Introduction

In our last project, we developed a site for a large number of users. Larger number of client means larger number of requests to your web server and heavy load on the network causing performance issue. For solving this problem, I worked on using caching on our web application. I then thought, why not write an article on CodeProject on it. I am writing this article about whatever I have learned from my practical experience, net surfing, and different books for completing my assignment. Most of the things are very commonly known to a lot of the readers, but I have tried to write in a different way so that it can be understood by beginners also. An interesting thing that I have found while writing the article is setting up different locations for caching. I have also given the corresponding Visio diagram. Hope you will like it.

What is Caching?

Web applications are accessed by multiple users. A web site can have a heavy load on the site which can increase exponentially, which can slow down the server as well as the access of the site. Slow access is the most common problem for web sites when accessed by a large number of clients simultaneously. For resolving this problem, we can use a high level of hardware configuration, load balancer, high bandwidth, but load is not the only reason that makes a website slow, so we need to provide a kind of mechanism which will also provide fast data access and provide performance improvements. Caching provides the solution.

*Caching* is a technique where we can store frequently used data, and web pages are stored temporarily on the local hard disk for later retrieval. This technique improves the access time when multiple users access a web site simultaneously, or a single user accesses a web site multiple times. Caching for web applications can occur on the client (browser caching), on a server between the client and the web server, (proxy caching / reverse proxy caching), and on the web server itself (page caching or data caching).

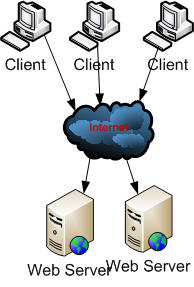
We can choose a big amount of time to store cached data so it improves the performance but it does not solve our purpose every time. If we consider the load on a Web Server we have to consider the location where the cached data is stored. The following section will describe different locations for storing cached data.

Different Caching Locations

Caching in a web application can be done either on the client side (client browser), in between the client and the server (proxy and reverse proxy caching), or on the server side (data caching/page output caching). So we can classify caching locations like this:

1. Client Caching
2. Proxy Caching
3. Reverse Proxy Caching
4. Web Server Caching

**1. Client Caching**: In Client Caching, the client browser performs caching by storing cached data on the local disk as a temporary file or in the browser internal memory. This provides quick access of some information which reduces the network load and the server load also. This information can't be shared by other clients so it is client specific.



***Fig. 1.0: Client caching***

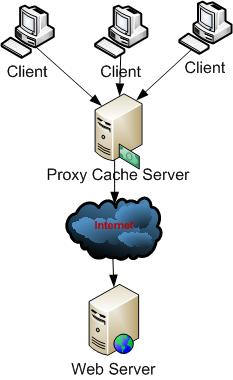
**Advantages**

1. Data that is cached on the local client can be easily accessed
2. Reduces network traffic

**Disadvantages**

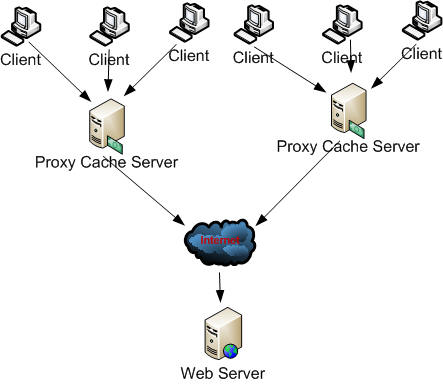
1. Cached data is totally browser dependent, so it is not shareable

**2.** **Proxy Caching**: The main disadvantage of client caching is data that is stored on the client browser is client specific. Proxy caching uses a dedicated server that stores caching information in between the client and the web server in a shared location so that all clients can use the same shared data. The proxy server (e.g., Microsoft Proxy Server) fulfills all the requests for the web page without sending out the request to the actual web server over the internet, resulting in faster access.



***Fig. 1.0: Proxy caching***

Proxy caches are often located near network gateways to reduce bandwidth usage. Some times multiple proxy cache servers are used for larger number of clients. This is called a cache array.



***Fig. 1.1: Cache array***

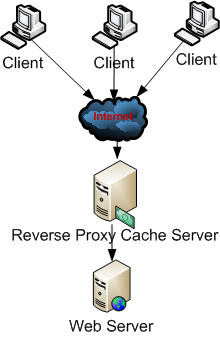
**Advantages**

1. Data that is cached on a proxy server can be accessed easily
2. Reduces network traffic

**Disadvantages**

1. Involves deployment and infrastructure overhead to maintain a proxy cache server

**3. Reverse Proxy Caching**: Some proxy cache servers can be placed in front of the web server to reduce the number of requests that they receive. This allows the proxy server to respond to frequently received requests and only pass other requests to the web server. This is called a reverse proxy.



***Fig. 1.2: Reverse proxy caching***

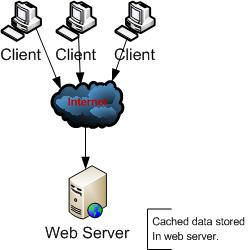
**Advantages**

1. Data that is cached on a reverse proxy server can be accessed easily
2. Reduces the number of requests

**Disadvantages**

1. As the server is configured in front of the web sever, it could increases network traffic

**4. Web Server Caching**: In web server caching, cached data is stored inside the web server. Data caching and page caching uses the web sever caching mechanism.



***Fig. 1.3: Web server caching***

**Advantages**

1. Improves the performance of sites by decreasing the round trip of data retrieval from the database or some other server

**Disadvantages**

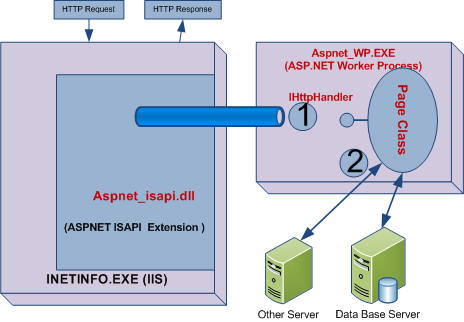
1. Increases network load

Advantages of Caching

1. Reduces server load
2. Reduces bandwidth consumption

Caching Opportunity in ASP.NET

ASP.NET provides support for page, partial page (fragment), and data caching. Caching a page that is dynamically generated is called page output caching. In page caching, when a page that is dynamically generated is cached, it is accessed only the first time. Any subsequent access to the same page will be returned from the cache. ASP.NET also allows to cache a portion of a page, called partial page caching or fragment caching. Other server data are cached (e.g., SQL Server data, XML data) that can be easily accessed without re-retrieving data using data caching. Caching reduces the number of round trips to the database and other data sources. ASP.NET provides a full-featured data cache engine, complete with support for scavenging (based on cache priority), expiration, file and key, and time dependencies. There are two locations where caching can be used to improve performance in ASP.NET applications.



***Fig 1.4: Caching Opportunity in ASP.NET***

In the above picture, (1) is used for return caching of page which means it is used in output caching, and (2) saves the round trip by storing the data using data caching.

ASP.NET supports two types of expiration policies, which determine when an object will be expired or removed from the cache.

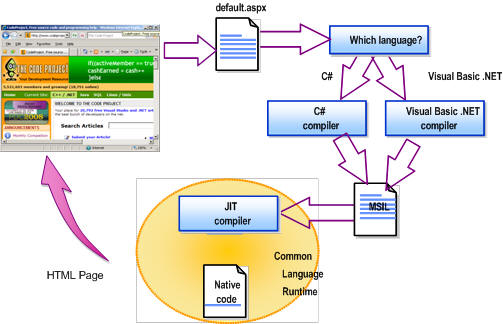
**Absolute expiration**: Determines that the expirations occur at a specified time. Absolute expirations are specified in full-time format (hh:mm:ss). The object will expire from the cache at the specified time.

ASP.NET supports three types of caching:

1. Page output caching [Output caching]
2. Fragment caching [Output caching]
3. Data caching

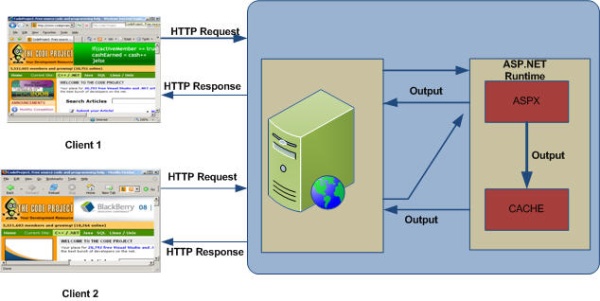
Different Types of Caching

**1. Page Output Caching**: Before starting page output caching, we need to know the compilation process of a page, because based on the generation of the page, we should be able to understand why we should used caching. An ASPX page is compiled in a two stage process. First, the code is compiled into Microsoft Intermediate Language (MSIL). Then, the MSIL is compiled into native code (by the JIT compiler) during execution. The entire code in an ASP.NET web page is compiled into MSIL when we build sites, but at the time of execution, only the portion of MSIL converted to native code which is needed by the user or user requests is executed, which improves performance.



***Fig. 1.5: ASP.NET page execution process***

Now whatever we are getting, if there is some page which changes frequently, JIT needs to compile it every time. We can use page output caching for those pages whose content is relatively static. So rather than generate a page on each user request, we can cache the page using page output caching so that it can be accessed from the cache itself. Pages can be generated once and then cached for subsequent fetches. Page output caching allows the entire content of a given page to be stored in the cache.



***Fig. 1.5: Page output caching***

In the picture, when the first request is generated, the page is cached and for the same page request in future, the page is retrieved from the cache rather that regenerating the page.

For output caching, an OutputCache directive can be added to any ASP.NET page, specifying the duration (in seconds) that the page should be cached.

**Example**

http://www.codeproject.com/images/minus.gifCollapse | [Copy Code](http://www.codeproject.com/Articles/29899/Exploring-Caching-in-ASP-NET)

<%@ Page Language="C#" %>

<%@ OutputCache Duration='300' VaryByParam='none' %>

<html>

<script runat="server">

protected void Page\_Load(Object sender, EventArgs e) {

lbl\_msg.Text = DateTime.Now.ToString();

}

</script>

<body>

<h3>Output Cache example</h3>

<p>Page generated on:

<asp:label id="lbl\_msg" runat="server"/></p>

</body>

</html>

We can also set the caching property from the code-behind:

http://www.codeproject.com/images/minus.gifCollapse | [Copy Code](http://www.codeproject.com/Articles/29899/Exploring-Caching-in-ASP-NET)

void Page\_Load(Object sender, EventArgs e) {

Response.Cache.SetExpires(DateTime.Now.AddSeconds(360));

Response.Cache.SetCacheability(

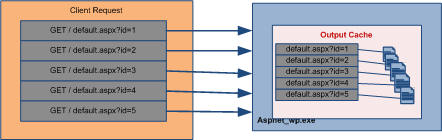
HttpCacheability.Public);

Response.Cache.SetSlidingExpiration(true);

\_msg.Text = DateTime.Now.ToString();

}

We have to mention the duration and VaryByParam attribute. Duration defines how long the cache will persist. VaryByParam defines if there the cache varies with parameter values.



***Fig. 1.6: Caching multiple pages based on parameters***

As shown in the above picture, if we are using a query string for a page and we need to cache all pages based on the query string, we have to use the VaryByParam attribute of output cache. Based on the query string, data should be cached, and when the user requests a page with a query string (ID in the picture), page should be fetched from the cache. The following example describes the use of VaryByParam attributes.

**Example:**

http://www.codeproject.com/images/minus.gifCollapse | [Copy Code](http://www.codeproject.com/Articles/29899/Exploring-Caching-in-ASP-NET)

<%@ OutputCache Duration="60" VaryByParam="\*" %>

<! page would cached for 60 seconds, and would create a separate cache

entry for every variation of querystring -->

The following table shows you the most commonly used and most important attributes of output cache:

| **Attribute** | **Values** | **Description** |
| --- | --- | --- |
| Duration | Number | Defines how long the page will be cached (in seconds). |
| Location | 'Any'  'Client'  'Downstream'  'Server'  'None' | It defines the page cache location. I have discussed it later in detail. |
| VaryByCustom | 'Browser' | Vary the output cache either by browser name and version or by a custom string. |
| VaryByParam | 'none' '\*' | This is a required attribute, which is required for a parameter for the page. I have already discussed this. |

All the attributes that we specify in an OutputCache directive are used to populate an instance of the System.Web.HttpCachePolicy class. The complete implementation of cache policies provided by ASP.NET is encapsulated in the HttpCachePolicy class. Following is another implementation of caching from the code-behind.

**Output caching location**

As I have already mentioned, we can store cached data in different locations like client, server, or in between the client and the server. Now I am going to discuss how to set the location of cached data. If we store cached data, it saves the page rendering time by fetching data from the cache. There is another way that we can save cached data on the client browser, which reduces network traffic. The OutputCache directive on a page enables all three types of caching—server, client, and proxy—by default.

The following table shows you the location details. It shows the location of cache and the effects of the Cache-Control and Expires headers.

| **Value of Location** | **Cache-Control Header** | **Expires Header** | **Page Cached on Server** | **Description** |
| --- | --- | --- | --- | --- |
| 'Any' | public | Yes | Yes | Page can be cached on the browser client, a downstream server, or the server. |
| 'Client' | private | Yes | No | Page will be cached on the client browser only. |
| 'Downstream' | public | Yes | No | Page will be cached on a downstream server and the client. |
| 'Server' | no-cache | No | Yes | Page will be cached on the server only. |
| 'None' | no-cache | No | No | Disables output caching for this page. |

For example, if you specify a value of Client for the Location attribute of an OutputCache directive on a page, the page would not be saved in the server cache, but the response would include a Cache-Control header (pages can indicate whether they should be cached on a proxy by using the Cache-Control header) value of private and an Expires header (HTTP response, indicating the date and time after which the page should be retrieved from the server again) with a timestamp set to the time indicated by the Duration attribute.

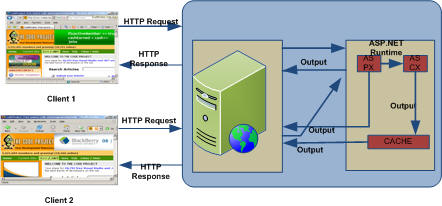
**Example**

http://www.codeproject.com/images/minus.gifCollapse | [Copy Code](http://www.codeproject.com/Articles/29899/Exploring-Caching-in-ASP-NET)

<%@ OutputCache Duration='120' Location='Client' VaryByParam='none' %>

This would save the cache for 120 seconds and cached data should not be saved on the server, it should be stored only on the client browser.

**2. Page Fragment Caching**: ASP.NET provides a mechanism for caching portions of pages, called page fragment caching. To cache a portion of a page, you must first encapsulate the portion of the page you want to cache into a user control. In the user control source file, add an OutputCache directive specifying the Duration and VaryByParam attributes. When that user control is loaded into a page at runtime, it is cached, and all subsequent pages that reference that same user control will retrieve it from the cache.



***Fig. 1.7: Fragment caching***

The following example shows you the details of fragment caching:

**Example**

http://www.codeproject.com/images/minus.gifCollapse | [Copy Code](http://www.codeproject.com/Articles/29899/Exploring-Caching-in-ASP-NET)

<!— UserControl.ascx —>

<%@ OutputCache Duration='60'

VaryByParam='none' %>

<%@ Control Language="'C#'" %>

<script runat="server">

protected void Page\_Load(Object src, EventArgs e)

{

\_date.Text = "User control generated at " +

DateTime.Now.ToString();

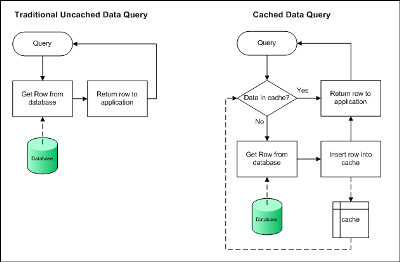
}

</script>

<asp:Label id='\_date' runat="'server'" />

Here I have user caching on a user control, so whenever we use it in a page, part of the page will be cached.

**3. Data Caching**: Caching data can dramatically improve the performance of an application by reducing database contention and round-trips. Simply, data caching stores the required data in cache so that the web server will not send requests to the DB server every time for each and every request, which increases web site performance. For data caching, we need to cache data which is accessible to all or which is very common. The data cache is a full-featured cache engine that enables you to store and retrieve data between multiple HTTP requests and multiple sessions within the same application.



***Fig. 1.8: Data caching***

The above image shows how data can be accessed directly from the database server and how data is retrieved using cache. Data caching is not only related with SQL Server, we can store in other data sources as shown on Fig 1.4.

Now let us see how we can implement data caching in our web application. There are three different ways to add data or objects into cache. But based on the situation, we have to access it differently. These methods are Cache[], Cache.add(), cache.insert(). The following table will show you the clear difference of the there methods.

|  | **Stores data in cache** | **Supports dependency** | **Supports expiration** | **Support priority settings** | **Returns object** |
| --- | --- | --- | --- | --- | --- |
| cache[] | Yes | No | No | No | No |
| cache.insert() | Yes | Yes | Yes | Yes | No |
| cache.add() | Yes | Yes | Yes | Yes | Yes |

cache[] is a property that is very simple to use but cache.insert() and cache.add() give us more control on the cached data.

Now we should look into the details of the Cache.Insert() and Cache.Add() methods. Cache.Insert() has four overloads whereas Cache.Add() has no overloaded methods. The following table shows the most commonly used properties for those methods.

| **Property** | **Type** | **Description** |
| --- | --- | --- |
| Key | String | A unique key used to identify this entry in the cache. |
| Dependency | CacheDependency | A dependency this cache entry has—either on a file, a directory, or another cache entry—that, when changed, should cause this entry to be flushed. |
| Expires | DateTime | A fixed date and time after which this cache entry should be flushed. |
| Sliding Expiration | TimeSpan | The time between when the object was last accessed and when the object should be flushed from the cache. |
| Priority | CacheItemPriority | How important this item is to keep in the cache compared with other cache entries (used when deciding how to remove cache objects during scavenging). |
| OnRemoveCallback | CacheItem RemovedCallback | A delegate that can be registered with a cache entry for invocation upon removal. |

The first two are mandatory for Cache.Insert() methods, whereas others vary based on the situation.

**Cache Dependency**

Using cache dependency, we can set the dependency of the cache with some data or entity that might change. So we can set the dependency of cache by which we can update/remove cache. There are three types of dependencies supported in ASP.NET:

* File based dependency
* Key based dependency
* Time based dependency

**File Based Dependency**: File-based dependency invalidates a particular cache item when a file(s) on the disk changes.

Using cache dependency, we can force ASP.NET to expire cached data items from the cache when the dependency file changes. We can set the dependency to multiple files also. On such cases, the dependency should be built from an array of files or directories.

**Use**: File based dependency is very useful when you need to update data that is displayed to the user based on some changes on a file. For example, a news site always shows data from a file, and if some breaking news comes, they just update the file and the cache should expire, and during the expiry time, we can reload the cache with updated data using OnRemoveCallBack.

**Key Based Dependency**: Key-based dependency invalidates a particular cache item when another cache item changes.

**Use**: This is useful when we have multiple interrelated objects in the cache and if one of the objects changes, we need to updated or expire all of them.

**Time Based Dependency**: Time-based dependency causes an item to expire at a defined time. The Cache.Insert() method of the Cache class is used to create a time-based dependency. Two types of time based dependency are available.

* Absolute
* Sliding

**Absolute**: Sets an absolute time for a cache item to expire. Absolute expirations are specified in full-time format (hh:mm:ss). The object will be expired from the cache at the specified time.

**Sliding**: Resets the time for the item in the cache to expire on each request. This is useful when an item in the cache is to be kept alive so long as requests for that item are coming in from various clients.

In addition to these dependencies, ASP.NET allows the following:

**Automatic expiration**: The cache items that are underused and have no dependencies are automatically expired.

**Support for callback**: The cache object can be configured to call a given piece of code that will be executed when an item is removed from the cache. This gives you an opportunity to update the cache. We can use OnRemoveCallback().

Caching Considerations

**Output Caching Considerations**

1. Enable output caching on a page that is frequently accessed and returns the exact same contents for all of those accesses.
2. When enabling output caching for a page, be sure not to introduce incorrect behavior and/or rendering for any particular client.
3. Determine the duration of the cached page carefully to balance speed of access (throughput) with memory consumption and cache coherency correctness.
4. Consider enabling sliding expiration on a page if you end up using VaryByParam='\*'.

**Data Caching Consideration**

1. The data cache is not a container for shared updateable state.
2. Cache data that is accessed frequently and is relatively expensive to acquire.
3. If data is dependent on a file, directory, or other cache entry, use a CacheDependency to be sure it remains current.

Suggested Uses of Caching Types

| **Situation** | **Suggested Caching Type** |
| --- | --- |
| The generated page generally stays the same, but there are several tables shown within the output that changes regularly. | Use fragment caching. |
| The generated page constantly changes, but there are a few objects that don’t change very often. | Use data caching for the objects. |
| The generated page changes every few hours as information is loaded into a database through an automated processes. |  |