Migrating from Windows Server AppFabric Caching to ScaleOut StateServer®

Revision: 1.6 - May 9, 2017

Table of Contents

1. Introduction

This guide provides information and advice to developers, system administrators, and managers who are pursuing a migration from Microsoft Windows Server AppFabric (WSAF) Caching to ScaleOut StateServer (SOSS). It outlines the key differences between the two products and offers guidance to help you complete a successful migration.

The topics in this guide include:

- Key architectural differences between the two products that may need to be considered when deploying and configuring ScaleOut StateServer
- Guidance for converting to ScaleOut's ASP.NET session state and output cache providers
- Guidance for developers who are converting cache client applications, with equivalent API calls for core features and best practices for making the most effective use of ScaleOut APIs
- An overview of additional features in ScaleOut StateServer that can bring new levels of functionality and reliability to your application

1.1. Feedback and Questions

ScaleOut Software welcomes your feedback. Please send your comments and questions to the ScaleOut support team.

1.2. Other Resources

Documentation

Additional product documentation and guides are available on the Product Documentation page of ScaleOut Software's web site.

WSAF Caching Compatibility Library

This guide focuses on migrating applications to ScaleOut's APIs. For users that need to continue using WSAF-style Caching APIs, ScaleOut Software also offers a WSAF Caching Compatibility Library that is source-compatible with AppFabric's DataCache class. This library is shipped in version 5.4 and later.

Using the WSAF Caching Compatibility Library involves just a few easy steps:

1. Add the soss_wsaf_compat.dll assembly as a reference to your project. This assembly can be found in the ScaleOut StateServer installation (typically C:\Program Files\ScaleOut_Software\StateServer) under the Compat \WSAF_Caching folder.
2. Change the "`using Microsoft.ApplicationServer.Caching;`" statements in your source files to "`using Soss.Compat.WSAF;`".
3. Recompile your project to start using the WSAF Caching Compatibility Library.

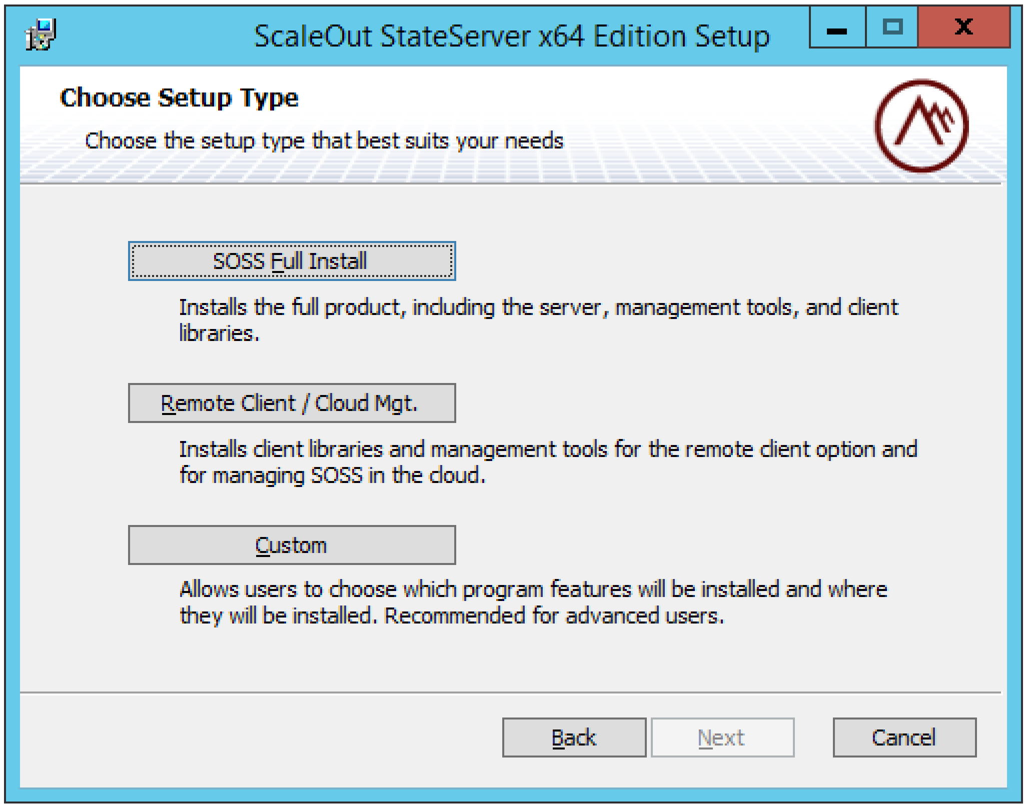
More information about the compatibility library can be found on the Product Documentation page of ScaleOut Software's web site.

2. Migration Roadmap

The caching APIs and ASP.NET providers supplied by ScaleOut StateServer and Windows Server AppFabric (WSAF) Caching share many similar concepts and features, so a migration can often be performed with minimal changes to your application. A typical migration process will involve the following steps:

- Download and install the ScaleOut StateServer service on one or more host systems using ScaleOut's lightweight MSI package. This installer can also be run on client machines to install the libraries needed by your application to connect to remote ScaleOut hosts.

Figure 1. The ScaleOut Windows Installer


ScaleOut StateServer x64 Edition Setup
Choose Setup Type
Choose the setup type that best suits your needs
SOSS Full Install

Installs the full product, including the server, management tools, and client libraries.

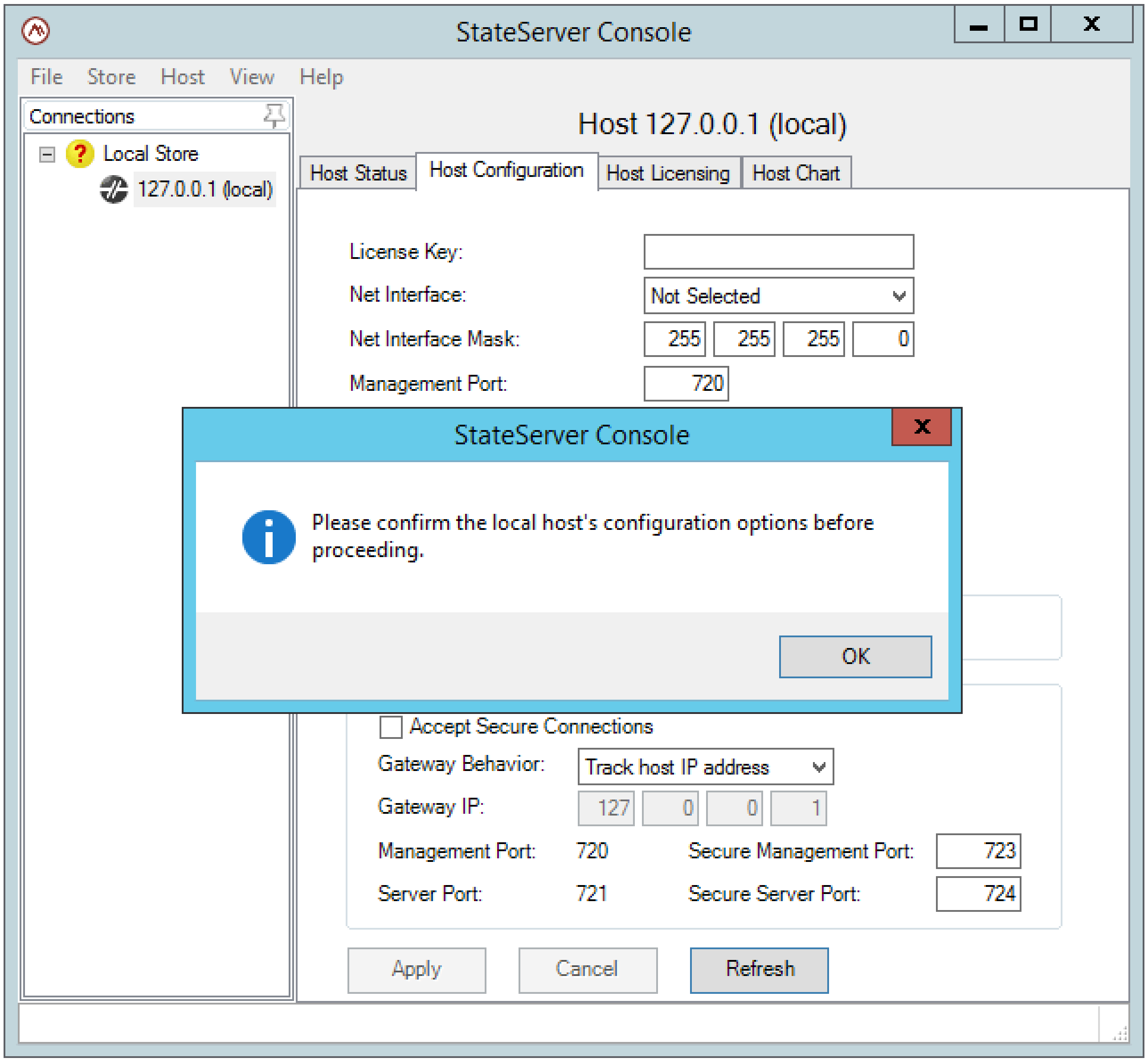
Remote Client / Cloud Mgt.

Installs client libraries and management tools for the remote client option and for managing SOSS in the cloud.

Allows users to choose which program features will be installed and where they will be installed. Recommended for advanced users.

- Once installation is complete, the ScaleOut Management Console application will automatically start and prompt you to configure the SOSS service. At a minimum, configuration involves selecting a network address for the service to use and, optionally, entering a license key to enable various product features. If you are only installing client libraries on a system, the Console will ask you how applications on the system should connect to SOSS hosts.

Figure 2. Initial Configuration



Host 127.0.0.1 (local)

License Key:
Net Interface: Not Selected
Net Interface Mask: 255 255 255 0
Management Port: 720

StateServer Console

Please confirm the local host's configuration options before proceeding.

Accept Secure Connections
Gateway Behavior: Track host IP address
Gateway IP: 127 0 0 1
Management Port: 720 Secure Management Port: 723
Server Port: 721 Secure Server Port: 724

- After using the SOSS Console to select a network address for the ScaleOut service to use, the service will automatically detect other ScaleOut hosts running on the subnet and will become part of the data grid's membership with no further configuration needed. This guide's **Server Configuration and Deployment** topic has an overview of ScaleOut's peer-to-peer architecture.

- It is recommended that you review this guide's **Overview of Key Architectural Differences** section to learn more about the similarities and important differences that you will encounter when moving from Windows Server AppFabric to ScaleOut StateServer.

- If you will be using ScaleOut to store an ASP.NET web application's session state or output cache data, you will need to modify your `web.config` to swap out the WSAF providers with the ScaleOut providers. This guide's sections on **Using the ASP.NET Session State Provider** and **Using the ASP.NET Output Cache Provider** provide instructions on how to configure the providers.

- If you will be migrating a cache client application that was using AppFabric's `DataCache` class, this guide's **Migrating a Cache Client** section provides information about how to use equivalent features in ScaleOut's `NamedCache` API.

3. Terminology

The following table highlights common terms and concepts used in Windows Server AppFabric Caching and their corresponding terms in ScaleOut StateServer.

Windows Server AppFabric Term	**Equivalent SOSS Term**
Cluster	In-Memory Data Grid (IMDG), or "store"
Cache host	ScaleOut host
Cache client	Client application
Local cache	Client cache
Primary object (high availability)	Master object
Secondary object (high availability)	Replica object
`DataCache` class	`NamedCache` class
`DataCacheFactory` class	`CacheFactory` class

4. Overview of Key Architectural Differences

4.1. Server Configuration and Deployment

ScaleOut StateServer takes a decentralized, peer-to-peer approach to cluster management. Differences between configuration and management of ScaleOut StateServer (SOSS) and Windows Server AppFabric Caching are highlighted in the following sections.

Configuration Storage

Windows Server AppFabric Caching

WSAF Caching requires storage of cluster configuration information in a single, centralized location, such as a SQL Server database or an XML file that is available to all cache hosts from a shared network folder.

ScaleOut StateServer

ScaleOut StateServer does not require centralized storage for cluster configuration. ScaleOut hosts automatically discover each other on a network subnet. When global, cluster-wide configuration changes are applied to a host, those changes are shared with the all of the hosts in the cluster and stored locally on each host.

Tip

ScaleOut StateServer's automatic host discovery feature uses IP multicast to allow hosts to find each other on a network. If multicast is disabled or not allowed on the network, a manually configured roster of potential hosts in the group can be set up and is automatically shared between ScaleOut hosts. More information about manual group management is available in the Group Management topic of the *ScaleOut StateServer Help* online guide.

Lead Hosts

Windows Server AppFabric Caching

WSAF Caching clusters may be configured to store cluster-wide settings in an XML file on a shared network folder. In this configuration, the cluster management role must be performed by one or more special cache hosts that are called "lead hosts". If a majority of lead hosts are offline then the entire cluster shuts itself down.

ScaleOut StateServer

ScaleOut StateServer does not use the concept of "lead hosts" to perform cluster management duties. A ScaleOut data grid is fully peer-to-peer, and all hosts in the grid share and coordinate cluster management. Hosts can (and should) be configured and deployed in an identical manner. This approach to shared responsibility allows you to take any host offline without affecting cluster management.

Platform Requirements

Windows Server AppFabric Caching

Cache hosts in Windows Server AppFabric Caching must run on the Windows platform.

ScaleOut StateServer

ScaleOut StateServer hosts can be deployed as Windows or Linux servers. ScaleOut's communication protocols are platform agnostic; .NET client applications running on Windows can seamlessly connect to ScaleOut daemons running on Linux systems.

4.2. Management

Windows Server AppFabric Caching

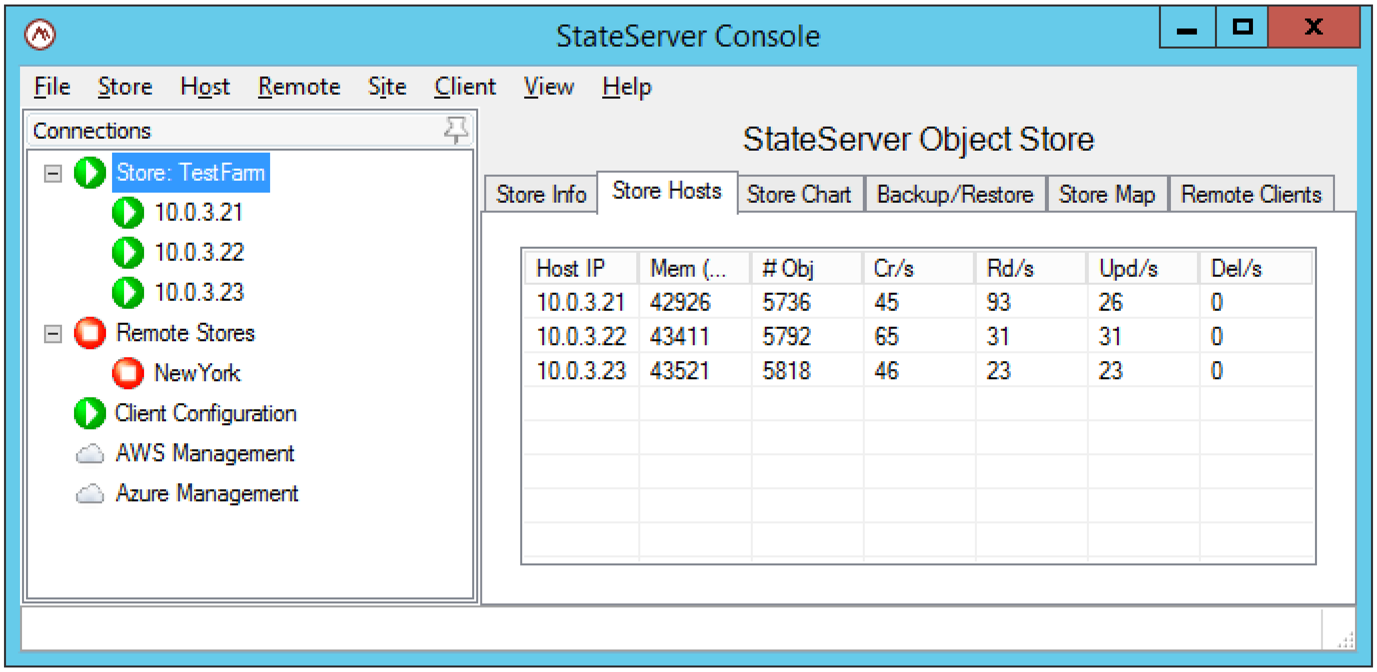
WSAF Caching clusters are configured and administered using a set of Windows PowerShell commands.

ScaleOut StateServer

ScaleOut StateServer offers four approaches to managing a data grid:

- The SOSS Console (pictured below), a straightforward Windows-based management tool, capable of managing all hosts in the data grid from a single location.
- The `soss.exe` command line tool, available on both Windows and Linux systems, for command line management or scripted administration.
- .NET management APIs, for programmatic management of the grid through your custom .NET application.
- A set of Powershell commands to configure and administer the in-memory data grid.

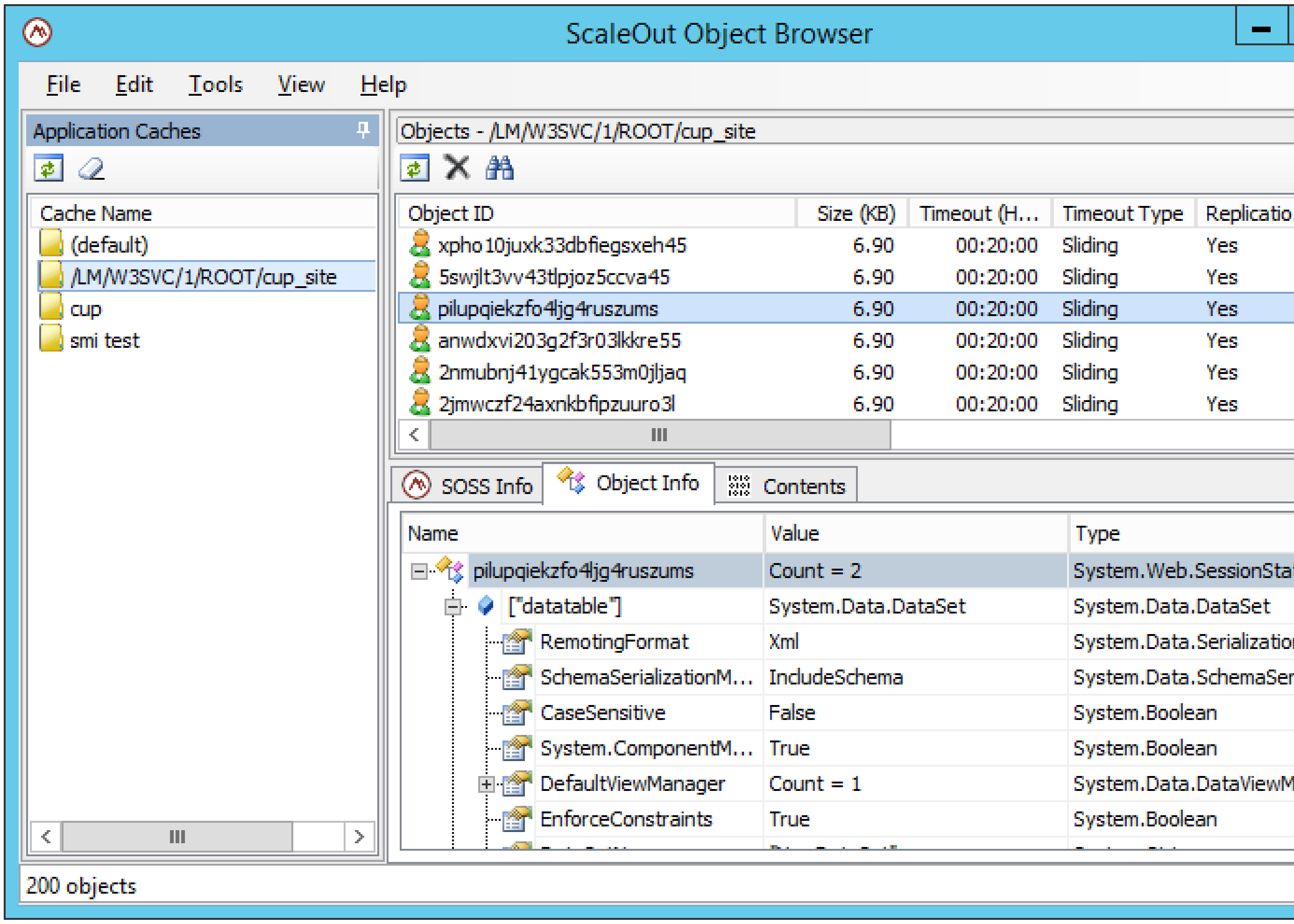
Figure 3. The SOSS Console Application



Host IP	Mem (...	# Obj	Cr/s	Rd/s	Upd/s	Del/s
10.0.3.21	42926	5736	45	93	26	0
10.0.3.22	43411	5792	65	31	31	0
10.0.3.23	43521	5818	46	23	23	0

In addition to the tools that manage the hosts in your data grid, ScaleOut's Object Browser tool allows for management of *individual objects* in your grid. The Object Browser is able to browse the objects that are stored in your farm and view their contents in a UI that is similar to the Visual Studio debugger's Watch Window.

Figure 4. The SOSS Object Browser Application



Objects - /LM/W3SVC/1/ROOT/cup_site

Cache Name
(default)
/LM/W3SVC/1/ROOT/cup_site
cup
smi test

Object ID	Size (KB)	Timeout (H...	Timeout Type	Replicatio
xpho10juxk33dbfiegsxeh45	6.90	00:20:00	Sliding	Yes
5swjlt3vv43tlpjoz5ccva45	6.90	00:20:00	Sliding	Yes
pilupqiekzfo4ljg4ruszums	6.90	00:20:00	Sliding	Yes
anwdxvi203g2f3r03lkkre55	6.90	00:20:00	Sliding	Yes
2nmubnj41ygcak553m0jljaq	6.90	00:20:00	Sliding	Yes
2jmwczf24axnkbfipzuuro3l	6.90	00:20:00	Sliding	Yes

Name	Value	Type
pilupqiekzfo4ljg4ruszums	Count = 2	System.Web.SessionSta
["datatable"]	System.Data.DataSet	System.Data.DataSet
RemotingFormat	Xml	System.Data.Serializatio
SchemaSerializationM...	IncludeSchema	System.Data.SchemaSer
CaseSensitive	False	System.Boolean
System.ComponentM...	True	System.Boolean
DefaultViewManager	Count = 1	System.Data.DataViewM
EnforceConstraints	True	System.Boolean

200 objects

Cmdlets

ScaleOut StateServer's drop-in replacement offers syntax-matching cmdlets for AppFabric veterans. The following commonly used AppFabric cmdlets have been implemented with ScaleOut StateServer in mind. The SOSS PowerShell module can be used by directly importing the SossAdministration Powershell module within Powershell (for example, *Import-Module SossAdministration*) or by using the ScaleOut StateServer PowerShell shortcut which will automatically import the SossAdministration PowerShell module. The ScaleOut StateServer PowerShell shortcut is located in the installation directory.

Store Monitoring

Get-Cache

The Get-Cache command displays a collection of CacheInfo objects that contain meta-data for a NamedCache. The CacheInfo object contains properties for an AppId (a unique identifier for a NamedCache), a NamedCache name (used to create the AppID), and a mapping of regions to hosts.

Get-CacheHost

The Get-CacheHost command displays a collection of all hosts participating in the ScaleOut StateServer store as a set HostInfo objects. The HostInfo object contains properties for an IP address, a HostStatus enum (e.g., active, inactive), and an isLocal boolean.

Get-CacheRegion

The Get-CacheRegion command displays a collection of all regions stored within the ScaleOut StateServer store represented as a set of RegionInfo objects. Each RegionInfo object contains properties for a region, the associated NamedCache and the host that the region lives on.

Get-CacheStatistics

The Get-CacheStatistics command displays the Create/Read/Update/Delete load for the entire ScaleOut StateServer store, or a target host.

Store Administration

Start-CacheHost

The Start-CacheHost command joins a target ScaleOut StateServer host to the active membership group of the ScaleOut StateServer store. When a host joins the active membership, it participates in load balancing and handles user requests as a part of the distributed in-memory data grid.

Start-CacheCluster

The Start-CacheCluster command joins all ScaleOut StateServer hosts to the active membership of the ScaleOut StateServer store.

Stop-CacheHost

The Stop-CacheHost command removes a target ScaleOut StateServer host from the active membership group. When a host leaves the active membership group, it no longer participates in load balancing and does not recieve user requests.

Stop-CacheCluster

The Stop-CacheCluster command removes all ScaleOut StateServer hosts from the active membership group.

Restart-CacheCluster

The Restart-CacheCluster command restarts all of the ScaleOut StateServer processes within the ScaleOut StateServer store. All data stored within the in-memory data grid will be removed.

Get-CacheHostConfig

The Get-CacheHostConfig command displays the configuration parameters for a ScaleOut StateServer host. For a complete description of the configuration parameters, please view the "Configuration Parameters" section within the ScaleOut StateServer help file.

Set-CacheHostConfig

The Set-CacheHostConfig command sets a configuration parameter for a target host. The Get-CacheHostConfig and Set-CacheHostConfig cmdlets can be used in tandem to quickly and easily alter the configuration of each host participating in the ScaleOut StateServer store.

Note

For detailed descriptions and examples of these cmdlets, use the "Help/Get-Help" commands within PowerShell.

Note

Additional ScaleOut StateServer specific cmdlets can be found within the ScaleOut StateServer help file.

Cmdlets Not Implemented

The following cmdlets were not implemented as a part of the drop-in solution with explanations below.

Use-CacheCluster, Export-CacheClusterConfig, Import-CacheClusterConfig, Set-CacheConnectionString, Get-CacheConnectionString, New-CacheCluster, Remove-CacheCluster, Register-CacheHost, Unregister-CacheHost, Add-CacheHost, Remove-CacheHost

> After ScaleOut StateServer (SOSS) is installed, any management API (including PowerShell Cmdlets) will automatically detect the connection settings through the soss_params.txt configuration file and connect to the SOSS store (SOSS does not use connection strings). Also, all configuration changes to a host are automatically persisted to disk (as soss_params.txt in the installation directory). Additionally, ScaleOut StateServer hosts are automatically added to the store through host auto-discovery after installation. To remove a host from the ScaleOut StateServer store, issue the Stop-CacheHost command to remove the target host from the SOSS store’s active membership.

New-Cache, Remove-Cache

> ScaleOut StateServer’s (SOSS) NamedCaches are automatically created and configured within an application. Each NamedCache has an associated namespace which is registered and persisted with the lifetime of the SOSS service. The NamedCache object only lives within the scope of the application. There is no configuration object for an administrator to create or remove.

Add-CacheAdmin, Remove-CacheAdmin

> Applications can register specific users with the ScaleOut StateServer service through the ScaleOut StateServer pluggable authorization provider. Please see the ScaleOut StateServer main help file for more details. For ease of use, cache-administration is strictly based on system priviliges.

Set-CacheConfigurationLog

> ScaleOut StateServer’s log file is located in the installation directory and named "soss_tlog0.txt" for easy access.

Get-CacheClusterHealth, Get-CacheClusterInfo, Test-CacheClusterConnection

> The ScaleOut StateServer store’s overall health can be verified by issuing the Test-Store command.

Scenarios

First Host Install

> For detailed installation instructions, visit the SOSS help file’s "Installation" sub-topic. The following quick start instructions will install the ScaleOut StateServer host on a single machine: Download the SOSS installer from

the scaleoutsoftware.com website. Run the soss_setup64.msi installer to install ScaleOut StateServer on the system (or soss_setup32.msi for 32-bit machines). After the installation has completed, the SOSS service will start. After the SOSS service is installed, bind the service to a network interface with the Set-CacheHostConfig command, specifying the -NetInterface parameter. Example: "Set-CacheHostConfig -HostAddress 127.0.0.1 -NetInterface 10.0.4.0".

Start First Host

After the First Host Install, the state of the SOSS store is a single SOSS host that has been configured and is an inactive participant of an automatically created host group. The host needs to "Join" the host group to create a store and accept the workload. The Start-CacheHost command with the -HostAddress parameter will join the host to the SOSS store. Alternatively, the Start-CacheCluster command will join all inactive hosts to the SOSS store. Example: "Start-CacheHost -HostAddress 10.0.4.19" or "Start-CacheCluster".

Add Host

You can add a host by conducting the same steps in First Host Install. After the host is started and configured, it will automatically connect to the existing host group. After the host has connected, the Start-CacheHost command with the -HostAddress parameter will join the target host to the SOSS store. Alternatively, the Start-CacheCluster command will join all inactive hosts to the SOSS store. Example: "Start-CacheHost -HostAddress 10.0.4.20" or "Start-CacheCluster".

Remove Host

A host can be removed from the active store membership by issuing the Stop-CacheHost command with the -HostAddress parameter. Once the host is inactive, the service can be uninstalled to fully remove the host. For example, "Stop-CacheHost -HostAddress 10.0.4.19".

Start Cluster

If the SOSS store was restarted and multiple hosts need to join the store, the Start-CacheCluster command can be used to join all of them to the SOSS store. This command will cause all hosts to handle a portion of the application’s workload.

Stop Cluster

The Stop-CacheCluster command can be used to leave all inactive hosts from the ScaleOut StateServer store. This will stop the SOSS store from handling any user requests. For example, if the physical machines where the SOSS store is deployed need to be upgraded, the Stop-CacheCluster command can be used to make all hosts inactive. Once the hosts are inactive, any maintenance actions can take place.

Stop Host

If a host needs to be temporarily stopped for maintenance, the Stop-Cachehost command with the -HostAddress parameter can be used to leave the host from active host group. Once the host is inactive, the SOSS service process on that host can be stopped and any maintenance actions can take place.

Remove Cluster

The Stop-CacheCluster command will leave all active hosts from the ScaleOut StateServer store. After the SOSS store is inactive, the SOSS service on each host can be uninstalled.

Get Statistics

To review the overall store or specific host workload, use the Get-CacheStatistics to display workload statistics for the ScaleOut StateServer store. The -HostAddress parameter can optionally be specified to display workload statistics for a specific host. For example, "Get-CacheStatistics" or "Get-CacheStatistics -HostAddress 10.0.4.19".

Verify Cluster is Active

An administator can verify that the SOSS cluster is active by using the Get-CacheHost command. The Get-CacheHost command will display all of the hosts in the SOSS store and their respective status. The -HostAddress parameter can optional be specified to target a specific host. For example, "Get-CacheHost" or "Get-CacheHost -HostAddress 10.0.4.19".

4.3. Client Configuration

Windows Server AppFabric Caching

WSAF cache clients can be configured to connect to a cache either through an XML configuration file or programmatically through the API. In both approaches, the locations of lead hosts should be provided to the AppFabric client libraries.

ScaleOut StateServer

Connection information for remote client applications is configured through the ScaleOut Management Console application on each client machine. This connection information is stored on the client in the `soss_client_params.txt` file (located in the ScaleOut installation directory), which is shared by all client applications on that system.

More information about configuring remote client machines is available in the Remote Client Configuration topic of the *ScaleOut StateServer Help* online guide. Clients can also be configured on the command line by using the soss.exe command line application or by directly editing the `soss_client_params.txt` file.

Tip

ScaleOut StateServer also provides Java, C, C++, and REST APIs.

4.4. High Availability

Windows Server AppFabric Caching

When enabled, high availability of WSAF Caching maintains a "primary" copy of each object as well as a "secondary" copy of each object on a remote cache host for fault tolerance. This feature is disabled by default, and it is limited to use on Enterprise Editions of Windows Server. (Because there is no Enterprise SKU for Windows Server 2012, WSAF high availability was recently made available to Standard Editions of Windows Server 2012.)

ScaleOut StateServer

ScaleOut hosts always store objects in a highly available manner—the primary copy of an object is called a "master" and the secondary backups are referred to as "replicas". Management and configuration of replicas is automatic and transparent.

Replicas are kept fully consistent with master objects, and the service can be configured to store 1 or 2 replicas of each object for fault tolerance; using two replicas allows your data grid to handle the sudden loss of two ScaleOut hosts without data loss.

More information about fault tolerance and recovery from failures is available in the Recovery from Failures topic in the *ScaleOut StateServer Help* online guide.

4.5. Regions

Windows Server AppFabric Caching

Regions in WSAF Caching exist to allow a set objects to be searched by tag. This search functionality requires all objects in a region to be stored on a single cache host, which is a potential limit on scalability.

ScaleOut StateServer

Searching by tag is supported by ScaleOut StateServer, but objects do not need to be grouped together on the same ScaleOut host if you want to search for them by tag—ScaleOut StateServer is able to perform a distributed query across all objects on all hosts in the data grid. Regions, therefore, are neither needed nor supported.

Tip

In addition to tags, ScaleOut StateServer is able to automatically index properties on your .NET objects. Once indexed, you can query against your object properties using ScaleOut's powerful LINQ provider.

4.6. Local Cache

Windows Server AppFabric Caching

WSAF Caching can maintain an in-process "local cache" in your cache client application to minimize deserialization and network overhead. Once an object is in the local cache, the AppFabric APIs will use that local copy exclusively until it is invalidated (either through a timeout or a notification). Objects in the local cache are therefore only loosely coherent with their master copies in the AppFabric cache hosts, so it is possible for the DataCache class to return stale objects to your application.

ScaleOut StateServer

ScaleOut StateServer's local cache is called the "client cache". Like AppFabric's local cache, ScaleOut maintains it as in-process collection in order to minimize deserialization and network costs. However, the SOSS client cache is fully coherent and synchronized with the authoritative server—with every access, the client cache performs a quick version check with the ScaleOut host to ensure that it is returning the most recent version of an object to the caller. This provides several benefits:

- Objects returned by ScaleOut APIs are never stale.
- Applications can safely employ distributed, sequential logic (that is, when you change a cached object, all other clients in the farm will see the change immediately).
- Locking operations can be used in conjunction with the client cache.
- The ScaleOut ASP.NET session provider can safely use the client cache in all page requests, including those that use read/write session state.

4.7. Concurrency Models

Windows Server AppFabric Caching

WSAF Caching provides both optimistic and pessimistic locking models.

ScaleOut StateServer

ScaleOut APIs also provide both optimistic and pessimistic locking models. There are several key usage differences that are covered in more detail in the Migration section's **Locking** topic:

- Instead of throwing an exception in the event of a pessimistic lock collision, the ScaleOut NamedCache API can optionally block the caller and perform retries on your behalf if a collision occurs.
- Pessimistic lock handles are managed transparently by the NamedCache API and do not need to be managed by the client application.
- Optimistic lock version information is stored and automatically updated directly in your objects when the `IOptimisticConcurrencyVersionHolder` interface is implemented in your classes.

4.8. Expiration and Eviction

Windows Server AppFabric Caching

Object stored in WSAF Caching can be automatically removed from a cache through an expiration (due to a timeout) or eviction (due to memory consumption). Evictions are based on an LRU (least recently used) policy so that the most active objects are retained.

ScaleOut StateServer

SOSS also provides expirations and evictions. Timeouts can be either absolute or sliding. Evictions, like WSAF, use an LRU algorithm when removing objects. Also, objects can be given priority with respect to eviction, allowing high-priority objects to remain in the store in low-memory scenarios.

4.9. Notifications

Windows Server AppFabric Caching

WSAF Caching can be configured to provide asynchronous notifications to cache clients when changes occur at the cache, region, or individual item level. Internally, cache hosts maintain notification queues, and WSAF client libraries periodically poll the hosts (every 300 seconds, by default) to learn about changes that need to be raised as notifications in your application code.

ScaleOut StateServer

Events in ScaleOut StateServer differ significantly from WSAF Caching. ScaleOut StateServer's event feature can notify your application whenever an object is being automatically removed by the service. Events are raised under the following circumstances:

- An object is about to expire due to a timeout.
- An object is about to be evicted due to a lack of available memory.
- An object is about to be removed because it is dependent on another object, and that parent object is being removed/updated.

A common use case for expiration events is to use them to save an expiring object to a more persistent storage medium, such as a database. Event handlers are also given the opportunity to cancel the removal so that the object will remain in the data grid.

Another significant difference is that event delivery is load balanced among instances of your client application. ScaleOut hosts push events to clients rather than requiring clients to poll for them, and the service will retry delivery if a client application is temporarily unavailable. This results in a stronger guarantee of event delivery and a more scalable event distribution model.

ScaleOut StateServer does not directly support delivering events to all clients for all Add/Replace/Remove events. (A primary use case for these notifications in WSAF Caching was to keep AppFabic's local caches loosely synchronized with the authoritative service—ScaleOut's fully synchronized client cache doesn't require this type of infrastructure.) ScaleOut's backing store integration can be used for asynchronous notification of Add/Replace/Remove activity; please contact ScaleOut's support team if you would like to discuss your requirement in more detail.

Tip

ScaleOut's event feature can raise your ASP.NET application's Session_End event if you use ScaleOut's ASP.NET session state provider.

5. Using the ASP.NET Session State Provider

5.1. Configuration

ScaleOut's ASP.NET session provider is a custom, out-of-process provider that can be enabled by simply modifying your application's `web.config` file.

Like WSAF Caching's session provider, your `<sessionState>` element's `mode` attribute must be set to `"Custom"`. Simply add the ScaleOut provider to the <providers> element as illustrated below and then change the `customProvider` attribute to point to the new `"SossStoreProvider"`:

```
<sessionState
  mode="Custom"
  customProvider="SossStoreProvider"
  cookieless="UseCookies"
  timeout=60
>
  <providers>
    <add
      name="SossStoreProvider"
      type="Soss.Web.SossStoreProvider, soss_storeprovider, Version=5.0.0.0,
 Culture=neutral, PublicKeyToken=a1ec0b86f746a476"
    />
  </providers>
</sessionState>
```

The standard attributes of the `<sessionState>` element are supported by the SOSS provider. For example, the `timeout` attribute controls the lifetime of the session object in the SOSS store.

5.2. Sharing Session Data Between Web Apps

By default, ScaleOut's ASP.NET session provider does not share session objects between multiple ASP.NET applications. If session data needs to be shared across two or more applications, the `soss_SharedAppName` setting can be added to each application's `web.config` file. This setting specifies a common application namespace that ScaleOut StateServer will use for storing ASP.NET session objects, and it overrides ScaleOut StateServer's default use of the Web application's name for this purpose. For example, all ASP.NET applications that need to share session objects in the namespace called MyName should add the following lines to their web.config's <appSettings> section:

```
<configuration>
  <appSettings>
    <add key="soss_SharedAppName" value="MyName"/>
  </appSettings>
</configuration>
```

5.3. The Session_End Event

ScaleOut StateServer's session provider will automatically register for expiration events in order to fire a web application's Session_End event in Global.asax. To prevent this registration and stop events from being delivered to Session_End, add the setting soss_RegisterForEvents with the value False to the application's web.config file, as follows:

```
<configuration>
  <appSettings>
    <add key="soss_RegisterForEvents" value="False"/>
  </appSettings>
...
</configuration>
```

6. Using the ASP.NET Output Cache Provider

6.1. Configuration

ScaleOut's ASP.NET output cache provider is a custom, out-of-process provider that can be enabled by modifying your application's `web.config` file. This provider can be used in web applications that target the .NET 4.0 runtime and higher.

Like WSAF Caching's output cache provider, you enable this custom provider by adding it under your `<outputCache>` element's `<providers>` section. Once added, simply change the `defaultProvider` attribute to point to the new `"SossOutputCacheProvider"`:

```
<caching>
  <outputCache defaultProvider="SossOutputCacheProvider">
    <providers>
      <add name="SossOutputCacheProvider"
           type="Soss.Web.SossOutputCacheProvider, soss_storeprovider,
 Version=5.0.0.0, Culture=neutral, PublicKeyToken=a1ec0b86f746a476"
           throwOnError="false"
           sossAccessTimeoutMilliseconds="250" />
    </providers>
  </outputCache>
</caching>
```

In addition to the standard `name` and `type` attributes, the provider supports several custom attributes that give you fine-grained control over how the provider should behave in situations when you application is unable to communicate with the ScaleOut data grid:

`throwOnError`

If an error occurs when accessing the ScaleOut StateServer service (for example, if a network problem prevents a web server from accessing the ScaleOut data grid), the `throwOnError` attribute controls whether exceptions are thrown from the provider—if exceptions are suppressed, the provider will inform ASP.NET pipeline that the requested output cache entry does not exist. The resulting "cache miss" will cause ASP.NET to execute the full request handler instead.

`sossAccessTimeoutMilliseconds`

The `sossAccessTimeoutMilliseconds` attribute specifies the maximum time (in milliseconds) that the Output Cache Provider will wait to receive a response from the ScaleOut data grid. If the timeout is elapsed, the provider will simply inform the ASP.NET pipeline that the output cache entry does not exist, and the resulting "cache miss" will cause ASP.NET to execute the full request handler instead. A value of `0` will cause the provider to wait indefinitely, causing the web request to be always wait until a result is returned.

More information about output cache configuration is available in the ASP.NET Output Cache Provider topic of the *ScaleOut StateServer Help* online guide.

6.2. Instrumentation

The output cache provider offers performance counters under the *ScaleOut Output Cache Provider* counter group in Perfmon. These can be used to track the health and effectiveness of the ScaleOut Output Cache provider. If detailed trace information is needed, the provider implements a standard .NET trace source that can log diagnostic information to the trace listener of your choosing.

More information about ASP.NET output cache instrumentation is available in the ASP.NET Output Cache Provider topic of the *ScaleOut StateServer Help* online guide.

7. Migrating a Cache Client

The topics in this section highlight the main features in the Windows Server AppFabric (WSAF) Caching APIs and demonstrates their equivalent calls in the ScaleOut StateServer NamedCache APIs.

7.1. Core Classes in the ScaleOut API

The `Soss.Client.NamedCache` class is the primary way to interact with your objects that are stored in a ScaleOut in-memory data grid. Like the AppFabric `DataCache` class, a `NamedCache` instance represents a named collection of objects in the distributed data store, and it allows you to work with your objects using familiar .NET collection syntax.

A NamedCache instance is accessed through the `Soss.Client.CacheFactory` class as follows:

```
NamedCache cache = CacheFactory.GetCache("My Cache");
cache["Welcome"] = "Hello World!";
```

The `CacheFactory` returns the same `NamedCache` instance for each unique string identifier that is passed into the `GetCache` method. In other words, your application is handed a singleton for each named cache. This allows you to set your caching options once at application startup and then continue to use those same options throughout the life of your client application.

Tip

It does not matter if you use repeatedly use the CacheFactory to acquire a `NamedCache` instance or have your application re-use a single reference to a `NamedCache` class instance—the same instance will be used in either case. The `NamedCache` is thread safe, so the same instance can be used to access objects concurrently from multiple threads.

Unlike AppFabric Caching, a ScaleOut client application is free to use the `CacheFactory.GetCache` method to create as many named caches as it needs on the fly at runtime—the ScaleOut APIs do not require each named cache to be specified in a configuration file, nor do named caches need be explicitly created through ScaleOut management tools.

A number of advanced caching features are available to your application, both at the cache level using the `NamedCache` class and at the individual object level using the `CreatePolicy` class.

In addition to the migration guidance laid out in the following topics, you can learn more about the API's features in the Conceptual Overview of the *.NET API Reference*.

7.2. Basic Create/Read/Update/Delete Operations

For basic cache operations that do not use locking, expirations, or any other advanced features, AppFabric `DataCache` calls can be replaced with the following ScaleOut `NamedCache` calls:

AppFabric DataCache method	**ScaleOut NamedCache method**
DataCache.Add	NamedCache.Insert
DataCache.Get	NamedCache.Retrieve
DataCache.Put	NamedCache.Insert
DataCache.Remove	NamedCache.Remove

The `NamedCache.Insert` method's *updateIfExists* parameter allows the method to behave either like a `DataCache.Add` or a `DataCache.Put` call:

```
// "Put" an object that may or may not already exist in the SOSS service:
NamedCache cache = CacheFactory.GetCache("Sample Cache");
cache.Insert("My key", "This is a value in the cache",
             cache.DefaultCreatePolicy, true, false);
```

7.3. Timeouts

AppFabric's `DataCache` class allows you to set the amount of time that the object will reside in the cache before expiration using either the `Add` or `Put` methods.

ScaleOut StateServer allows you to set timeouts (and many other cache policies) on an object-by-object basis using the `CreatePolicy` class when inserting an object into the cache.

```
NamedCache cache = CacheFactory.GetCache("Sample Cache");

// Insert an object with a 20 minute timeout:
CreatePolicy policy = new CreatePolicy();
policy.Timeout = TimeSpan.FromMinutes(20);
policy.IsAbsoluteTimeout = true;
cache.Insert("Key2", "Object value with 20 min timeout",
             policy, true, false);
```

The `IsAbsoluteTimeout` property specifies whether the timeout is *absolute* (that is, the object expires after the elapsed time regardless of whether it's accessed) or *sliding* (the timeout resets back to its original timeout every time it is accessed).

7.4. Locking

Like AppFabric, ScaleOut StateServer supports both pessimistic and optimistic locking strategies. Key differences between the two products' APIs are discussed in this topic.

Pessimistic Locking

AppFabric DataCache method	ScaleOut NamedCache method
GetAndLock	Retrieve (set *acquireLock* parameter to `true`)
PutAndUnlock	Update (set *unlockAfterUpdate* parameter to `true`)
Unlock	ReleaseLock

Lock Handles

WSAF Caching returns a `DataCacheLockHandle` instance from a locking `GetAndLock()` call. This handle instance must then be supplied to one of the DataCache's unlock calls (`PutAndUnlock()`/`Unlock()`) to release the lock on the object.

In contrast, ScaleOut StateServer's NamedCache API automatically manages lock handles on your behalf. Handles to locks that you acquire are stored behind the scenes in thread-local storage, so your application does not need to store and manage any lock handles.

Handling Lock Collisions

In WSAF Caching, if a caller attempts to lock an object that has already been locked by another client/thread then a `DataCacheException` will be thrown. AppFabric client applications must implement their own retry logic.

ScaleOut StateServer's NamedCache API will block and automatically perform retries if you attempt to lock an object that has already been locked by another caller. Your locking call will return once your thread successfully acquires the lock.

The NamedCache's MaxLockAttempts and LockRetryInterval properties control the number and frequency of lock retry attempts. By default, the NamedCache will attempt 20000 retries every 5 milliseconds. If these retries are exhausted then an ObjectLockedException will be thrown.

Tip

Setting the MaxLockAttempts count to 1 will make the NamedCache behave like the AppFabric DataCache—if lock acquisition fails on the first attempt then the locking call will return immediately and an ObjectLockedException will be thrown if there is a lock collision.

Lock Timeouts

In WSAF Caching, the `DataCache.GetAndLock` method allows you specify a lock timeout argument, which causes a lock to be released if a programming error or application crash causes a client application to leave an object locked in the cache.

ScaleOut StateServer offers a configurable lock timeout that is applied to the entire data grid. The default timeout is 90 seconds (this default was chosen to match ASP.NET's default session lock timeout). The value can be adjusted by modifying the `lock_timeout` configuration setting in the service's `soss_params.txt` file.

If locking is being used, you are encouraged to call `NamedCache.ReleaseLock` in a `finally` block to reduce the likelihood of leaving an object locked in the ScaleOut data grid. The `ReleaseLock` method has minimal overhead and will only make a round-trip to a ScaleOut host if it knows that the current thread is still holding a lock on the object. For example:

```
NamedCache cache = CacheFactory.GetCache("Locking sample cache");

try
{
  // Read and lock:
  cache.Retrieve("key1", true);

  // Exception-prone code:
  int zero = 0;
  int mistake = 42 / zero;

  // Update and unlock:
  cache.Update("key1", mistake, true);
}
catch (DivideByZeroException)
{
  // handle error...
}
finally
{
  // Ensure object is always unlocked.
  if (cache != null) cache.ReleaseLock("key1");
}
```

Synchronizing Object Creation

"Forcing" Locks on Non-existent Objects

Several of the AppFabric `GetAndLock()` overloads provide a *forceLock* boolean parameter. Contrary to the behavior suggested by this parameter name, it does not allow a caller to force an override of a lock that is held by another client/thread. Instead, this parameter allows you to lock an object even if it does not yet exist in the AppFabric cache.

The intended usage patterns for AppFabric's *forceLock* parameter are not fully explained in the MSDN documentation, but a common use case for such a feature is to provide a way to synchronize creation of an object in the cache. That is, if multiple AppFabric clients attempt to access an object and encounter a cache miss, the *forceLock* parameter can be used to prevent all of the clients from simultaneously attempting to retrieve the missing object from a system of record (a potentially expensive operation) and then overwriting each other's objects in the cache.

AppFabric clients must use locking calls to perform this synchronization, and clients must be written to perform retries or otherwise gracefully handle exceptions that are thrown by lock collisions.

ScaleOut's CreateHandler Callback

ScaleOut's NamedCache API does not require the use of error-prone locking to synchronize and coordinate the creation of objects in the distributed cache. When calling the NamedCache.Retrieve() method, a caller can simply provide a delegate to a callback method (called a "CreateHandler") that will be invoked if a cache miss occurs. This user-supplied callback is responsible for providing an object instance that will be automatically inserted into the cache.

If multiple clients/threads simultaneously attempt to read the missing object, only one thread in one client will be permitted to execute the CreateHandler callback so as to prevent multiple clients/thread from simultaneously creating the object—this behavior is valuable when creating the cached object involves expensive calls to a database or when it is otherwise undesirable for an object to be repeatedly created in the cache. While the object is being created, other threads that try to retrieve the object while the callback is executing will be blocked, even if they're running on other client machines. Once the object has been added to the data grid, those other threads will be unblocked and the newly-stored object will be returned to all of them.

The synchronization performed by ScaleOut's APIs is transparent to the caller—a client only needs to provide a CreateHandler callback to a NamedCache `Retrieve()` call.

Tip

A full example of how to efficiently synchronize the creation of expensive objects is available in the "RetrieveOptions.CreateHandler") reference documentation.

Optimistic Locking

A number of WSAF Caching methods return a `DataCacheItemVersion` instance that can be used to implement optimistic concurrency in your application. If there is a version mismatch when performing a `Put` call, the `Put` method returns `null` and your application may choose to re-retrieve the object and retry the update.

ScaleOut's approach to optimistic concurrency is similar to AppFabric's, but the object's version is not returned separately from the object. Instead, users implement the `IOptimisticConcurrencyVersionHolder` interface on objects that need to implement optimistic concurrency control. Version information is essentially stored *in* the object, and, if a mismatch is detected when updating an object in the cache, an `OptimisticLockException` is thrown.

An optimistic update can be performed by using the Update method. This overload takes an `UpdateOptions` struct as a parameter, whose LockingMode property allows you to specify an `UpdateLockingMode` enum. This enum controls whether your update operation performs optimistic locking, pessimistic locking, or no locking at all.

If an `OptimisticLockException` is thrown from an update operation, your application may choose to either discard the update or else refresh its local copy of the object and retry the update. Because this retry logic can be expensive (depending on the size and complexity of your object and the changes you are making to it), optimistic locking is best used in situations where updates to the object are infrequent relative to reads.

Tip

Resolving collisions can be the trickiest part of implementing an optimistic concurrency strategy. ScaleOut's *.NET API Reference* has a helpful topic on Optimistic Concurrency that has sample code illustrating best practices for performing updates and handling collisions.

7.5. Regions

Regions in WSAF Caching exist to allow a set objects to be searched by tag. This search functionality requires all objects to be stored on a single cache host, and regions are used to keeps these objects together.

Objects do not need to be grouped together on the same ScaleOut StateServer host if you want to search for them by tag—ScaleOut StateServer is able to perform a distributed query across all objects in all hosts that are part of the distributed data grid. Regions, therefore, are neither needed nor supported—there are no API calls that require a region to be specified.

7.6. Tags

The AppFabric `DataCache` class allows user-defined tags to be associated with an object using the `Add` or `Put` methods. Tags allow groups of objects to be selected and removed together instead of accessing them through their individual keys.

In the ScaleOut NamedCache API, tags are stored on your objects themselves rather than specified separately through API calls. Classes that need to support tags must implement the `ITaggable` interface, after which a set of extension methods (`AddTags` and `RemoveTags`) can be used to specify which tags are associated with an object.

Once an object is tagged, ScaleOut's LINQ provider allows you to query for tagged objects—the service supports HasAllTags, HasAnyTag, and HasTags methods in LINQ where clauses.

ScaleOut's *.NET API Reference* contains a Support for Tags topic that has helpful code samples and a conceptual overview of how to tag your objects. The Querying Tags topic illustrates how you can query for tagged objects using ScaleOut's LINQ provider.

Moving Beyond Tags with Indexing and Queries

ScaleOut StateServer offers indexing and query support that is much richer than simple tags. Marking your class properties with a `SossIndex` attribute allows the service to index objects using values in that property, which makes those objects eligible for selection using ScaleOut's LINQ provider.

While full coverage of the LINQ provider is outside of the scope of this document, the *.NET API Reference* covers indexing and LINQ in great depth starting with its LINQ Overview topic.

7.7. Local Cache

AppFabric's local cache feature is disabled by default because of its implications on the behavior of your application. The local cache is not fully coherent with the authoritative cache hosts—it is only synchronized periodically, so it is possible for an AppFabric cache client to receive stale data.

ScaleOut's local "client cache" is enabled by default for all named caches. The client cache is fully coherent with the distributed cache hosts and will not return stale data. This makes the client cache suitable for scenarios where objects are frequently updated and those updates must be sequentially consistent (and immediately visible) to all instances of your client application.

ScaleOut's client cache achieves this consistency by performing an inexpensive version check with the authoritative service every time an object is accessed—if the client cache confirms that the latest version of your object is already available in your client process, the APIs will not pull the entire object over the network and will also avoid any deserialization overhead.

ScaleOut's client cache can be disabled using the `NamedCache.AllowClientCaching` property. The reference documentation for this property also has information and guidelines for its usage.

7.8. Bulk Operations

ScaleOut StateServer's APIs support a different approach to bulk operations when compared to AppFabric with its `DataCache.BulkGet()` method. Rather than using a single client to retrieve and evaluate a large set of objects, SOSS allows you to employ powerful, distributed strategies to efficiently work with large datasets while minimizing network overhead. Distributed LINQ query support and ScaleOut's in-memory compute engine allow you to spread bulk operations across the entire server farm.

Common use cases that are addressed by the APIs include:

- *Distributed queries:* If you are only retrieving a set of objects so that you can perform a deeper inspection of properties, consider indexing your objects and using ScaleOut's LINQ query provider instead. This approach allows you to use the service's built-in support for distributed queries, effectively using all the hosts in your distributed data grid to filter your objects instead of just one client.

- *Distributed analysis/updates:* If you need to perform analysis or updates on a large number of objects then ScaleOut's in-memory compute engine can be used. The compute engine deploys your custom analysis/update logic to the cache hosts where your objects reside, minimizing network overhead and taking advantage of data locality to improve processing time. A straightforward data-parallel programming model is provided through the `NamedCache.Invoke` method. `Invoke()` operations can be combined with ScaleOut's LINQ provider to filter the objects being analyzed.

- *Minimizing "wall clock time":* If you are simply trying to retrieve a small set of objects in the least amount of time, using the .NET Framework's Task Parallel Library is a very effective approach (for example, use a `Parallel.ForEach()` loop to gather a set of objects). ScaleOut's client libraries work well in multithreaded contexts—accesses can be performed concurrently and are load balanced across farm's caching hosts. Parallel operations also work well for bulk insert scenarios where a large number of objects need to be loaded by a single client.

Note

Using the `NamedCache.Invoke` method to perform distributed operations with the in-memory compute engine requires a ScaleOut ComputeServer™ license.

7.9. Expiration Events

ScaleOut client applications can subscribe to the `NamedCache.ObjectExpired` event to receive notifications when objects are about to be automatically removed from the distributed data grid. The expiration event will be fired for one of three reasons:

- An object is about to expire due to a timeout.
- An object is about to be evicted due to a lack of available memory.
- An object is about to be removed because it is dependent on another object, and that parent object is being removed/updated.

The reason for the event expiration is available in the `NamedCacheObjExpiredEventArgs` instance that is provided to your event handler. This event argument contains other useful properties such as the key of the object and gives you the option to cancel the expiration so that the object remains in the store.

In the following example, an expiration event handler is registered with a named cache, which uses the provided event arguments to print out the reason for an object's removal:

```
using System;
using System.Threading;
using Soss.Client;

class Program
{
  static void Main(string[] args)
  {
    NamedCache cache = CacheFactory.GetCache("Sample Cache");
    cache.ObjectExpired += cache_ObjectExpired;

    // Insert an object with a 5-second timeout:
    CreatePolicy policy = new CreatePolicy();
    policy.Timeout = TimeSpan.FromSeconds(5);
    cache.Insert("key1", "value", policy, true, false);

    // Wait for the expiration so we can see the event fire:
    Thread.Sleep(7000);
  }

  static void cache_ObjectExpired(object sender,
                                  NamedCacheObjExpiredEventArgs eventArgs)
  {
    Console.Write("'{0}' is expiring: ", eventArgs.Key.GetKeyString());
    switch (eventArgs.NamedCacheEventCode)
    {
      case NamedCacheEventCode.ObjectTimeout:
        Console.WriteLine("Object timed out.");
        break;
      case NamedCacheEventCode.LowMemory:
        Console.WriteLine("Object evicted due to low mem.");
        break;
      case NamedCacheEventCode.Dependency:
        Console.WriteLine("Object removed due to dependency relationship.");
        break;
    }
  }

}

// OUTPUT:
// 'key1' is expiring: Object timed out.
```

Expiration events are load balanced among instances of your client application, so only one client will be notified of each expiration. The SOSS service pushes events to clients rather than requiring clients to poll for them, and the service will retry delivery if a client application is temporarily unavailable. This results in a strong guarantee of event delivery and a more scalable, reliable event distribution model.

Expirations events are enabled by default in the SOSS service. By default, the service will try to deliver an event to your application one time for each expiring object—the number of retries can be increased by modifying the `max_event_tries` parameter in the service's `soss_params.txt` file. If the value is greater than `1` then the service will attempt to retry the delivery of an event once per minute. Event delivery can be completely disabled by setting the `max_event_tries` parameter to `0`.

7.10. Object Serialization

By default, ScaleOut's NamedCache API uses .NET's BinaryFormatter to perform serialization. This differs from Windows Server AppFabric Caching, which uses the NetDataContractSerializer.

ScaleOut's `NamedCache` class allows you to use custom serializers, so, if you would like to continue using the NetDataContractSerializer, you can use the `SetCustomSerialization` method to register the NetDataContractSerializer as your serializer. Registration needs to be done once for each named cache (typically when your client application first starts up). For example:

```
// Note: your project must reference System.Runtime.Serialization.dll
using System;
using Soss.Client;
using System.Runtime.Serialization;

class Program
{
  static void Main(string[] args)
  {
    NamedCache cache = CacheFactory.GetCache("Sample Cache");
    NetDataContractSerializer ndcs = new NetDataContractSerializer();

    // Register the serialize/deserialize callbacks that
    // the "Sample Cache" namespace should use:
    cache.SetCustomSerialization(ndcs.Serialize, ndcs.Deserialize);

    // Create/Read/Update/Delete objects as you normally would, and the
    // new serialization callbacks will be used:
    cache.Insert("key1", "NDCS-serialized value",
                 cache.DefaultCreatePolicy, true, false);

    // Note that all instances of "Sample Cache" that are returned by
    // CacheFactory will continue using the custom serialization methods.
  }
}
```

Tip

A custom serialization callback is a great place to perform encryption or compression of your objects. The example in the `SetCustomSerialization` documentation illustrates how you can use perform compression using a customized serializer.

Improving Serialization Performance

While .NET's "off the shelf" serializers like the NetDataContractSerializer and the BinaryFormatter are flexible and easy to use, your application's performance can be greatly enhanced by using leaner serializers such as protobuf-net or msgpack-cli.

Serialization is often one of the top performance bottlenecks in distributed applications—we recommend using a custom serializer like protobuf-net or msgpack-cli for non-trivial data types whenever possible, as they can improve serialization performance by an order of magnitude and often reduce memory usage in the ScaleOut hosts.

The following example illustrates how an application could use Marc Gravell's protobuf-net library for efficient serialization:

```
// Note: Add the protobuf-net NuGet package to project.
using System;
using System.IO;
using ProtoBuf;
using Soss.Client;

// Type of protobuf-serialized objects to store:
[ProtoContract]
public class Person
{
    [ProtoMember(1)]
    public string FirstName { get; set; }

    [ProtoMember(2)]
    public string LastName { get; set; }
}

class Program
{
    static void Main(string[] args)
    {
        NamedCache cache = CacheFactory.GetCache("Protobuf cache of Persons");

        // Register our callbacks that use the protobuf-net serializer:
        cache.SetCustomSerialization(Serialize, Deserialize);

        // Cache operations now use Protocol Buffer serialization:
        Person person = new Person() { FirstName = "George",
                                       LastName  = "Washington" };

        cache.Insert("president1", person,
                     cache.DefaultCreatePolicy, true, false);
    }

    // Serialization callback:
    static void Serialize(Stream stream, Object obj)
    {
        var person = obj as Person;
        if (person == null) throw new ArgumentException("obj not a person");

        ProtoBuf.Serializer.Serialize<Person>(stream, person);
    }

    // Deserialization callback:
    static object Deserialize(Stream stream)
    {
        return ProtoBuf.Serializer.Deserialize<Person>(stream);
    }

}
```

8. Beyond Caching

Migrating to ScaleOut StateServer opens up a number of features and possibilities beyond the basic caching support that has been outlined by this guide. For example:

- Dependencies can be defined between objects so that an update/removal of a parent object will cause all dependent children to be removed.
- Replication can be performed across datacenters using ScaleOut GeoServer for fault tolerance at a geographic level.
- ScaleOut ComputeServer's in-memory compute engine can be used to deploy and invoke custom logic on your hosts so that you can perform distributed analysis of all the object in your farm as efficiently as possible.
- ScaleOut's Object Browser management tool allows you to browse through all of the objects that are stored in your farm and view their contents in a UI that is similar to the Visual Studio debugger's Watch Window.

ScaleOut Software welcomes your feedback and would be happy to discuss any questions you have about the features offered by ScaleOut StateServer. Please send your comments and questions to the ScaleOut support team.