you can have is 4GB. And, out of that 2GB is taken by the OS. However, in Windows 2003 Enterprise Edition, you can configure the OS to only use 1GB and leave 3GB for the applications.

In a 64-bit system, you should plan on leave 2GB for the OS and around 0.5GB as extra unused memory. You never want to reach the boundary limit of available memory because it results in a lot of paging activity by your virtual memory subsystem. And, paging has a negative impact on the performance of NCache.

3.3 CPU Usage Requirements

Ideally, you should have a dual-CPU or higher configuration for your cache servers. NCache is a multi-threaded application and makes full use of multiple cores to do parallel processing. Additionally, if you're thinking about keeping the cache on your web servers, please calculate how much CPU is your ASP.NET application is consuming. NCache consumes around 15-30% CPU normally so if you're application is already consuming 70% CPU, then you should not keep the cache on your web servers.

3.4 Do not use InProc Mode

InProc is not recommended to be used with ASP.NET applications. You should run your cache in OutProc mode so the cache itself lives in a separate stable process and then either run local clients or remote clients.

The reason for not recommending InProc for ASP.NET is because ASP.NET application runs in worker processes. And, these worker processes are frequently recycled. Now, an InProc mode means that your application process is actually considered one of the cache instances in a cluster. So, when your worker process recycles, it results in one cache node leaving the cluster and a new node joining the cluster. And, for larger caches, this can cause a lot of overhead especially in a high transaction environment.

InProc is more suitable for server applications that always run in a single process (e.g. Windows Service).

3.5 Auto Start Cache on Reboot

By default, if your cache server reboots or if you restart the NCache service, it does not automatically rejoin the cache cluster that it was part of before rebooting. This is an intended behavior and the reason is because we don't know what caused the node to reboot and whether it is safe to rejoin a clustered cache running in production.

However, we give you the ability to specify that you want to automatically rejoin one or more clustered caches upon reboot. It is specified in `Alachisoft.NCache.Service.exec.config` file like this:

```
<appSettings>
  ...
  <add key="autoStartCache" value="demoClusteredCache,myreplicatedcache" />
  ...
</appSettings>
```

Please note that multiple cache names can be specified separated by commas. Please also note that if you have enabled security (Please read online help for this), then you'll also have to specify "cacheUser" and "cachePassword" along with "autoStartCache" tag. You can find out example of this in Alachisoft.NCache.Service.exe.config file.

NOTE: Whenever you update Alachisoft.NCache.Service.exe.config, you must do it on all cache servers and then you must restart NCache service afterwards. Otherwise, your changes will not take effect.

4 Multiple Geographical Locations

If your web application is hosted in multiple geographical locations but you want to treat all these locations as one logical web farm and share sessions across these sites, you can do that with NCache. NCache provides special handling for multiple geographical locations in such a way that your sessions are accessed from any location.

However, please note that NCache makes an assumption that most of your users who start from one location stay in that location and only a small percentage (perhaps 10-20%) are sent to a different location once their session is created in one location. NCache stores session on a user's original location and then if the user is sent to a different location, that web farm obtains session from the original location. So, if too many users are sent back and forth, the WAN latency between these multiple locations will have a negative performance impact on your application's response time.

Follow these steps to setup a multi-site web farm.

4.1 Step 1: Create a separate cache for each location

For each geographical location, create a separate clustered cache. So, if you have three locations (e.g. London, New York, and Tokyo), you'll create three clustered caches one for each location. The topology of each clustered cache is based on how many total web servers to you have in each location (as explained above).

Please follow all the details on how to create a clustered cache from "Getting Started" section of this document. For our example, let's assume that you have three geographical locations (London, New York, and Tokyo). So, you have created three replicated caches, one for each location and they are called "londonReplicatedCache", "newyorkReplicatedCache", and "londonReplicatedCache".

4.2 Step 2: Modify web.config for each geographical location

Following are the web.config changes that are different from regular web.config changes already described earlier in the "Getting Started" section. Please note that you need to also specify `<assemblies>` tag as specified in "Getting Started" section. Since that does not change here, that is why it is not separately described here.

4.2.1 Web.config changes for Session State Provides (SSP)

Please note that in "Getting Started" section, you're already told how to specify SSP in your web.config. However, for a multi-site configuration, the SSP is specified differently and is described below. Add the following for your SSP in web.config:

```
<sessionState cookieless="false"
    regenerateExpiredSessionId="true"
    mode="Custom"
    customProvider="NCacheSessionProvider"
    timeout="60"
    sessionIDManagerType=
    "Alachisoft.NCache.Web.SessionStateManagement.CustomSessionIdManager,
    Alachisoft.NCache.SessionStateManagement">
  <providers>
    <add name="NCacheSessionProvider"
        type="Alachisoft.NCache.Web.SessionState.NSessionStoreProvider"
        sessionAppId="1234"
        cacheName="myReplicatedCache"
        writeExceptionsToEventLog="false"
        asyncSession="false"
        enableLogs="false"/>
  </providers>
</sessionState>
```

Please note that you are specifying `regenerateExpiredSessionId="..."` and `sessionIDManagerType="..."` values. This allows you to use the custom session-id manager from NCache. NCache modifies your session-id to put a location prefix to it. This location prefix is used in subsequent requests to determine from which cache to obtain the session for this user. This optimizes the performance by reducing unnecessary trips to secondary caches that may be across WAN.

4.2.2 Web.config changes for NCache section

Add the following to your web.config.

```
<configSections>
  <section name="ncache"
    type="Alachisoft.NCache.Web.SessionStateManagement.NCacheSection,
    Alachisoft.NCache.SessionStateManagement,
    Version=X.X.X.X,
    Culture=neutral,
    PublicKeyToken=CFF5926ED6A53769"/>
</configSections>

<ncache>
  <sessionLocation>
    <primaryCache id="londonReplicatedCache" sid-prefix="lndr"/>
    <secondaryCache id="newyorkReplicatedCache" sid-prefix="nykr"/>
    <secondaryCache id="tokyoReplicatedCache" sid-prefix="tkyr"/>
  </sessionLocation>
</ncache>
```

Please note the **Version=X.X.X.X** This version must match the version of NCache you have downloaded. You can check the NCache version by viewing HELP->ABOUT dialog in NCache Manager.

Please note that by specifying the above, you've created a custom section in your web.config that is read by NCache. The "section" portion specifies the NCache assembly that processes this information and the <nocache> tag specifies all the multi-site information.

Below is an explanation of the <sessionLocation> tags:

1. **primaryCache:** This specifies the cache that is local to this geographical location. Each geographical location will have its own primary cache and will specify the other locations as secondary caches.
 - a. **id:** This is the cache name
 - b. **sid-prefix:** this is a prefix that is added at the beginning of your SessionId by [Alachisoft.NCache.Web.SessionStateManagement.CustomSessionIdManager](#) module. This helps NCache determine where to find the session in subsequent requests and avoid expensive trips to secondary caches across the WAN.
2. **secondaryCache:** Use this tag to specify one or more secondary caches. Each secondary geographical location is specified separately with its own tag and its own sid-prefix.

With the information specified above (of course the primary and secondary caches being different for each geographical location), your application will be able to fetch sessions from the correct location. Below is a typical control flow of a user who originated in one location and then was routed to another location.

1. User accesses the web application for the first time in London and his session is created in London.
2. His SessionId gets a prefix of "Indr" by [Alachisoft.NCache.Web.SessionStateManagement.CustomSessionIdManager](#) module. From this point onward, his SessionId contains this prefix.
3. He continues to use the web app. Whenever, he is sent to the London web farm, the web farm immediately recognizes the "Indr" prefix to indicate that this user's session is stored in the primaryCache for London.
4. Whenever, this user is sent to either New York or Tokyo web farm, they immediately recognize that this user's session should be in the "[londonReplicatedCache](#)" secondary cache. So, they directly look in that cache to fetch the session.
5. The net effect is that the user is able to use the application from all three locations and bounce back and forth without experiencing any problems.

5 Web.config properties for NCache

When you use NCache for storing sessions, you end up specifying certain properties in web.config. They are all explained here.

1. **cacheName:** This is name of the cache you've created.
2. **enableLogs ("true" or "false"):** This turns on or off error logging. If it is turned on, then NCache logs all errors in NCache\log-files\SessionStoreProvider or NCache\log-files\HttpModule depending on whether you're using SSP or HttpModule.

3. **sessionAppId**: If you have multiple applications or app domains running on the same web farm and accessible from the same application, then you have the choice of either sharing your sessions across app domains or not. If you don't want to share sessions across app domains, then specify a unique "sessionAppId" for each app domain. This will ensure that each app domain puts its own sessionAppId to the SessionId thus making it impossible for the other app domains that might be using the same SessionId to fetch the same session.
4. **writeExceptionsToEventLog ("true" or "false")**: If this is set to true, then NCache logs errors to the Event log. However, please make sure that your ASP.NET user-id has permission to write to Event Log. Otherwise, you'll get errors when you run your application.
5. **exceptionsEnabled ("true" or "false")**: If this is set to false, then NCache ignore exceptions thrown by NCache server. If this is set to true, then NCache throws exceptions upward.

6 Special Considerations for Sessions HttpModule

If your application is in .NET 1.1 and you're using NCache HttpModule to handle sessions, you must be aware of the following special considerations.

6.1 Session.Redirect(url, false)

You must specify "false" as the second argument in all your Session.Redirect calls. Otherwise, the new page will not see any changes you made to Session in the current page. The reason is that if you don't specify "false", ASP.NET terminates the response and does not invoke NSessionStateModule (NCache's HttpModule) for saving Session into the cache. And, when the new page retrieves Session from the cache, it gets the older copy which does not have your changes.

6.2 Session.Abandon()

If you make Session.Abandon() call, it does not delete the session from the cache in .NET 1.1. So, the preferred way of abandoning the session is by calling NSessionStateModule's Abandon call. However, you'll have to first get a handle to this HttpModule from HttpApplication object. Here is the sample code:

```
Using Alachisoft.NCache.Web.SessionState;  
  
//you may write the following code under logout button  
NSessionStateModule sm = Application["NSessionStateModule"] as NSessionStateModule;  
sm.Abandon(Session);
```

The reason you have to do this is that if you simply call Session.Abandon(), there is no way ASP.NET provides for NSessionStateModule to find this out. Therefore, it cannot take the appropriate action.

Please note that this is not a problem in .NET 2.0 or later versions where you're able to use SSP.

6.3 Using Session in your own custom HttpModules

NCache provides you an `HttpModule` (`NSessionStateModule`) to handle Session storage in the cache. And, this `HttpModule` is called at `AcquireRequestState` and `ReleaseRequestState` events. However, if you've also developed an `HttpModule` and your event handlers are called before `NSessionStateModule`'s event handlers are called, then your code is not likely to see the correct data in the Session object.

If your `HttpModule` is called before `AcquireRequestState` event handler of `NSessionStateModule` is called, then you'll see an empty session. And, to handle this situation, you'll need to use the following code at the beginning of your `HttpModule`'s event handler:

```
Using Alachisoft.NCache.Web.SessionState;  
  
//you may write the following code under logout button  
NSessionStateModule sm = Application["NSessionStateModule"] as NSessionStateModule;  
sm.Load(HttpContext);
```

And, if your code is making changes to the Session object and wants to make sure these changes are seen by other ASP.NET pages, then you should call `NSessionStateModule.Update(HttpContext)` in the following way:

```
Using Alachisoft.NCache.Web.SessionState;  
  
//you may write the following code under logout button  
NSessionStateModule sm = Application["NSessionStateModule"] as NSessionStateModule;  
sm.Update(HttpContext);
```

This will ensure that everything is working properly.

6.4 If you're using `Session.Clear()`

Please note that if you're using `Session.Clear()`, NCache will remove this Session from the cache. Upon trying to save the Session content, NCache checks to see if Session is empty (`Session.Count == 0`). If that is the case, then it calls a `cache.Remove(...)`.

6.5 `clearASPSession` property

If your website contains multiple App Domains and you allow user to go from one App to another either through `Response.Redirect(url, false)` or by providing a link on one of the pages, then you must use this property. You can use it in `web.config` as following:

```
<appSettings>  
  <add key="clearASPSession" value="true"/>  
</appSettings>
```

This property calls `Session.clear()` before trying to load the Session from the cache. This way, even if the session is not in the cache anymore (e.g. your user just did a logout but all App Domains still hold the old Session object) then your Session object will reflect the correct state. Otherwise, it might show old data in some cases.

7 Troubleshooting: Common Problems Encountered

Here are some common problems that you might encounter. Please note that all of this is also discussed in the online help and also in NCache forum on our website's support page.

7.1 No server available to process request

If your client throws the following exception, it can be due to one of the following reasons.

1. You didn't put a correct cache server name in your client.nconf
2. The cache server name does not match the ip-address to which you have bound NCache server (see earlier sections). This can be due to old entries on your DNS server.
3. The communication between the client node and cache server is blocked by a firewall. See the previous section on how to configure this.

7.2 Out of memory

You didn't plan your memory usage correctly. Your application is putting more data in the cache than the available memory on the cache server. Please either increase memory or reduce the amount of data being stored in the cache server.

7.3 Sessions are being evicted

You have specified a "Maximum Size" of your cache to be too low. And, your application is putting more data than what the cache can handle (meaning the number of sessions times the average session size is more than Maximum Size). Additionally, you have not specified "Turn off eviction".

As a result, NCache is evicting active sessions from the cache in order to make room for new sessions being added.

7.4 Items added are not showing up on all cache servers

The most likely cause is that your cluster has not been formed correctly. You either have multiple network cards in your cache servers but you have not bound your cache servers to a specific one. Or, there is a firewall between the two cache servers. To detect whether there is a firewall between cache servers or not, type the following from one of the cache servers command prompt:

```
C:\> telnet otherCacheServer clusterPort
```

If you see that the other server has answered your call then port is not blocked.