14,839,412 members

Sign in

articles Q&A forums stuff lounge ?

Search for articles, questions,

MyCache: Distributed caching engine for an ASP.NET web farm - Part I: The demonstration

Al-Farooque Shubho

14 Dec 2010 CPOL

Rate me: 4.85/5 (39 votes)

Demonstration of a distributed caching engine for ASP.NET applications deployed under a load-balanced environment in a web farm, which is built on top of WCF and the Microsoft Caching Application Block.

Download source code - 689 KB

Introduction

It's the same old classic story:

Duke and his team started developing an ASP.NET web based application, and after months of hard work in analyzing, developing, and testing the system, the team was able to hit the production deployment date, and they had a nice small celebration party on the occasion of their success.

A few months passed away. The team remained busy implementing some feature enhancements and change requests. The client was extremely happy with the popularity of the site and rapid growth of user activity in a quick amount of time, until the day they shoot an email, with the subject line "Site performance sucks"!

What happened? The **big** question was asked in the team meeting, and nobody was able to answer instantly. Everybody in the team were well-aware that they had designed the data storage and retrieval routines giving "performance optimization" the highest priority, and they had also utilized the infamous ASP.NET Cache as much as possible. After all, the performance testing was tremendously successful on the last staging site, wasn't it?

Yes, it was. But later, thorough investigation revealed that caching was not working correctly at all at the production sites. Almost all URLs hit were resulting in expensive database queries even if caching was heavily utilized all over; a completely different thing was happening in the production sites, whereas at the staging site, caching was working quite fine. What happened?

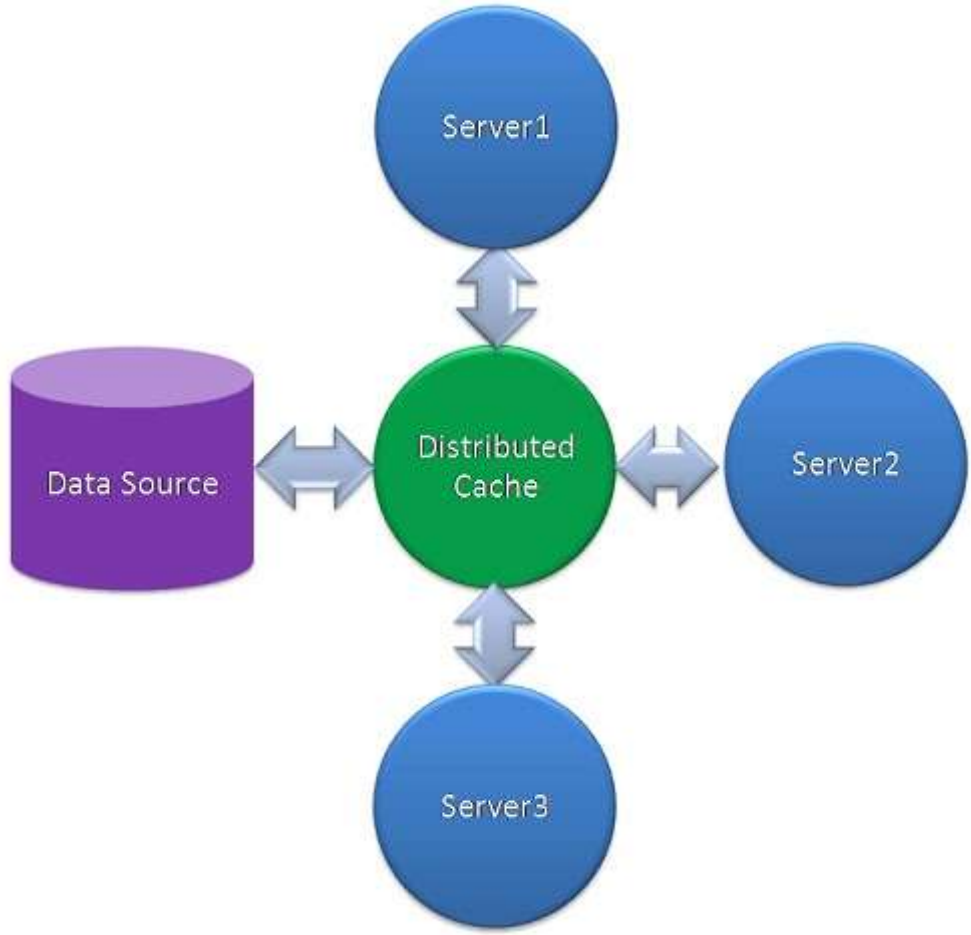
I can tell what happened. The production site had multiple load-balanced web servers configured under a web farm (whereas, at staging site, there was only one web server) pointing to the same code base, and not each request was being routed to the same specific server. So, even if the first hit caused a server to perform an expensive query and load the data in its cache, subsequent requests were likely to be redirected to a different server, which didn't have the data in their cache and hence they were performing the same expensive operations to load the data and put it into their in-memory cache, which is already available in another server's memory.

So, caching wasn't helping at all!

Duke and his team need a distributed cache

That's what all they need. They have multiple servers each having their own memory. So, even though caching has been implemented in the application, which was supposed to work on each server, the cached data was being loaded in multiple servers, multiple times (waste of memory and CPU), and the end result is poor performance.

A distributed caching engine solves this problem. It takes the responsibility of managing caching functionality, and the client web applications/sites can use it to store and retrieve cached data. Following is how the distributed caching engine works at its very basic level:



Distributed cache

- There is a cache server which is used by all servers in a web farm to store and retrieve cached data.
- While there is a request in a particular server in the load-balanced web farm, the server asks the cache server to get the data.
- Cache server serves the data from the cache if it is available, or lets the calling server know about the absence of the data. In that case, the application at the calling server loads data from the data storage/service, stores it inside the cache server, and processes the output to the client.
- A subsequent request arrives at a different server, which needs the same data. So, it asks the cache server to get the data.
- The cache server has that particular data in its cache now. So it serves the data.
- For any subsequent request to any server in the web farm requiring the same data, the data is served fast from the cache server, and the ultimate performance makes everybody happy.

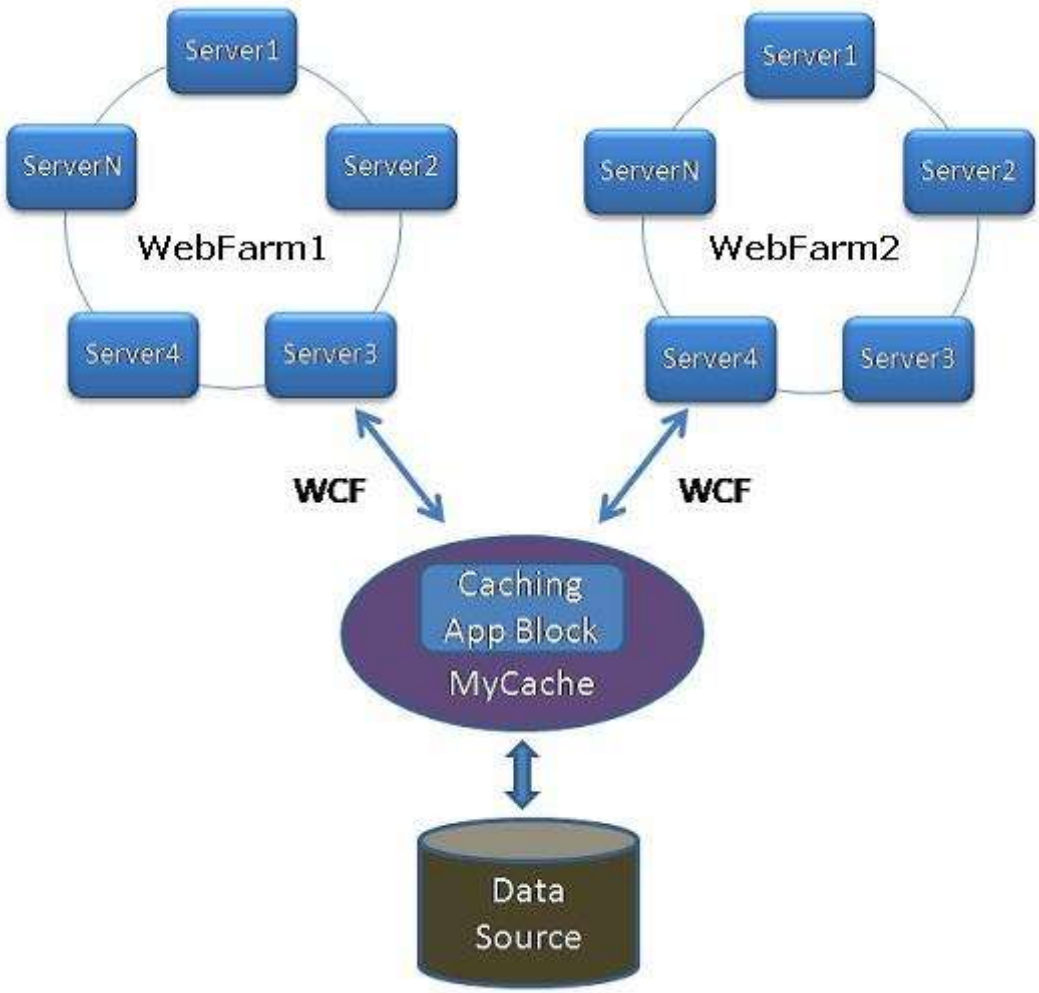
Which distributed caching engine to use?

For many reasons, NCache is the number one candidate to be considered as a distributed caching engine. But, it's not free (it has a free version, NCache Express, but, it has its own limitations). MemCache is another good option, but it's not purely a .NET implementation.

Recently, Microsoft has come up with its own caching engine, Velocity, which is purely .NET, and purely free (an obvious good option for caching).

But, for those who want to use a lightweight, customized, and simple home-made distributed caching engine, I do have one. This article is all about developing and using a distributed caching engine, which is built purely on the available basic technology which you are very familiar with, by utilizing the power of WCF and Microsoft Caching Application block. The following diagram depicts the basic architecture of the caching engine:

I named it MyCache:



MyCache architecture

The Microsoft Caching Application Block has been used as the core caching engine to manage "in-memory" cached data, which is used by a WCF Service application by exposing the basic caching services to the outside world. The external ASP.NET client applications communicate with the caching service application via WCF using a net.tcp binding.

The detailed implementation strategy will be covered later.

Is MyCache Production Ready?

Unfortunately, not yet. I have ended up developing the initial version of the system, and decided to demonstrate it to the great community CodeProject. Based upon its testing, performance, and feedback, I might need to improve it more to be able to declare it as "Production Ready".

But, MyCache works fine in my sample application, and hopefully should work in your system too. Give it a try, and let me know about any improvement suggestions or issues. I'll be glad to hear from you.

Available Services in MyCache

MyCache has roughly the same main feature set as the ASP.NET Cache. The following operations are currently available:

- Storing an object in cache with or without locking
- Getting an object from cache
- Checking whether a particular object is available in cache
- Removing an object from the cache
- Adding an object in the cache using:
    - FileDependency
    - Absolute expiration
    - Sliding expiration
    - `CacheItemRemovedCallBack`

The caching engine is exposed to ASP.NET client applications via a caching service, and ASP.NET applications interact with the caching service via *MyCache.dll*. The following methods are exposed by the caching service via WCF:

C# Copy Code

```
public interface ICacheService
{
    [OperationContract]
    void Add(string Key, byte[] Value);

    [OperationContract]
    void Insert(string Key, byte[] Value, string DependencyFilePath,
                string Priority, DateTime AbsoluteExpiration,
                TimeSpan SlidingExpiration);

    [OperationContract]
    void Remove(string Key);

    [OperationContract]
    byte[] Get(string Key);

    [OperationContract]
    bool Exists(string Key);

    [OperationContract]
    bool SetLock(string Key);

    [OperationContract]
    void ReleaseLock(string Key);
}
```

The caching service implements this interface, and MyCacheAPI is the client API to invoke the service methods over WCF.

Those who are used to programming with the ASP.NET Cache would find MyCache extremely familiar and easy to use. Following is a sample code-behind file taken from the sample application, which uses MyCacheAPI to utilize the features of MyCache to read some text data from a file and put it into MyCache for 5 minutes. Later, if the data is modified in the physical file, data is reloaded from the file and put into MyCache automatically. In a real application, you may want to retain data in MyCache for a longer period of time, and feel free to do that.

C# Shrink ▲ Copy Code

```
public partial class _Default : System.Web.UI.Page
{
    MyCache cache = new MyCache();

    protected void Page_Load(object sender, EventArgs e)
    {
```

```
        LoadData();
    }

    /// <summary>
    /// Load data from Cache, or read data and store in cache
    /// </summary>
    protected void LoadData()
    {
        lblWebFarm.Text = ConfigurationManager.AppSettings["WebFarmId"];
        string strData = cache.Get("Key") as string;
        if (strData == null)
        {
            divCacheMiss.Visible = true;

            string dependencyFilePath = GetDependencyFilePath();

            DateTime startTimeForCache = DateTime.Now;

            object Value = GetObjectFromFile(dependencyFilePath);

            DateTime endTimeForCache = DateTime.Now;
            cache.Add("Key", Value, dependencyFilePath,
                      Cache.NoAbsoluteExpiration, new TimeSpan(0, 5, 10),
                      CacheItemPriority.Normal,
                      new CacheItemRemovedCallback(onRemoveCallback));
            lblData2.Text = Value.ToString();
        }
        else
        {
            divCacheHit.Visible = true;
            DateTime startTimeForCache = DateTime.Now;
            StringBuilder sbOutput = new StringBuilder();

            string data = cache.Get("Key").ToString();

            DateTime endTimeForCache = DateTime.Now;

            lblCacheRetrievalTime.Text = string.Format("Retrieved following " +
               "data in <b>{0}</b> milliseconds",
               (endTimeForCache - startTimeForCache).TotalMilliseconds);
            lblData.Text = data;
        }

    }

    /// <summary>
    /// Get dependency file location
    /// </summary>
    /// <returns></returns>
    private string GetDependencyFilePath()
    {
        string WebRootPath = Path.GetDirectoryName(Server.MapPath(string.Empty));
        string dependencyFilePath =
          Path.Combine(Path.Combine(WebRootPath, "Data"), "Test.txt");
        return dependencyFilePath;        }

    /// <summary>
    /// Callback function to re-insert
    /// the item in cache if cache dependency changed
    /// </summary>
    /// <param name="Key"></param>
    /// <param name="Value"></param>
    /// <param name="reason"></param>
    protected void onRemoveCallback(string Key, object Value,
                   CacheItemRemovedReason reason)
```

```
    {
        if (cache == null)
        {
            cache = new MyCache();
        }
        if (reason == CacheItemRemovedReason.DependencyChanged)
        {
            //Aquire lock on MyCache service for the Key and proceed only if
            //no lock is currently set for this Key. This has been done to prevent
            //multiple load-balanced web application update the same data on MyCache service
            //sumultaneously when the underlying file content is modified
            if (cache.SetLock(Key))
            {
                string dependencyFilePath = GetDependencyFilePath();
                object modifiedValue = GetObjectFromFile(dependencyFilePath);
                cache.Add(Key, modifiedValue, dependencyFilePath,
                          Cache.NoAbsoluteExpiration, new TimeSpan(0, 5, 60),
                          CacheItemPriority.Normal,
                          new CacheItemRemovedCallback(onRemoveCallback));
                //Release lock when done
                cache.ReleaseLock(Key);
            }

        }
        if (reason == CacheItemRemovedReason.Expired ||
            reason == CacheItemRemovedReason.Removed)
        {
            if (cache.Exists(Key))
            {
                cache.Remove(Key);
            }
        }

    }

    /// <summary>
    /// Read data from File
    /// </summary>
    /// <param name="dependencyFilePath"></param>
    /// <returns></returns>
    private object GetObjectFromFile(string dependencyFilePath)
    {
        object updatedValue = File.ReadAllText(dependencyFilePath);
        return updatedValue;
    }
}
```

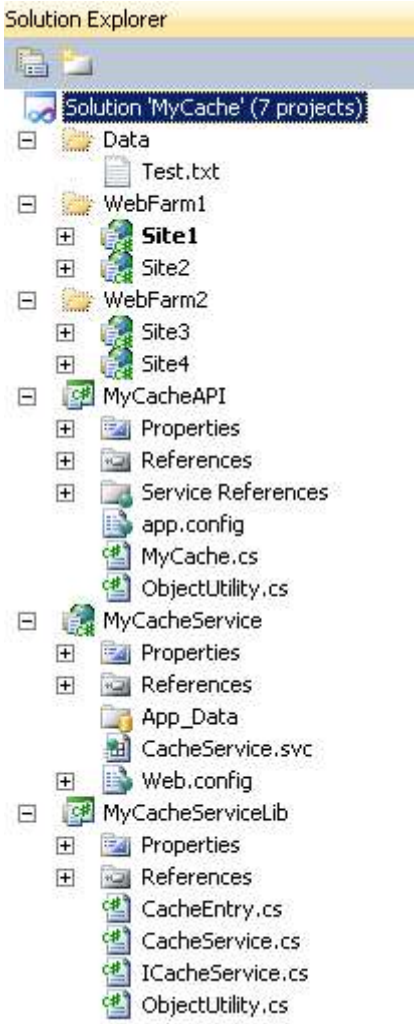
The above code is pretty straightforward and selfexplanatory. To be able to use MyCache in your application, all you need is to perform the following three steps:

1. Add a reference to *MyCache.dll*.
2. Set up the caching service (more on this later).
3. Configure your ASP.NET application as a client of the caching service (more on this later).

The following sections have detailed discussions on configuration issues, and a demonstration of a sample application which uses MyCache to store and retrieve data for its web applications deployed under a simulated web-farm environment.

Sample Application

I have developed a sample application for demonstrating the usage of MyCache and its core architecture and working principle. Download the sample application from the link (given at top of the article), extract it to a convenient location, and you will get the following view in Solution Explorer when opened in Visual Studio 2010:



Solution Explorer of sample application

Here is a description of the items in the Solution Explorer:

MyCacheService

The distributed caching service, which is used by the ASP.NET client applications. This is the WCF Service library which exposes the caching services via a WCF Service.

MyCacheServiceLib

The core distributed caching service engine. This is also a WCF Service library, which is used by MyCacheService to implement the actual services using Microsoft Caching Application Block as the core caching engine.

Mycache.dll

A simple class library which hides the complexity of the client ASP.NET applications/sites to communicate with the Caching Service to store and retrieve objects to and from the distributed cache engine. It also performs serialization/de-serialization of objects, and performs the `CachItemRemoveCallback` functionality by utilizing the same functionality available in the ASP.NET Cache.

WebFarm1 and WebFarm2

These are two ASP.NET folders, each containing two ASP.NET applications. WebFarm1 contains Site1 and Site2, and WebFarm2 contains Site3 and Site4. These two folders are being used to simulate two different web farms composed of load-balanced setting of two

different sites. In this particular case, we are assuming that Site1 and Site2 are not two different sites. Rather, they are actually two load-balanced deployments of the same site (they have the same WebFarmId = 1 in AppSettings). Similarly, we are assuming that Site3 and Site4 are two load-balanced deployments of the same site (they have the same WebFarmId = 2 in AppSettings). In reality, each deployment of the same site under the same load-balanced setup will point to the same code base, and hence will share the same WebFarmId (like what we tried to simulate here by sharing the same WebFarmId across two different sites within the same web farm).

All four sites use MyCacheService to store/retrieve data to/from MyCache, and interact with the caching service via *MyCacheAPI.dll*. MyCacheService lets any load-balanced group (web farm) of ASP.NET web sites/applications to utilize the caching service and share cached data among sites/applications within the web farm.

As said already, for demonstration purposes, Site1 and Site2 are assumed as the same web application deployed under a load-balanced web farm WebFarm1, and Site3 and Site4 are assumed as the same web application deployed under another load-balanced web farm WebFarm2. So, if Site1 stores data in MyCache, Site2 will get it, but Site3 and Site4 won't (because they belong to a different web farm, WebFarm2). For the same reason, if Site3 stores data in MyCache, Site4 will get it, but Site1 and Site2 won't (because they belong to a different web farm, WebFarm1).

Data/Test.txt

A text file which contains around 2 KB of data, and is used as a data source for all four ASP.NET applications within the sample application. These web applications (Site1, Site2, Site3, and Site4 read data from the file and puts into MyCache when data is not available in MyCache already).

Configuring the sample application

There are two main configuration issues with the Caching Service. These are:

Configuring the Caching Application block

You will most likely not need to do any configuration related change for the Caching Application Block, and this section is provided for informative purpose only.

The actual caching service uses the Microsoft Caching Application Block as the core engine to implement in-memory caching functionality, which is configured via the following steps:

- Adding references to the Caching and other required Application blocks:

Adding references to EnterpriseLibrary DLLs

- Configure the Caching Application Block to use in-memory caching:

XML Shrink ▲ Copy Code

```
<configSections>
    <section name="cachingConfiguration"
       type="Microsoft.Practices.EnterpriseLibrary.Caching.
             Configuration.CacheManagerSettings, Microsoft.Practices.
             EnterpriseLibrary.Caching, Version=4.0.0.0, Culture=neutral,
             PublicKeyToken=31bf3856ad364e35" />
</configSections>
<cachingConfiguration defaultCacheManager="Cache Manager">
    <cacheManagers>
     <add expirationPollFrequencyInSeconds="60"
       maximumElementsInCacheBeforeScavenging="10000"
       numberToRemoveWhenScavenging="10"
       backingStoreName="Null Storage"
       type="Microsoft.Practices.EnterpriseLibrary.Caching.CacheManager,
             Microsoft.Practices.EnterpriseLibrary.Caching,
             Version=4.0.0.0, Culture=neutral,
             PublicKeyToken=31bf3856ad364e35"
       name="Cache Manager" />
    </cacheManagers>
    <backingStores>
     <add encryptionProviderName=""
       type="Microsoft.Practices.EnterpriseLibrary.Caching.
             BackingStoreImplementations.NullBackingStore,
             Microsoft.Practices.EnterpriseLibrary.Caching,
             Version=4.0.0.0, Culture=neutral,
             PublicKeyToken=31bf3856ad364e35"
       name="Null Storage" />
    </backingStores>
 </cachingConfiguration>
```

Configuring the WCF Service

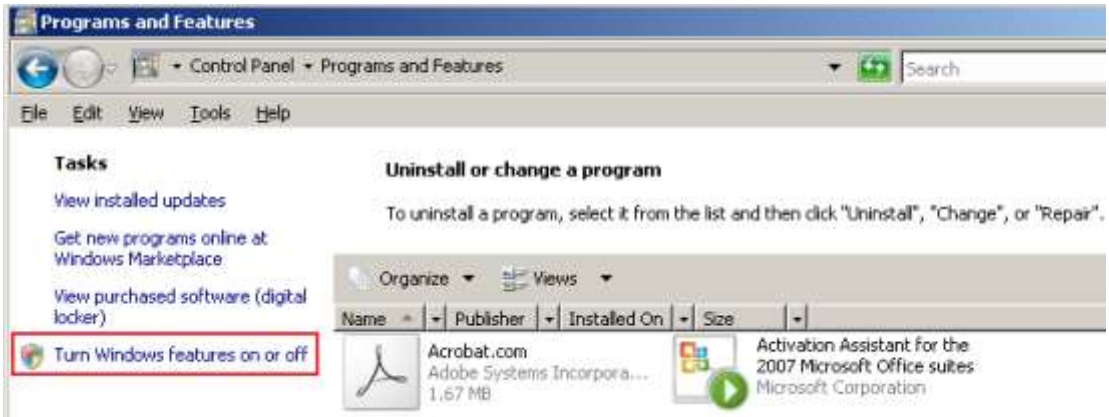
This is the main configuration that is needed to deploy the MyCache WCF Service and enable the client ASP.NET applications to use the caching service. The following configuration steps and screenshots are done in a Windows Vista PC running .NET Framework 4.0 and IIS 7, but it should not be very much different on Windows 7 or Windows 2008 machines. The configurations are to be done using the following steps:

Enabling WAS in OS to use net.tcp binding in WCF

WCF allows the service to be consumed via different protocols. The MyCache WCF Service exposes its services using net.tcp binding via IIS, and hence this has to be enabled and configured in the OS (because, by default, only HTTP binding and protocol is enabled for an IIS site).

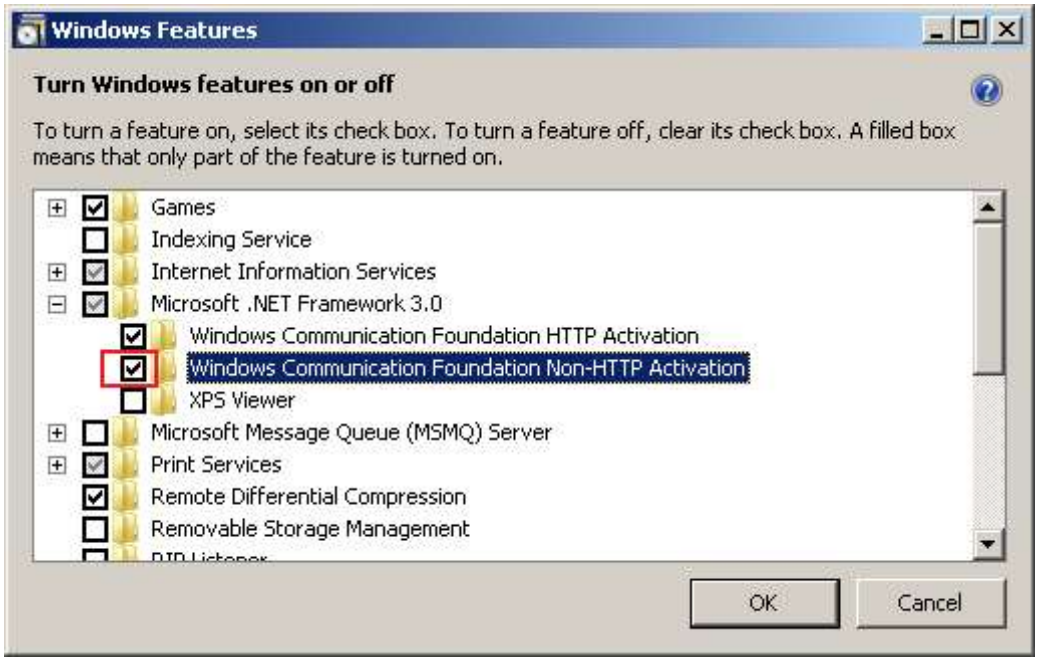
So, in order to host a WCF Service in IIS under net.tcp binding, WAS (Windows Activation Service) has to be configured, and here is how to do it:

- Go to "Start menu--> Control Panel", and switch to "Classic View" if not switched already.
- Click on "Programs and features", and then click on "Turn Windows features on or off" in the left pane (in Windows 7, you will get "Programs and features" directly after clicking the Control Panel).



Programs and features

- Expand the Microsoft .NET Framework 3.0 node (or Microsoft .NET Framework 3.1), check the "Windows Communication Foundation Non-HTTP Activation" check box, and press OK.



Enabling non-HTTP activation

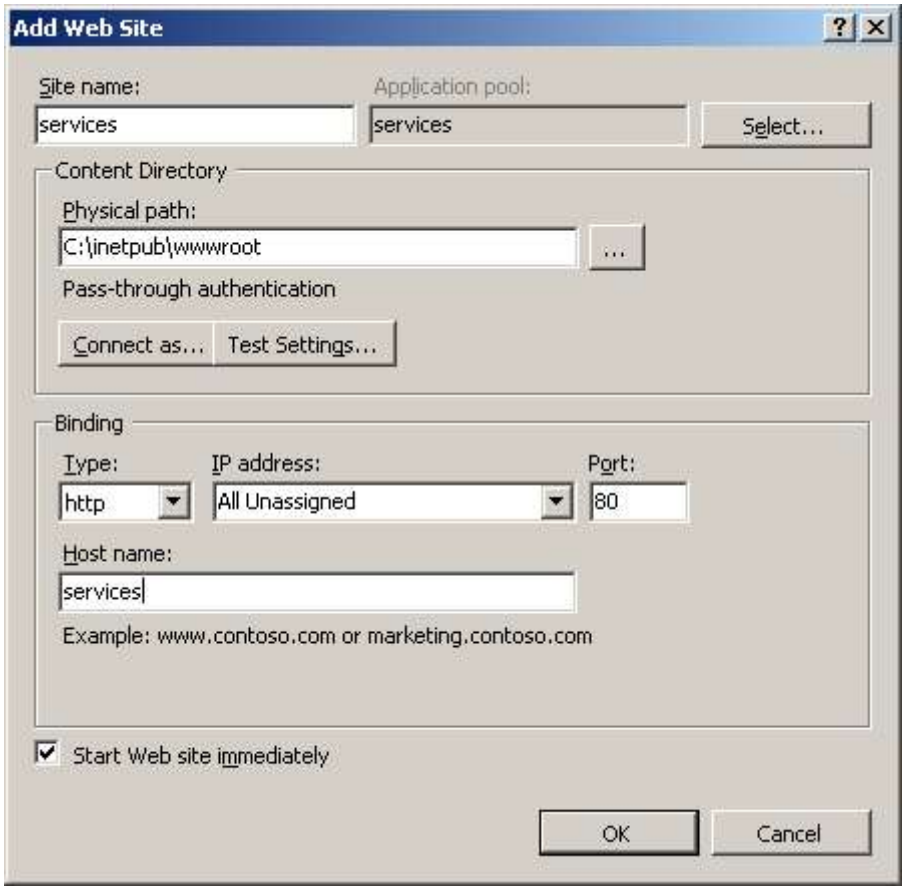
- The system might take quite some time to configure itself, and might or might not restart before finishing the new configuration, to take effect.

Configuring service hosting in IIS

Follow these steps to host the WCF Service in IIS using net.tcp binding:

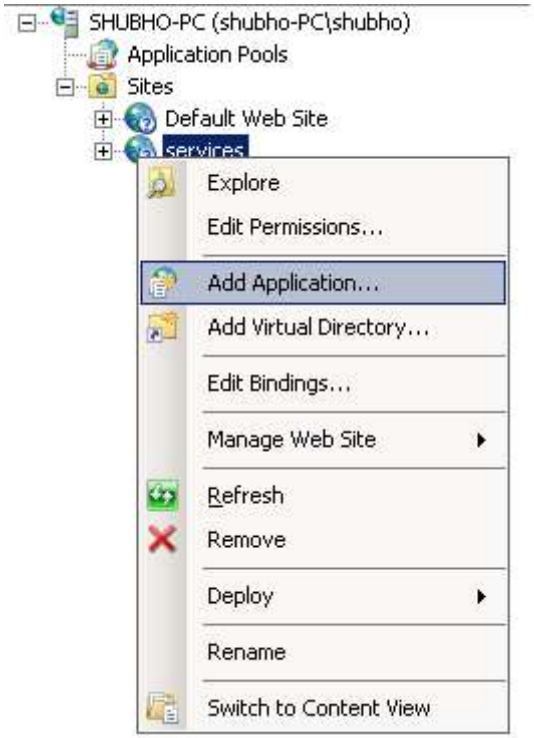
- Create a new site in IIS, say "services". You can name it whatever you like, but it would be better to name the site "services" as the sample application is already configured to use this site as the caching service provider. Make sure that the application pool of this site runs under .NET version 4.0.

Add new site in IIS

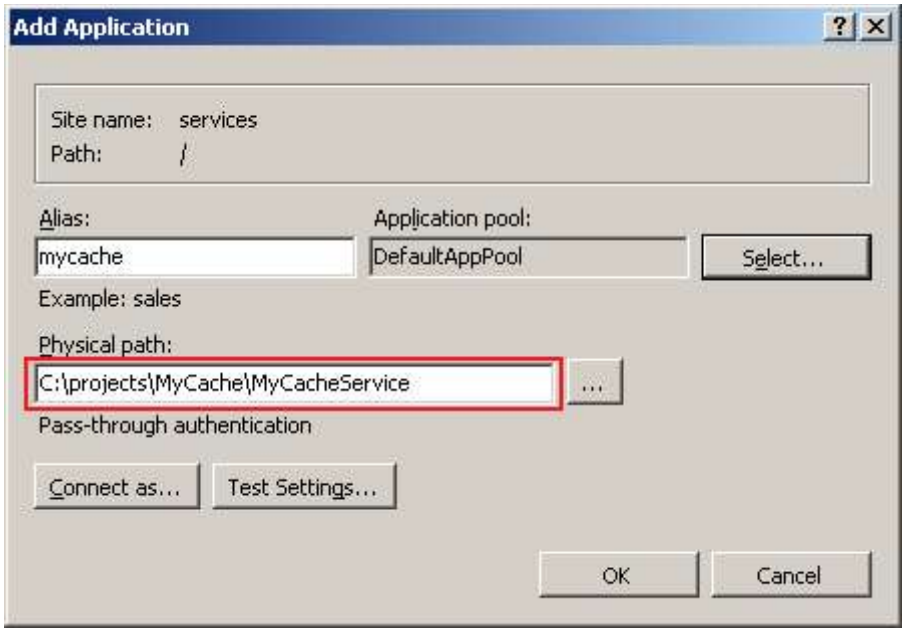


Add new site form

- Create a new application under the newly created site ("services") and name it "mycache" (again, you can name it whatever you like, but for running the sample application, name it "mycache", which you can change later if you like).

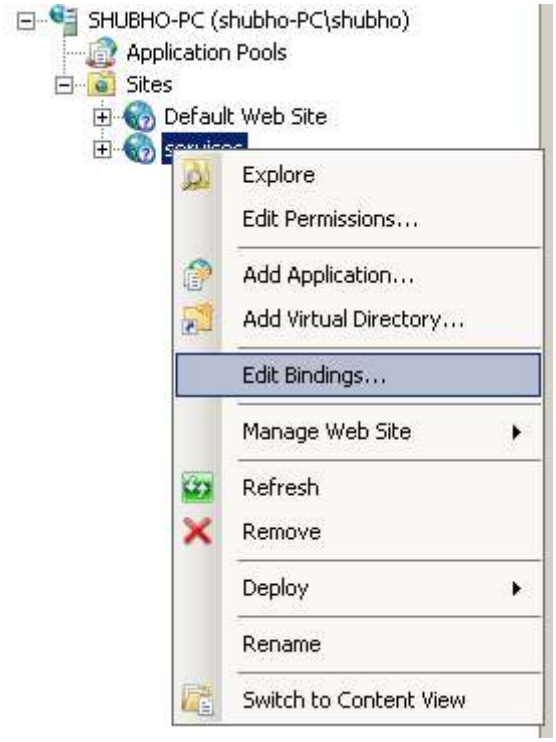


Add new application



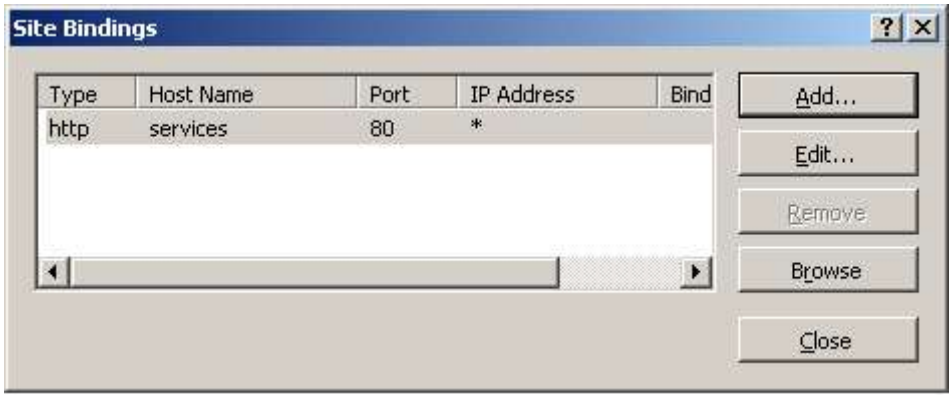
Set application physical path

- Right click on the newly created site "services" and click on "Bindings" to specify the net.tcp binding for the site:



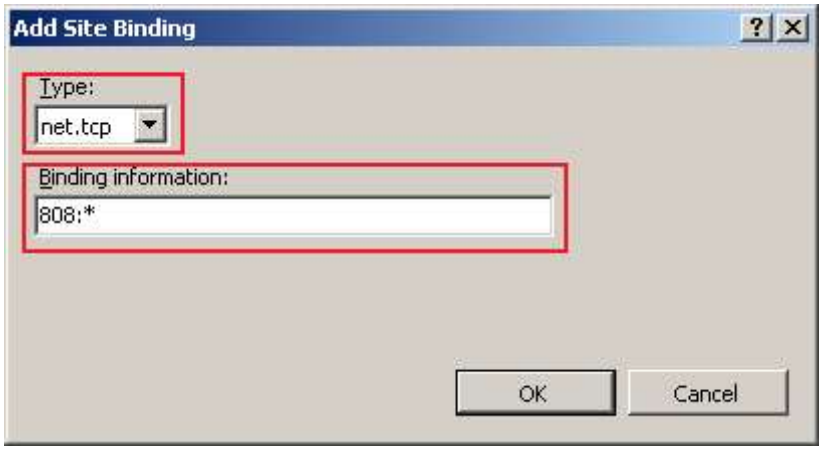
Edit bindings

The following configuration box "Site Bindings" would appear:



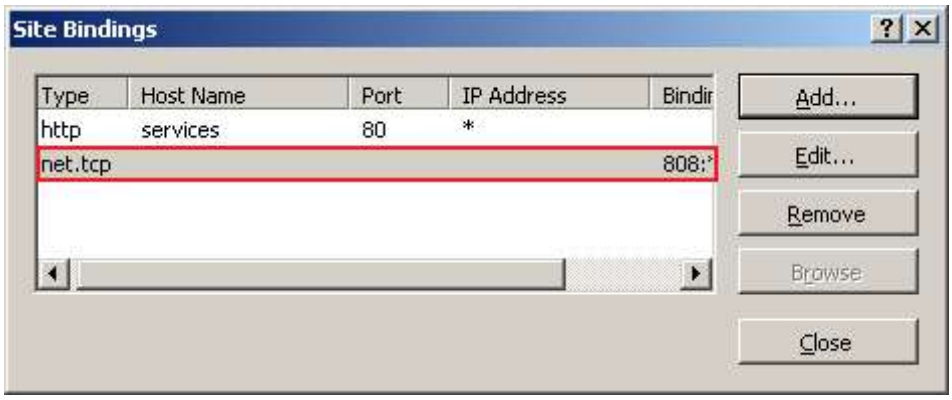
Add binding

- Click on the "Add..." button to add a net.tcp binding. A configuration box "Add Site Binding" would appear. Select "net.tcp" in the "Type" drop-down box, and write 808:* in the "Binding Information" text box (to specify the port) before pressing OK.



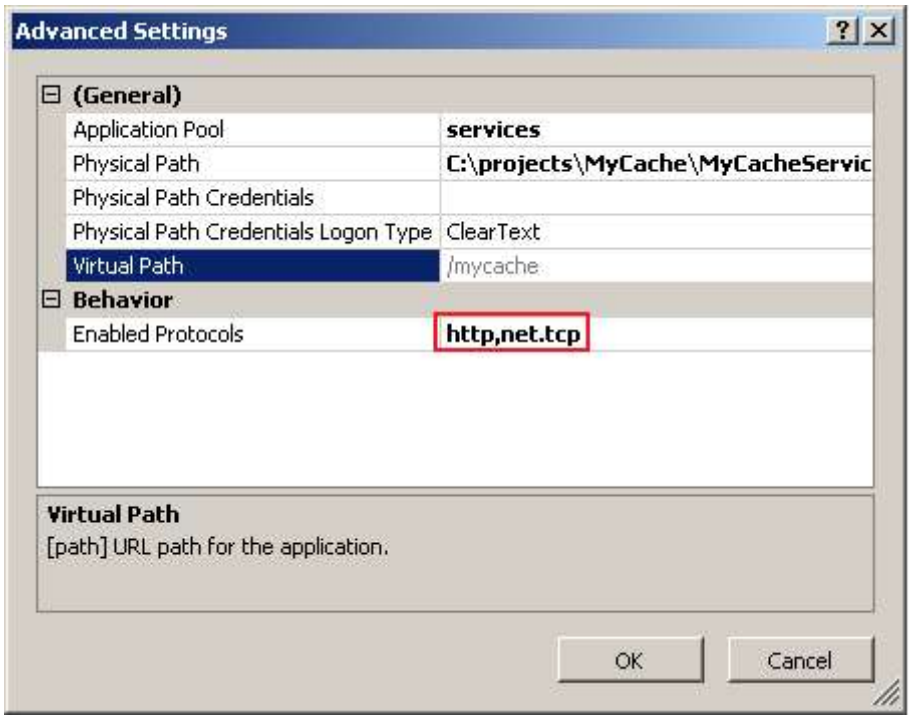
Add net.tcp binding

- Click on the "Close" button on the "Site Bindings" form. The net.tcp binding has been configured for the site "services", and it will be shown in the following box:



net.tcp binding added

- Press "Close" to finish the configuration. Now, right click on the application "mycache", select "Manage Application", and click on "Advanced Settings" to configure the application to allow the net.tcp protocol. The following configuration form would appear:



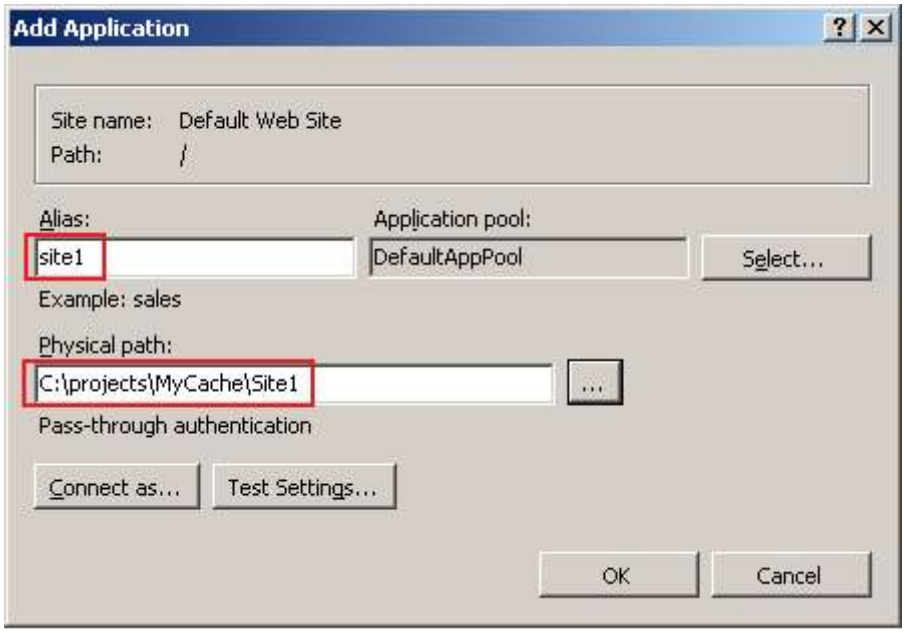
Allow net.tcp protocol

- Write "http, net.tcp" in the "Enabled protocols" text box, and press "OK" to save the settings.

Configuring client applications

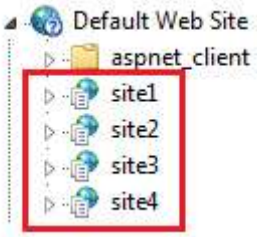
Follow these steps to configure the client ASP.NET applications which uses MyCache as their distributed caching engine.

- Create an application "Site1" under the "Default Web Site" (or you can create a new site "Site1" in IIS, but the sample application assumes an application created under "Default web site"), and set the physical path to the directory of "Site1" within the sample application folder, as follows:



Create client sites in IIS

- Make sure that the application pool of the newly created application is of .NET version 4.0.
- Create three other ASP.NET applications under "Default Web Site" having names "Site2", "Site3", and "Site4", following the same procedure (but by pointing to the right physical path for each application). After creating all four applications, they will look like:



Client sites created

Running client applications

So the hard work has been done, and it is time to see something in action now!

As has been mentioned already, there are two web farms (not a real web farm, rather a simulation of a web farm) in the sample application. These are:

- WebFarm1 (contains Site1 and Site2)
- WebFarm2 (contains Site3 and Site4)

As these applications have been configured in IIS, these applications are accessible via the following URLs:

Sites in **WebFarm1**:

- *http://localhost/site1/default.aspx*
- *http://localhost/site2/default.aspx*

Sites in **WebFarm2**:

- *http://localhost/site3/default.aspx*
- *http://localhost/site4/default.aspx*

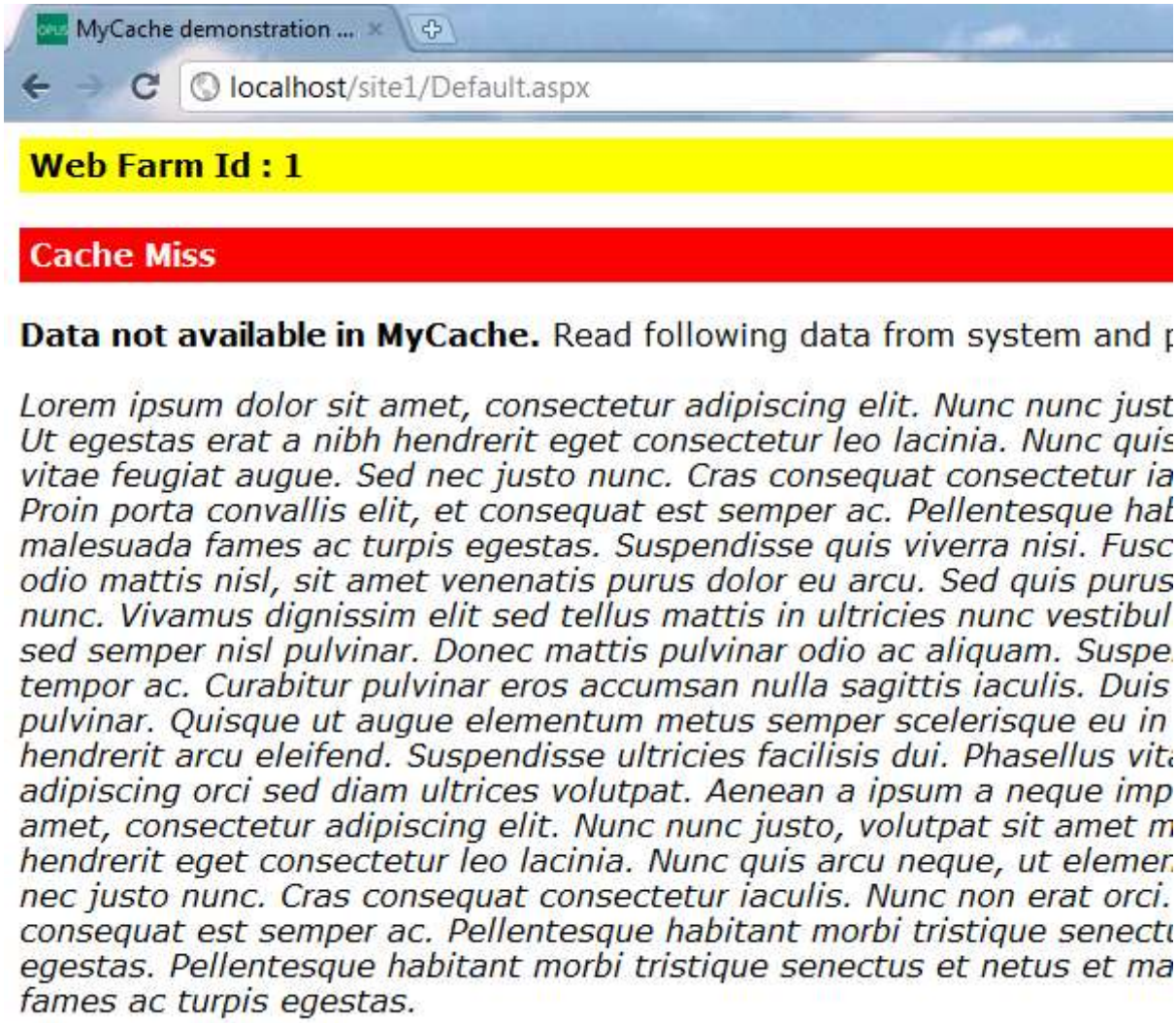
Because Site1 and Site2 belong to the same web farm (WebFarm1), they should share the same cached object stored inside the distributed cache (MyCache). Similarly, Site3 and Site4 belongs to the same web farm (WebFarm2) and should share the same cached

objects stored inside MyCache. This also means, a site in a particular web farm (say, Site1 in WebFarm1) cannot access a cached object which is stored by a site belonging to a different web farm (say, Site3 in WebFarm2).

Testing sites under WebFarm1

When browsed, all four applications read the same text file "*Test.txt*" to read around 2 KB of text data and put it into MyCache when data is not available in MyCache or a particular web farm. Also, when the *Test.txt* file content is changed (manually or programmatically), at least one of the load-balanced sites from each web farm re-reads the data from the text file and puts the updated data into MyCache. There is a conditional check which prevents all sites reading the same data and putting it into the cache in such a situation.

If everything goes perfect, hitting the following URL in the browser would present you the following screen:



Cache miss at Site1

The above output signifies that the data is not available in MyCache. So the application reads the data from the data source (the *Test.txt* text file having 2 KB text data in this case), stores it into the cache, and displays the data in the output.

Now, as the data is already available inside the cache (MyCache), hitting the URL again (or refreshing the page) will result in finding the data in the cache, and hence the data will not have to be re-read from the data source.

Refresh the current page (Ctrl+F5), and you will see the following screen output now:

Cache hit at Site1

The output says, this site belongs to WebFarm1 (WebFarmID = 1). The data is already available in cache for any site under WebFarm1. So hitting the Site2 URL should bring up the following screen, right?

Cache hit at Site2

Voila, it does. MyCache rocks!

Testing sites under WebFarm2

Well, even if the same cached data is available in MyCache, it is available for sites under WebFarm1 only. So now, hitting a URL of an application under WebFarm2 should result in a cache miss and the application should read data from the data source (*Test.txt*) and it put into the cache for WebFarm2. The following screen output will be shown if the Site3 URL is hit in the browser window:

Cache miss at Site3

As the data is already stored in MyCache for WebFarm2, refreshing the current page will result in a cache hit, and hence the following screen output will be shown in the browser:

Cache hit at Site3

Now, according to the above output, this site belongs to WebFarm2 (WebFarmID = 2). So the data should already be available in the cache for any site under WebFarm2, and hence hitting the Site4 URL should bring up the following screen:

Cache hit at Site4

Dude, it perfectly outputs the result as expected!

Testing CacheDependency

We are not done yet.

MyCache has the CacheDependency feature (like what we have in the ASP.NET cache), which is dependent upon a file in the file system. While putting data into MyCache, FileDependency can be specified, and later on, if the file content is modified (manually or programmatically), at least one of the load balanced ASP.NET applications within each web farm re-reads the file content and updates the data for the corresponding web farm in the cache so that all other client applications/sites can get the modified data from MyCache automatically, when required.

To test this functionality, open *Test.txt*, modify the content (say, append a new line "This is a new line added to the existing content." to the existing content), and save the file:

Modifying text file content

Now, browse any site within any one of the web farms (WebFarm1, WebFarm2). You will see that both sites get the modified data from MyCache and shows it in the browser output.

Output from a site in WebFarm1:

Modified data reloaded automatically in Site1

Output from a site in WebFarm2:

Modified data reloaded automatically in Site3

Internal implementation

Well, the demonstration is done, and you know what MyCache has to offer for your distributed caching management needs and how to use it. But, what about the inside stories? How does it really work?

This is Part 1 of the article, which focuses on introducing the component and demonstrating the basic architecture, functionality, and usage of MyCache. Part 2 of the article focuses on the internal implementation issues in detail, and demonstrates how those issues were addressed. Roughly, the following sections are covered:

- MyCacheAPI: The caching API
- Using MyCache in existing systems
- Implementation of the "in-memory caching" engine
- Providing the caching service via net.tcp binding
- Implementation of ASP.NET caching like features
- Serving multiple load-balanced web farms
- Locking/unlocking
- Performance

Have a look at the next part of this article here:

- MyCache: Distributed caching engine for an ASP.NET web farm - Part II: The internal details

Give MyCache a try and let me know your feedback. I'll be more than happy to hear from you.

License

This article, along with any associated source code and files, is licensed under The Code Project Open License (CPOL)

Share

About the Author

Al-Farooque Shubho
Founder SmartAspects
Bangladesh

I write codes to make life easier, and that pretty much describes me.

Comments and Discussions

You must Sign In to use this message board.

Search Comments

First Prev Next

where is the mycache code?
Member 11165481 19-Oct-14 17:48

clarification
Omar Gameel Salem 12-Aug-14 14:48

My vote of 5
vikro 26-Jun-13 3:00

My vote of 5
Manoj Kumar Choubey 2-Apr-12 1:18

My vote of 5
Zorro 15-Feb-12 5:25

ENDPOINTNOTFOUNDEXCEPTION: No DNS entries exist for the host services)
Member 4082860 5-May-11 20:59

No such host is known error
Member 4082860 5-May-11 20:38

My vote of 3
grey alexis 14-Apr-11 5:12

nice article !!!
lee 3 14-Mar-11 23:43

My vote of 5
Monjurul Habib 4-Mar-11 10:09

What about sticky sessions??
Sumit_Gupta 29-Dec-10 20:40

Re: What about sticky sessions??
Al-Farooque Shubho 30-Dec-10 0:17

Re: What about sticky sessions??
Sumit_Gupta 30-Dec-10 1:26

Re: What about sticky sessions??
Ashley|B 14-Jan-11 1:03

Thank you
Khazbak 28-Dec-10 0:56

Re: Thank you
Al-Farooque Shubho 28-Dec-10 2:59

My vote of 5
Khazbak 28-Dec-10 0:54

Runtime.Caching implementation
gocebe 26-Dec-10 11:34

Re: Runtime.Caching implementation
Al-Farooque Shubho 26-Dec-10 15:09

My vote of 5
prasad02 22-Dec-10 4:10

My vote of 5
Sajjan Kr Mishra 21-Dec-10 4:27

My vote of 5
23741 18-Dec-10 7:23

Memcached analog
Dmitry Fisenko 8-Dec-10 23:53

GemFire distributed data (cache) factory
Nedim Sabic 8-Dec-10 22:52

Re: GemFire distributed data (cache) factory
Al-Farooque Shubho 8-Dec-10 22:57

Refresh 1

General News Suggestion Question Bug Answer Joke Praise Rant Admin

Use Ctrl+Left/Right to switch messages, Ctrl+Up/Down to switch threads, Ctrl+Shift+Left/Right to switch pages.

Permalink
Advertise
Privacy
Cookies
Terms of Use

Layout: fixed | fluid

Article Copyright 2010 by Al-Farooque Shubho
Everything else Copyright © CodeProject, 1999-2021

Web03 2.8.20210412.1