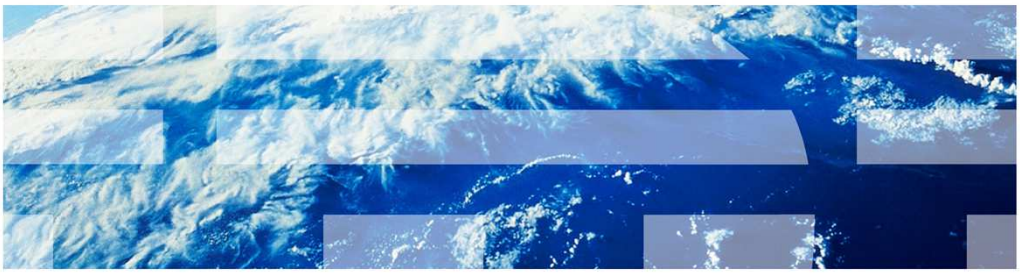
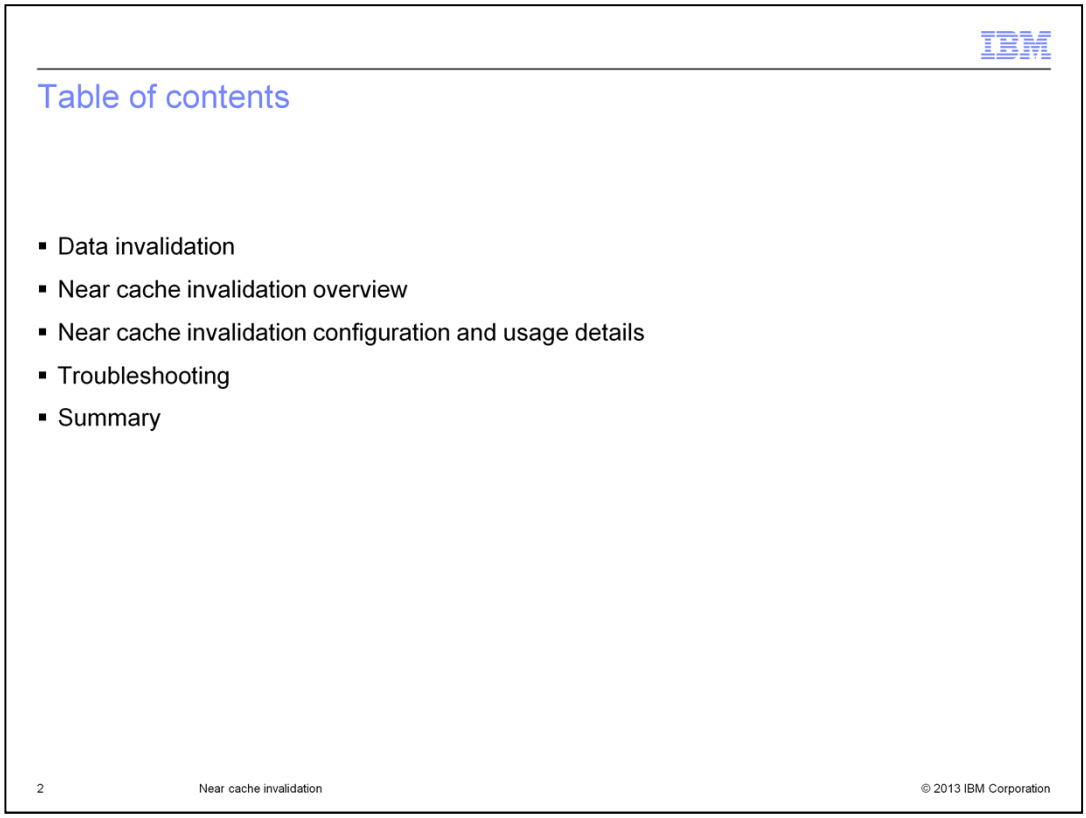
WebSphere eXtreme Scale Version 8.6

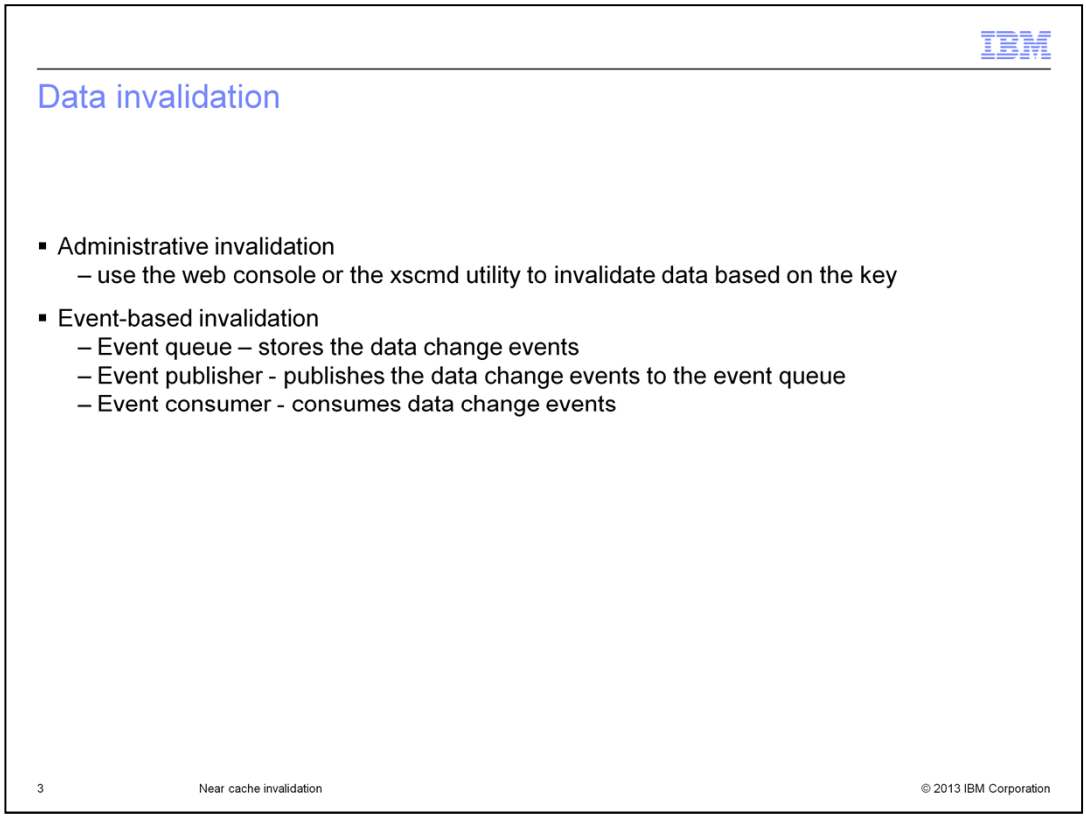
Near cache invalidation



This presentation covers the new near cache invalidation feature added in WebSphere® eXtreme Scale version 8.6. This presentation assumes you are familiar with WebSphere eXtreme Scale software or the WebSphere DataPower® XC10 appliance and have some knowledge of WebSphere eXtreme Scale configuration model and WebSphere eXtreme Scale APIs.

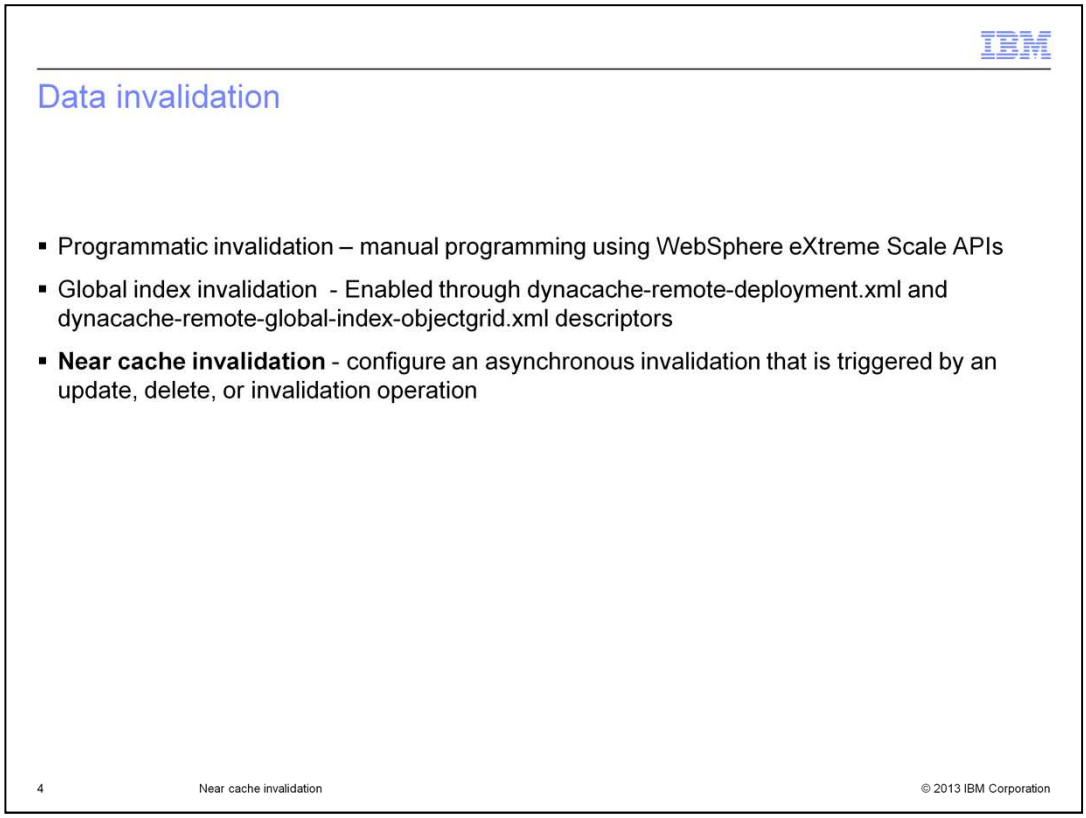


This presentation covers data invalidation in general and what mechanisms already exist for removing stale data, the new cache invalidation feature, and some troubleshooting pointers.



There are several ways to remove stale data from cache. One is administrative invalidation. You can use the web console or the xscmd utility to invalidate data based on the key. You can filter the cache data with a regular expression and then invalidate the data based on the regular expression. You can use the query interfaces in the monitoring console and in the xscmd utility to retrieve small sets of keys from a map and invalidate sets of data. If you are using the web console to query and invalidate data, configure the monitoring console first. If you are using xscmd to query and invalidate data, set up the xscmd utility.

Event based invalidation is another way of removing stale data. Sparse and complete caches can be invalidated or updated using an event generator such as Java Message Service (JMS). Invalidation using JMS can be manually tied to any process that updates the back-end using a database trigger. A JMS ObjectGridEventListener plug-in is provided in eXtreme Scale that can notify clients when the server cache changes. This type of notification decreases the amount of time that the client can see stale data. Event-based invalidation normally consists of three components; Event queue, Event publisher, and Event consumer. An Event queue stores the data change events. It can be a JMS queue, a database, an in-memory queue, or any kind of manifest as long as it can manage the data change events. An Event publisher publishes the data change events to the event queue. An event publisher is typically an application you create or an eXtreme Scale plug-in implementation. The event publisher knows when the data is changed or it changes the data itself. When a transaction commits, events are generated for the changed data and the event publisher publishes these events to the event queue. An Event consumer consumes data change events. An event consumer typically is an application to ensure the target grid data is updated with the latest change from other grids. This event consumer interacts with the event queue to get the latest data change and applies the data changes in the target grid. The event consumers can use eXtreme Scale APIs to invalidate stale data or update the grid with the latest data. For example, JMSObjectGridEventListener has an option for a client-server model, in which the event queue is a designated JMS destination. All server processes are event publishers. When a transaction commits, the server gets the data changes and publishes them to the designated JMS destination. All the client processes are event consumers. They receive the data changes from the designated JMS destination and apply the changes to the client's near cache.

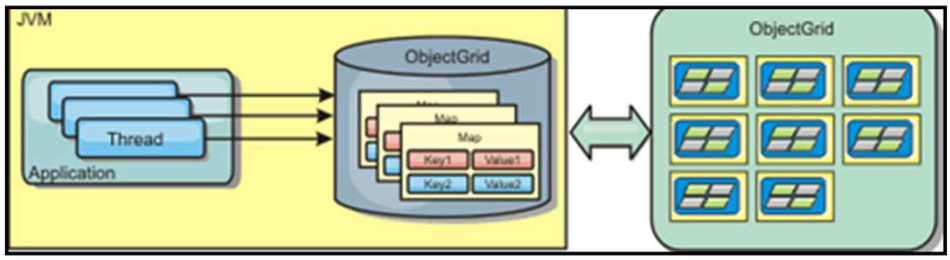


Another way to handle data invalidation is to do Programmatic invalidation. You can do manual programming using WebSphere eXtreme Scale APIs to handle stale data. You can also use programmatic invalidation with other techniques to determine when to invalidate the data. For example, this invalidation method uses event-based invalidation mechanisms to receive the data change events, and then uses APIs to invalidate the stale data.

Global index invalidation is also another way to handle data invalidation. You can optionally enable global index invalidation to improve invalidation efficiency in a large, partitioned environment; for example, more than 40 partitions. Without the global index feature, the dynamic cache template and dependency invalidation send agents to all partitions and therefore, you might experience slower performance. With global index, invalidation agents are sent only to applicable partitions that contain cache entries to invalidate and related template and dependency index data. In an environment with 100 partitions as an example, you might experience performance that is three times as fast when you remove stale cache data. Global index invalidation is enabled through dynacache-remote-deployment.xml and dynacache-remote-global-index-objectgrid.xml.

Near cache invalidation is new in WebSphere eXtreme Scale version 8.6. If you are using a near cache, you can configure an asynchronous invalidation that is triggered each time an update, delete, or invalidation operation is run against the data grid.

Near cache

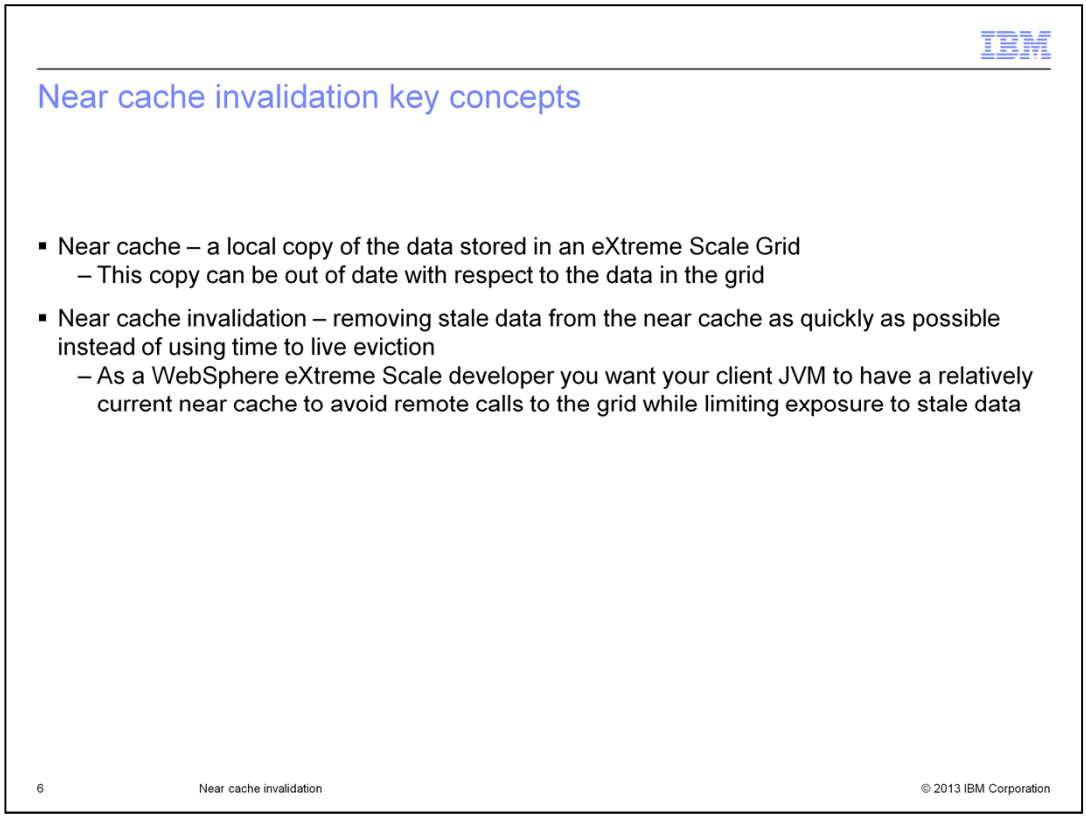


- local copy of the stored data in the eXtreme Scale Grid
- Independent ObjectGrid on each client serving as a cache for the remote, server-side cache
- Very fast – provides in-memory access to a subset of the entire cached data set that is stored remotely
- Not partitioned and contains data from any of the remote eXtreme Scale partitions
- Fast response time because all access to the data is local
- Disadvantage – stale data
  - Use when response time is important and stale data can be tolerated

5 Near cache invalidation

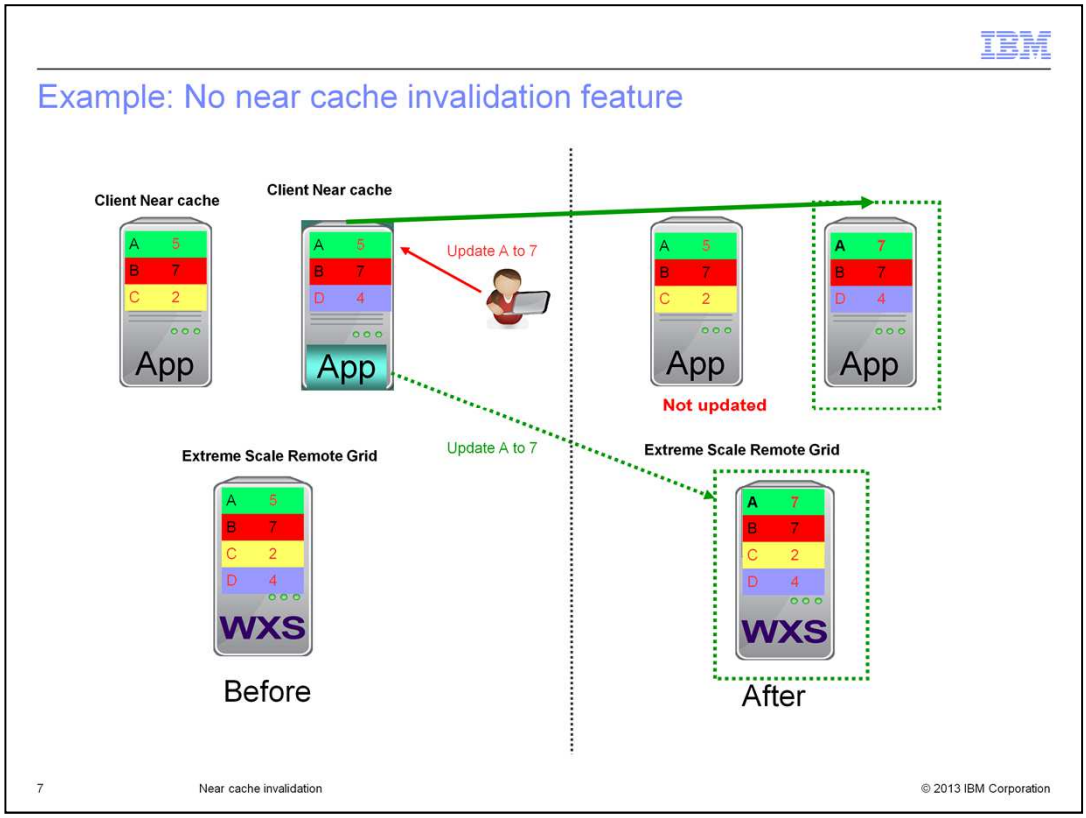
Near cache is a local copy of the stored data in the eXtreme Scale grid. It is an independent ObjectGrid® on each client, serving as a cache for the remote, server-side cache. Near cache is very fast because it provides in-memory access to a subset of the entire cached data set that is stored remotely in the eXtreme Scale servers. It is not partitioned and contains data from any of the remote eXtreme Scale partitions.

The advantage of near cache is that it gives fast response time because all access to the data is local. Looking for the data in the near cache first saves a trip to the grid of servers, thus making even the remote data locally accessible. The disadvantage of the near cache is that it increases the chance of stale data because the near cache at each tier can be out of sync with the current data in the data grid. It also relies on an evictor to invalidate data to avoid running out of memory. Near cache is recommended for use when response time is important and stale data can be tolerated.



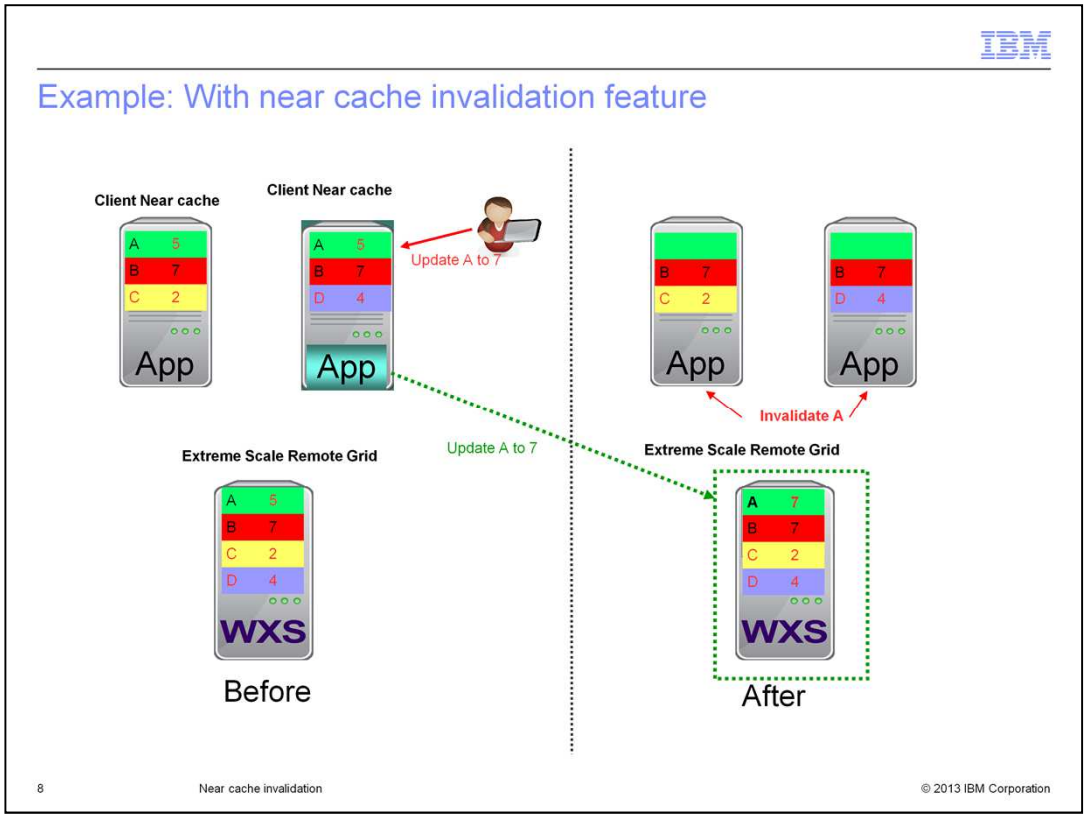
In a distributed WebSphere eXtreme Scale environment, the client side has a near cache by default when using the optimistic locking strategy or when locking is disabled. As you have seen in the previous slide, near cache is a local copy of the stored data in the eXtreme Scale grid. This data can become stale from time to time thus the need for near cache invalidation feature. Near cache invalidation feature has the ability to remove stale data from the near cache as quickly as possible instead of using TimeToLive eviction.

As a developer you want your client JVM to have a current near cache to avoid remote calls to the grid while limiting exposure to stale data. You can configure near cache invalidation to remove stale data from the near cache as quickly as possible. When an update, deletion, or invalidation operation is run against the remote data grid, an asynchronous invalidation gets triggered in the near cache. This mechanism works more quickly than the other option of using time-to-live (TTL) eviction in the near cache.

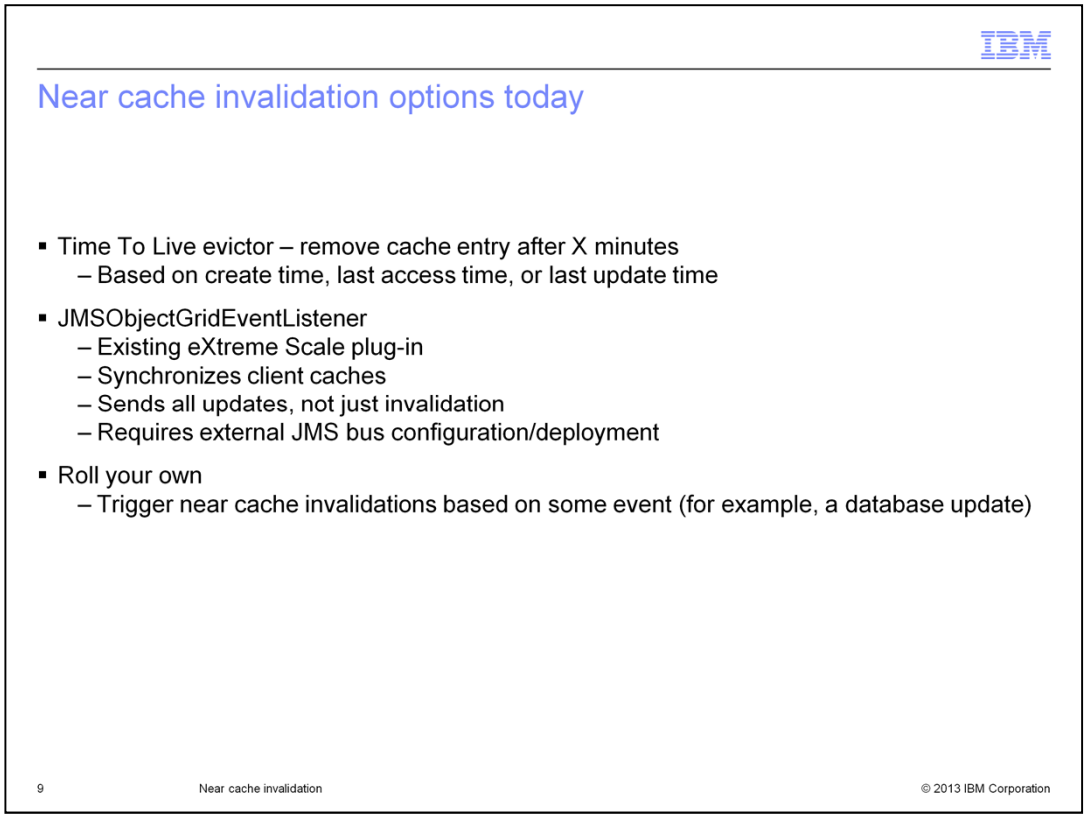


Here is an example of how data can become stale.

Starting from the left, suppose you have a grid and some clients. These clients have near caches, and so have cached a few values locally. As an example, suppose someone updates key A from 5 to 7 on one of the clients. The remote grid gets updated to 7, and the client that did the update now has a value of 7 as shown on the right. However, for the other client that had cached the key A value, the cache still has a value of 5 and is not aware of the update. This is the problem that the near cache invalidation feature solves.



With the near cache invalidation feature in place, if someone updates the key A value from 5 to 7 on one of the clients, the remote grid is updated with a value of 7 and the remote grid will issue an invalidation message to all the clients to invalidate their key A. This means that now those clients will need to go to the remote Grid to access the key A since it is no longer available in the near cache. The result is that stale data is removed from the near cache as quickly as possible.

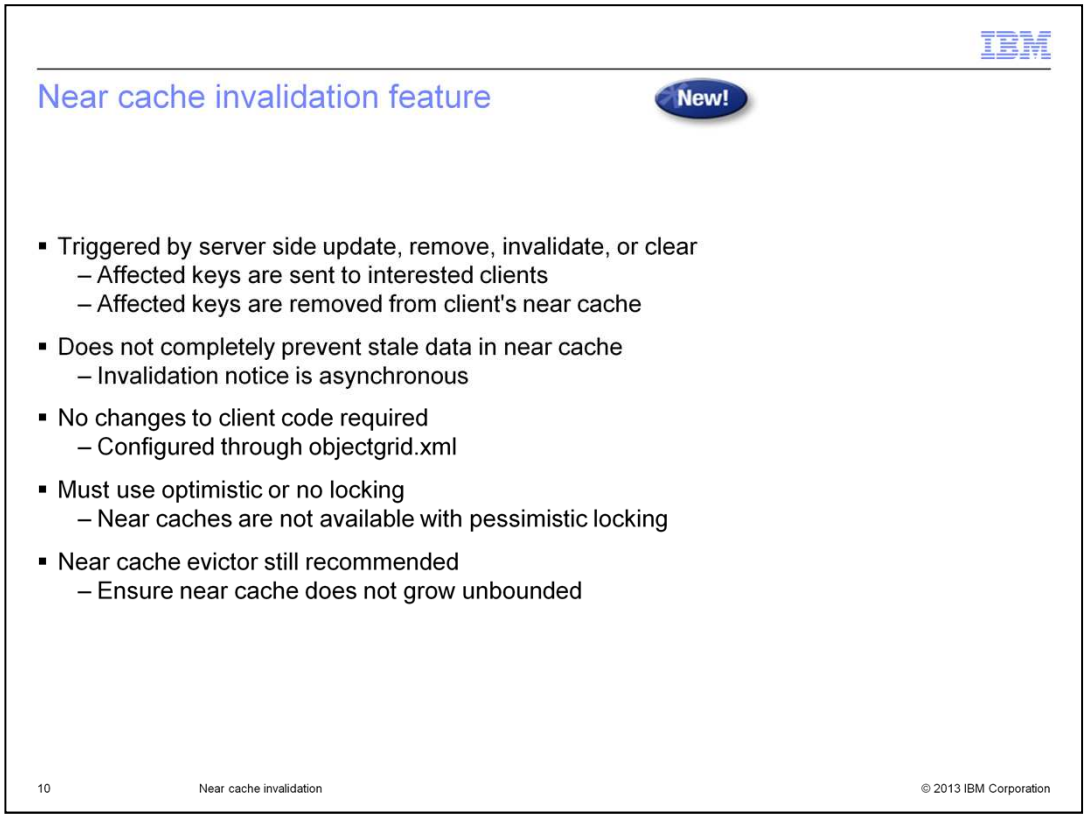


There are several cache invalidation options available. One of them is to configure the Timetolive evictor. You remove cache entries after a certain period of time based on when the cache was created, or when last accessed or updated. This way you are guaranteed the near cache entry will not hold an entry for more than five minutes.

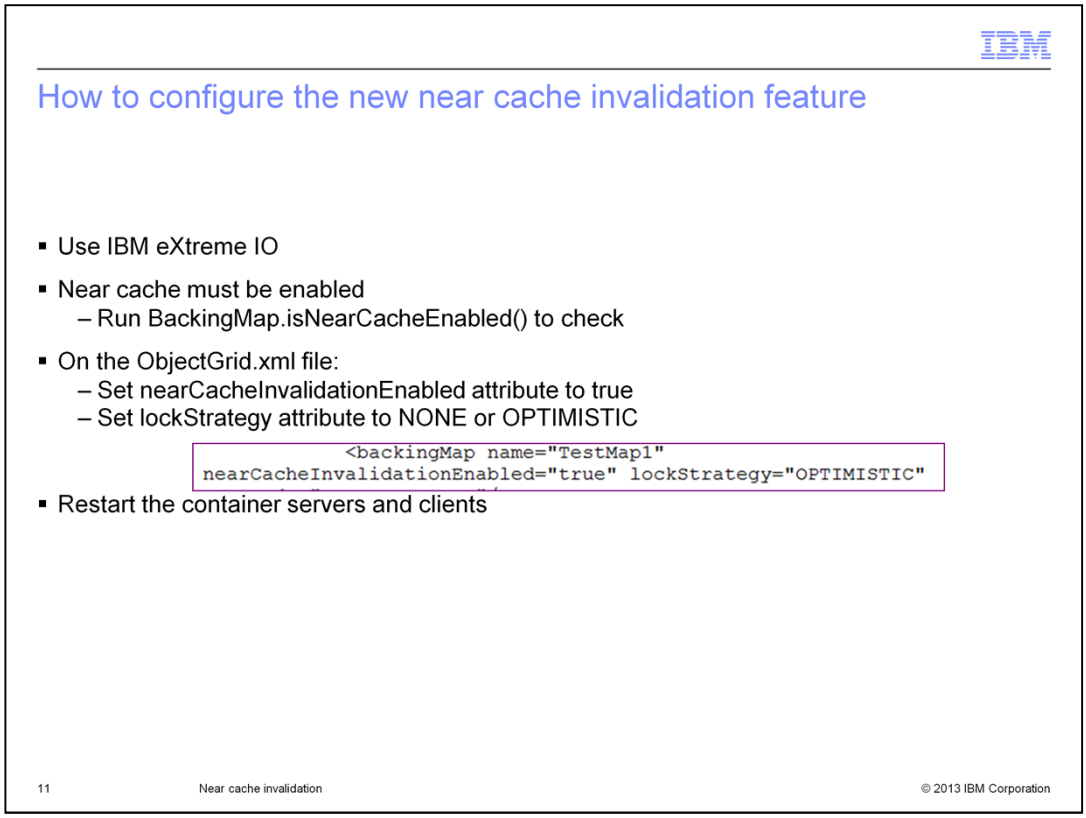
The second invalidation option is the JMSObjectGridEventListener. This is a plug-in in eXtreme scale that synchronizes client caches and sends all updates in addition to invalidation. This listener is a JMS-based implementation of ObjectGridEventListener. It is designed to support client side near cache invalidation and peer-to-peer replication mechanisms.

The client invalidation mechanism can be used in distributed ObjectGrid environment to ensure client near cache data is synchronized with servers or other clients. Without this function, the client near cache can hold stale data. However, even with this JMS-based client invalidation mechanism, you have to realize that there is a timing window for updating the client near cache. There is a delay for the ObjectGrid runtime to publish updates. The downside of this plug-in is that it is network-intensive and requires external JMS bus configuration and deployment.

The third invalidation option is to create your own near cache invalidations triggered by some event; for example, a database update.



The near cache invalidation feature allows for stale data removal from the near cache as quickly as possible. When a server side update occurs, affected keys are sent to the interested clients and the keys are removed from the clients' near cache. This helps lower the occurrence of stale data in the near cache. The feature does not require any changes to the client code instead it is configured through the objectgrid.xml. The near cache invalidation feature does not completely prevent stale data in near cache. It might take time before invalidation notice is received by clients since invalidation notice is asynchronous. The update can occur while taking some time for the clients to get the notification to remove the stale data. Also note that near caches are not available with pessimistic locking. They only use optimistic or no locking mechanism. This is because near caches work best where fast response is needed and data is not changed frequently. Another point to note is that the near cache evictor is still recommended to prevent near cache from growing unboundedly in terms of size. Use eviction policies like leastRecentlyUsed eviction policy, most recently used policy, and so on.

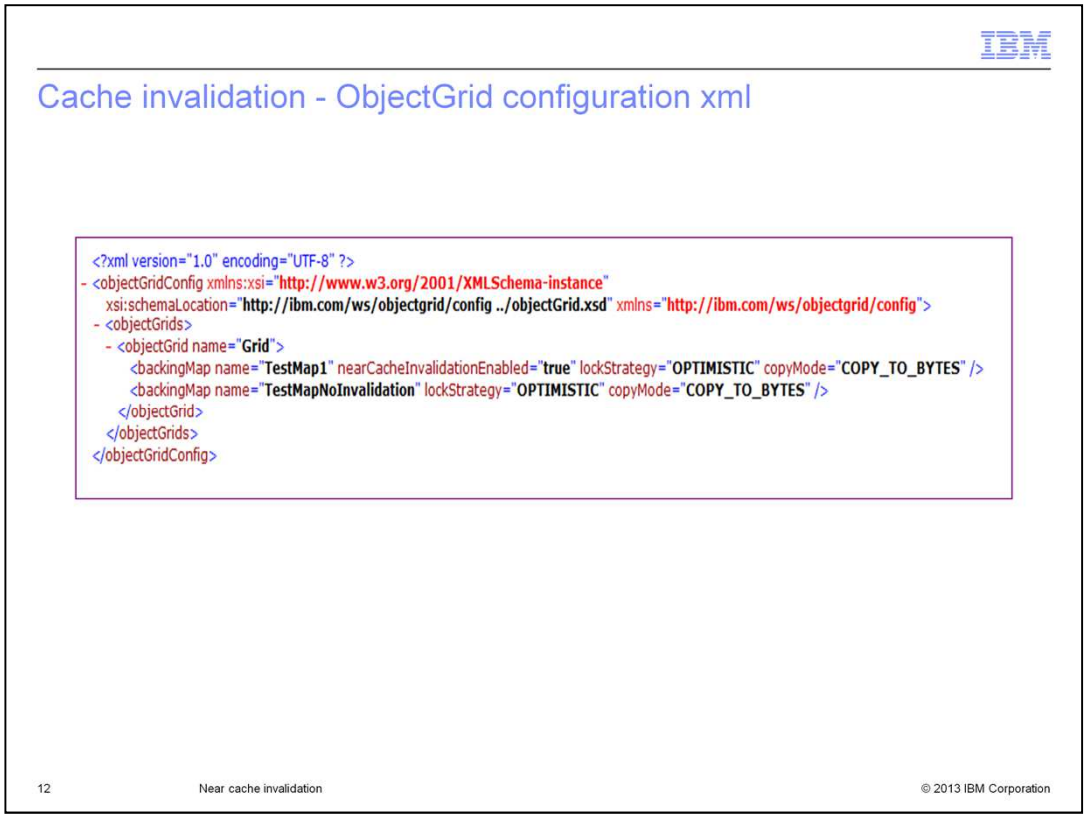


IBM eXtreme IO is a non-platform-specific network transport protocol that replaces the ORB for remote data transport. This transport protocol enables you to use clients other than Java. See the IBM WebSphere eXtreme Scale version 8.6 information center for more information on this transport. Use eXtreme IO with near cache.

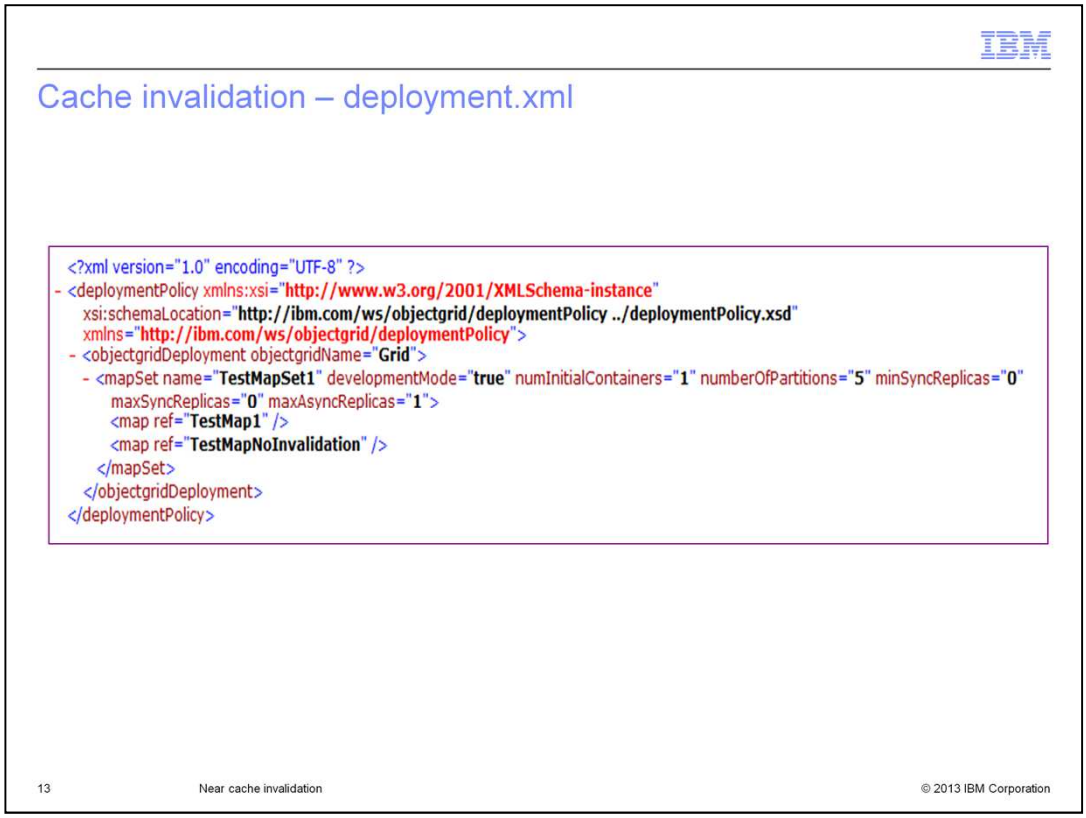
You can check to see if you are using the near cache by running the backingMap.isNearCacheEnabled method.

To enable the near cache invalidation feature you need to set two attributes in the backing map of the objectGrid.xml file. Set the nearCacheInvalidationEnabled attribute to true, and set the lockStrategy attribute to NONE or OPTIMISTIC. The settings in objectgrid.xml apply to all clients, unless you override the settings.

Save your XML file and you are ready to restart your container servers and clients.



Here is an example of an ObjectGrid xml file configured for near cache invalidation. An ObjectGrid descriptor XML file is used to define the structure of the ObjectGrid that is used by the application. It includes a list of backing map configurations. These backing maps store the cache data. Take note of the two map elements; TestMap1 that has nearcacheInvalidation enabled and TestMapNoinvalidation with no near cache invalidation. You can have some maps with this enablement and some maps without. Note the lock strategy must be optimistic.



Here is the corresponding deployment descriptor xml file. This file defines the objectgridDeployment element for the ObjectGrid. Note the number of partitions for the mapSet. The minSyncReplicas and maxSyncReplicas attributes of mapSet are to specify the minimum and maximum number of synchronous replicas for each partition in the mapSet. These are optional attributes and the default for each is 0.

Cache invalidation - example

```
UpdateClient
map1.update("key1map1", "value" + valueCounter);
map2.update("key1map2", "value" + valueCounter);
Object v1 = map1.get("key1map1");
Object v2 = map2.get("key1map2");
System.out.println("client "+clientName+" map1(invalidation) value is now: " + v1);
System.out.println("client "+clientName+" map2(no invalidation) value is now: " + v2);

output:
client updater map1(invalidation) value is now: value7
client updater map2(no invalidation) value is now: value7
client updater map1(invalidation) value is now: value8
client updater map2(no invalidation) value is now: value8
client updater map1(invalidation) value is now: value9
client updater map2(no invalidation) value is now: value9
client updater map1(invalidation) value is now: value10
client updater map2(no invalidation) value is now: value10
client updater map1(invalidation) value is now: value11
client updater map2(no invalidation) value is now: value11
```

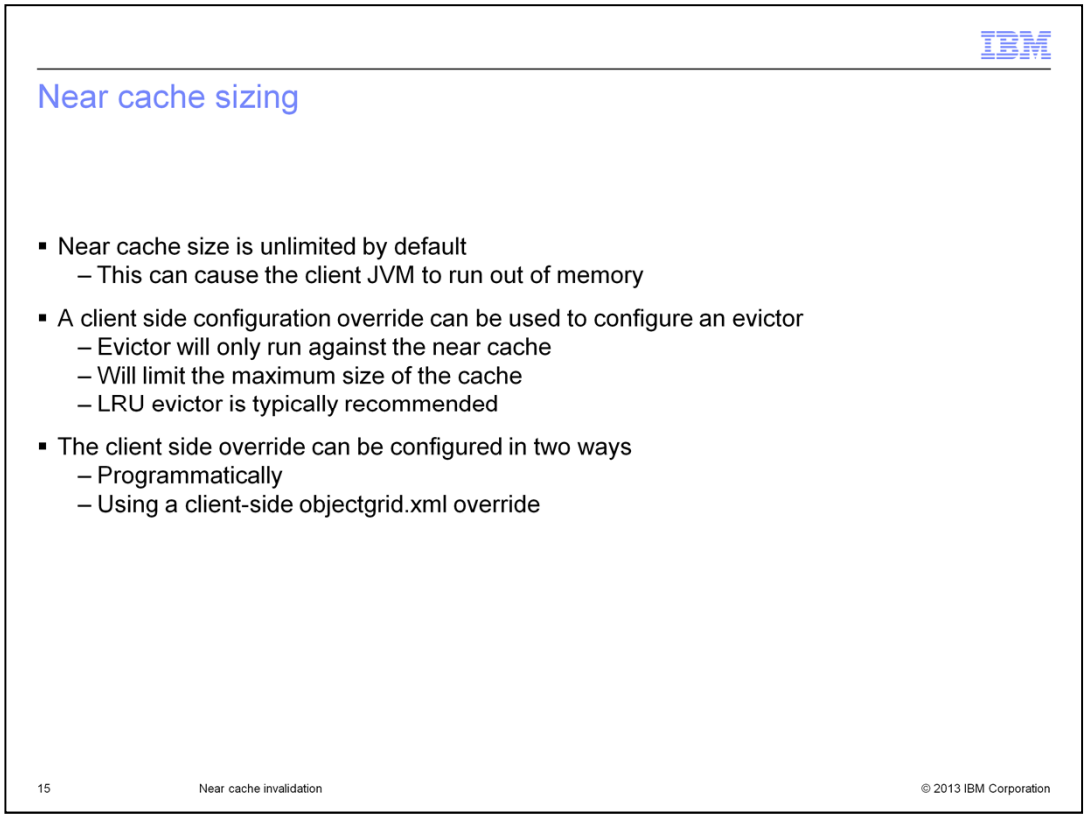
```
ReadClient
Object v1 = map1.get("key1map1");
Object v2 = map2.get("key1map2");
System.out.println("client "+clientName+" map1(invalidation) value is now: " + v1);
System.out.println("client "+clientName+" map2(no invalidation) value is now: " + v2);

output:
client reader map1(invalidation) value is now: value8
client reader map2(no invalidation) value is now: value8
client reader map1(invalidation) value is now: value9
client reader map2(no invalidation) value is now: value8
client reader map1(invalidation) value is now: value10
client reader map2(no invalidation) value is now: value8
client reader map1(invalidation) value is now: value11
client reader map2(no invalidation) value is now: value8
```

14 Near cache invalidation

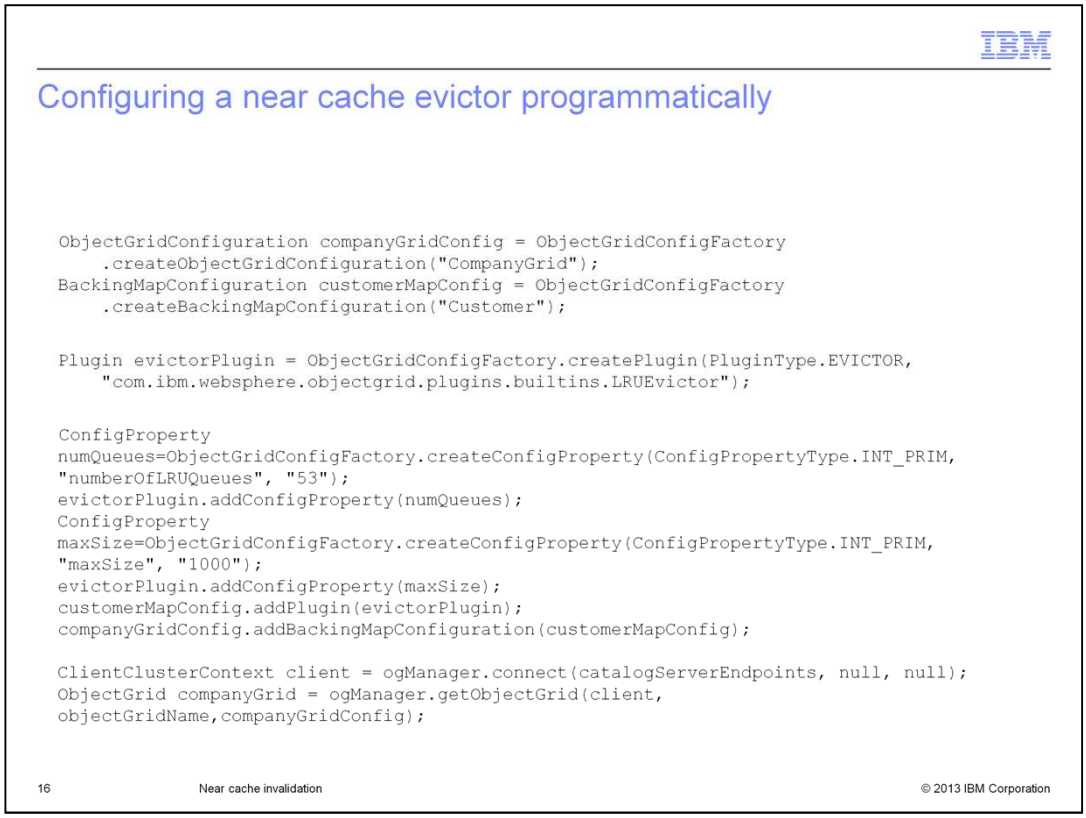
Here is a cache invalidation example that contains an update client that is updating keys in two different maps. One of those maps has invalidation enabled and one of them does not. The update client is running in a loop updating the keys - one in each map. The result is that it updates both key value pairs.

The read client is reading the values of the keys from the same maps; one with invalidation the other without.

Initially both keys had values of 8 but as the loop iterates, the client with invalidation sees the new value update from 8 to 9 and then to 10. This is because the value will keep getting "invalidated" by the updater client, so it will keep reading the latest value from the grid, rather than using a stale value from the near cache. Then the client with no invalidation caches the first value it reads and never sees an updated value. It only sees the old value of 8.

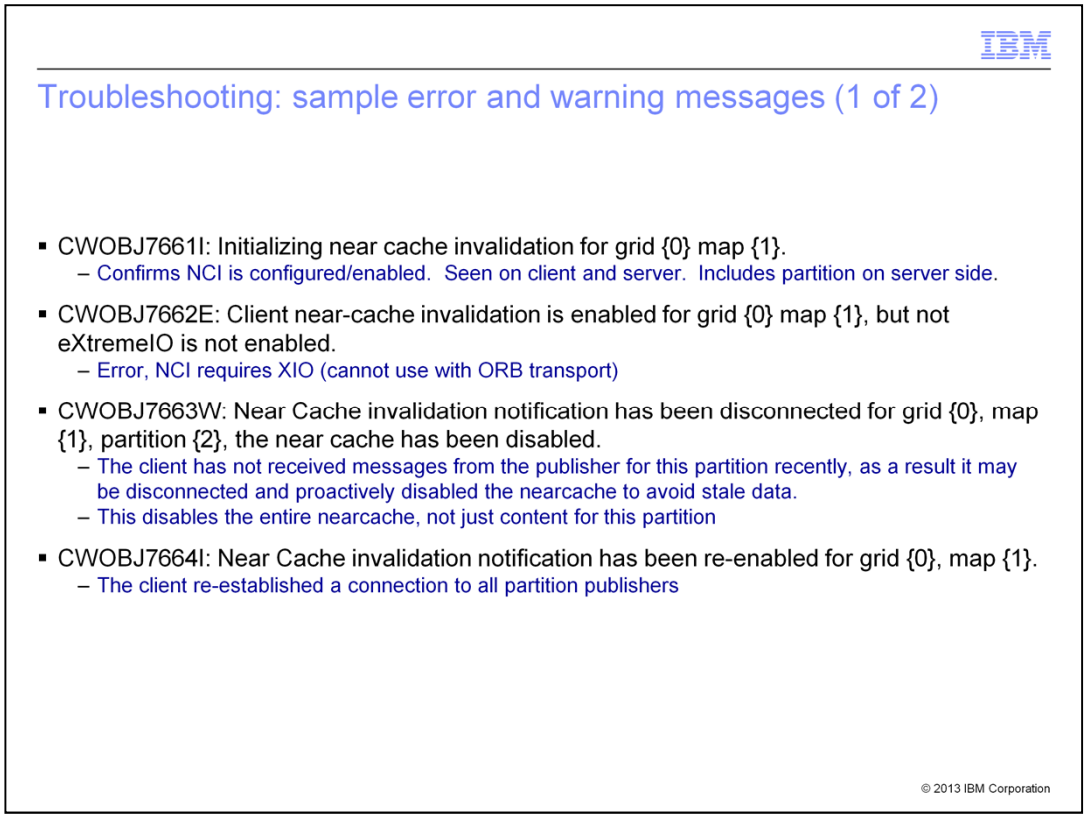


By default there is no restriction on the maximum size of a near cache. This means that a client can easily run out of memory by storing a large number of items in the near cache. To prevent this, an evictor can be configured to run against the near cache only. This evictor will have no effect on the data stored in the remote data grid, but it will control the maximum size of the near cache. The mechanism for accomplishing this is a client side override, which allows a client to specify configuration information that will only apply to the client process but not to the remote data grid configuration. These overrides can be specified programmatically or using an XML file provided by the client during initialization.

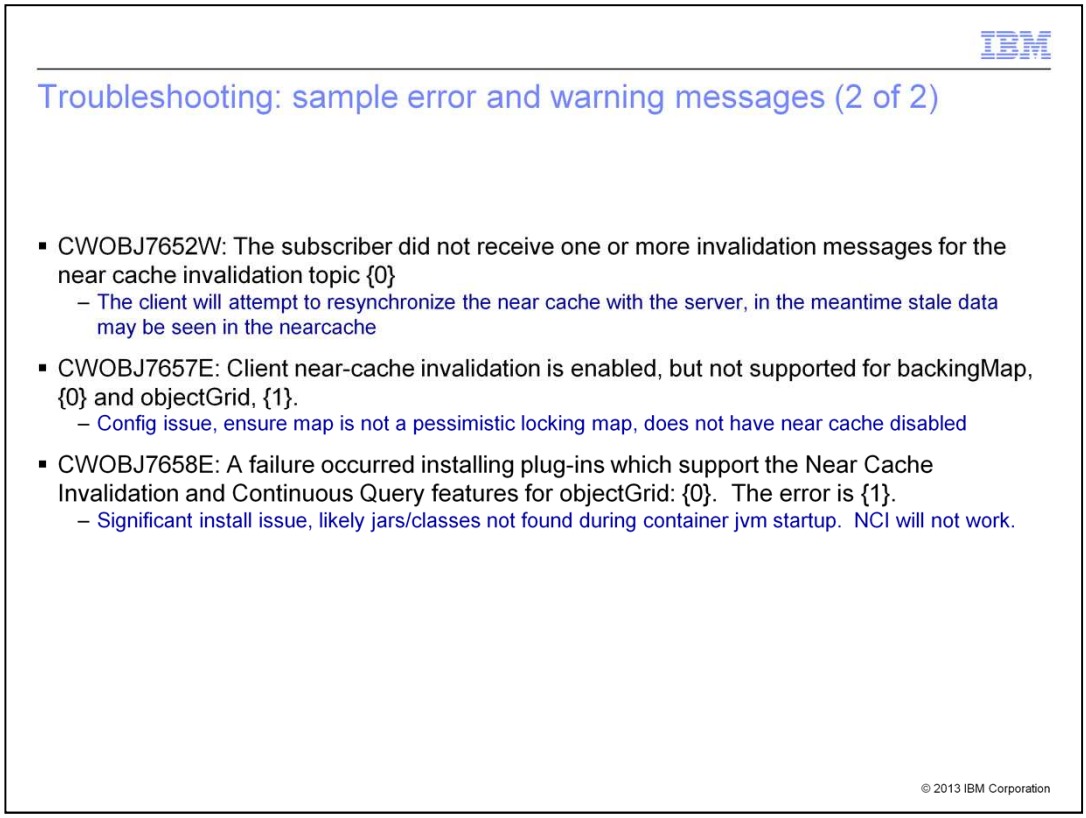


This example code illustrates the process of defining a client side configuration override that enables a Least Recently Used evictor. First, a new configuration object for the grid and map are created. Next, a new configuration plug-in of type Evictor is created, specifying the LRU evictor class name. The evictor is then configured with 53 queues and a maximum size of 1000 items per queue, resulting in an LRU evictor that will enforce a maximum size of 53,000 entries for the near cache supporting this particular map. Finally, the configured evictor is added to the configuration object and passed to the Object Grid Manager during the connection step.

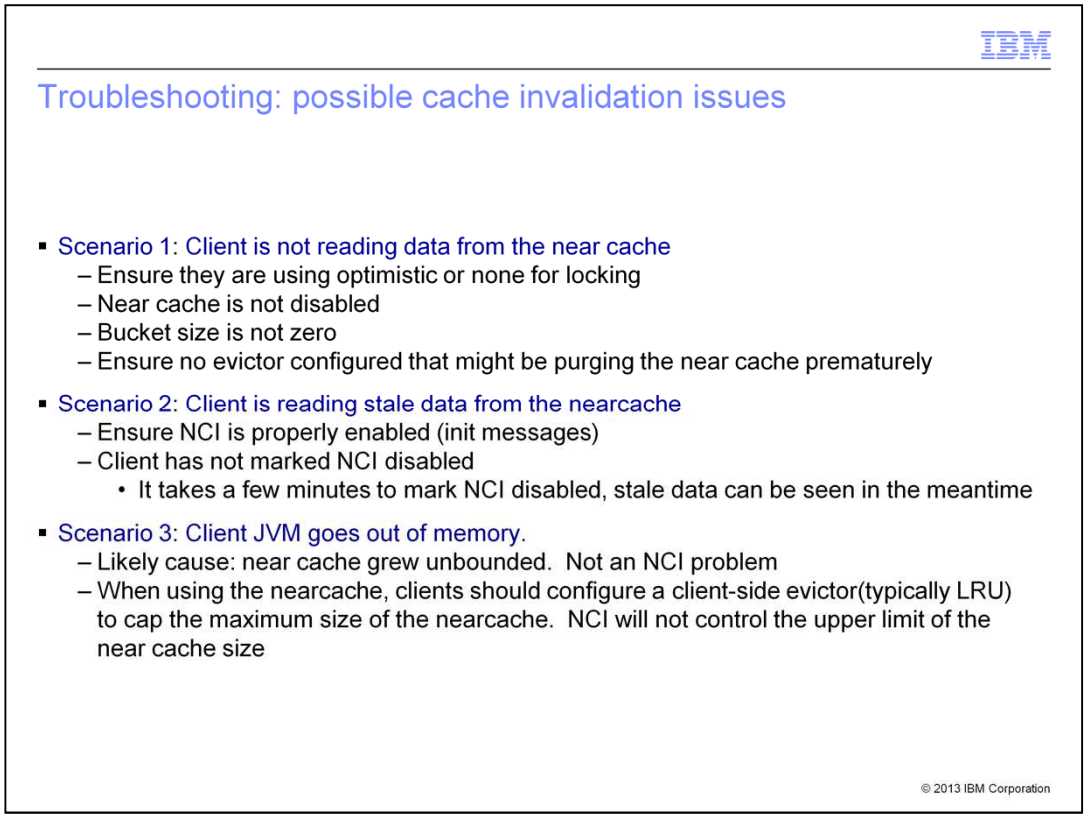
Here a client side override XML file is illustrated. The file copies the grid and map definitions from the server side objectgrid xml file, but adds a configuration definition for a Least Recently Used evictor on the Customer map. This file is then provided by the client during connection to the grid, as shown in the sample code. This will result in an LRU evictor being configured to control the size of the near cache for the Customer map.



Here are some troubleshooting error and warning messages. Some of the messages shown in this slide include a message confirming near cache invalidation is configured and enabled, near cache invalidation disabled, near cache invalidation re-enabled, and messages for failures with the feature. You can look up more of these messages in the information center.



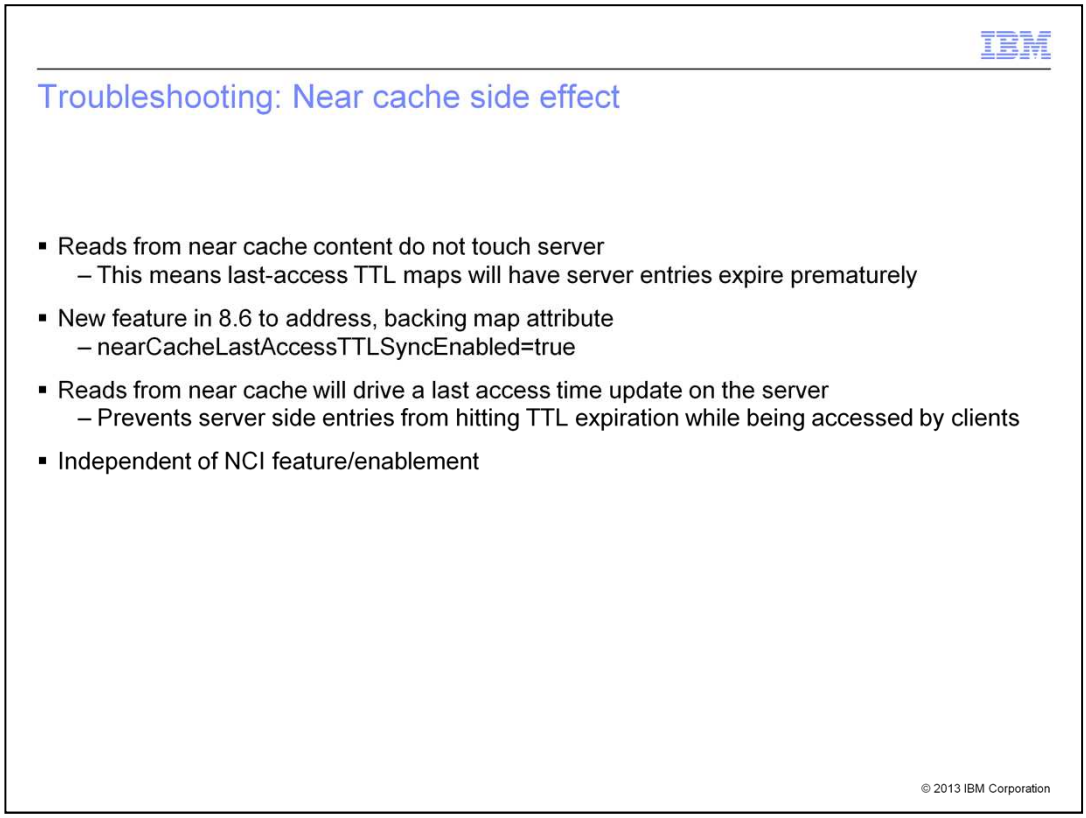
This slide shows more troubleshooting warning and error messages, including a warning that an invalidation message did not arrive, and error messages resulting from configuration and installation issues.



Here are some sample scenarios of possible near cache invalidation issues. In Scenario 1), the client is not reading data from the near cache. Make sure you are using optimistic or none for locking strategy, make sure near cache is enabled and the bucket size is not zero. Also make sure no evictor is configured that might be purging the near cache prematurely.

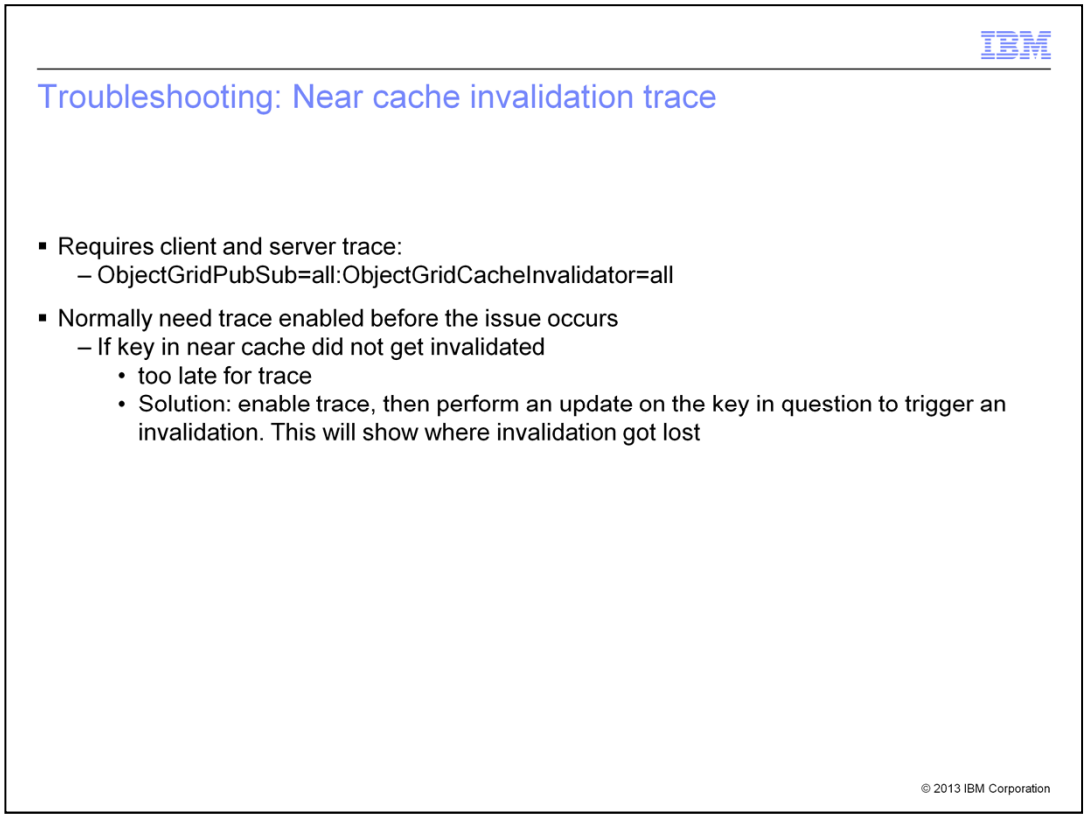
The second scenario is where client is reading stale data from the near cache. Make sure that near cache invalidation is enabled. You can do so by checking the init messages. Also make sure that the client process has not marked near cache invalidation as disabled. Note that It takes a few minutes to mark near cache invalidation disabled, and stale data can be seen in the meantime. Near cache invalidation does not guarantee that there is no stale data, and that invalidation messages take time to propagate.

In scenario three, the client JVM runs out of memory. In this scenario, the likely cause is that the near cache grew unbounded. When using the near cache, configure a client-side evictor (typically least recently used) to cap the maximum size of the near cache. Near cache invalidation will not control the upper limit of the near cache size.



Something to consider as a side effect of near cache is that reads from near cache content do not make it to the server. This means that last-access TTL maps will have server entries expire prematurely. The new feature addresses this with a backing map attribute. The attribute nearCacheLastAccessTTLSyncEnabled is set to true. Note that the backing map can be considered as an in-memory cache of committed data for an individual map.

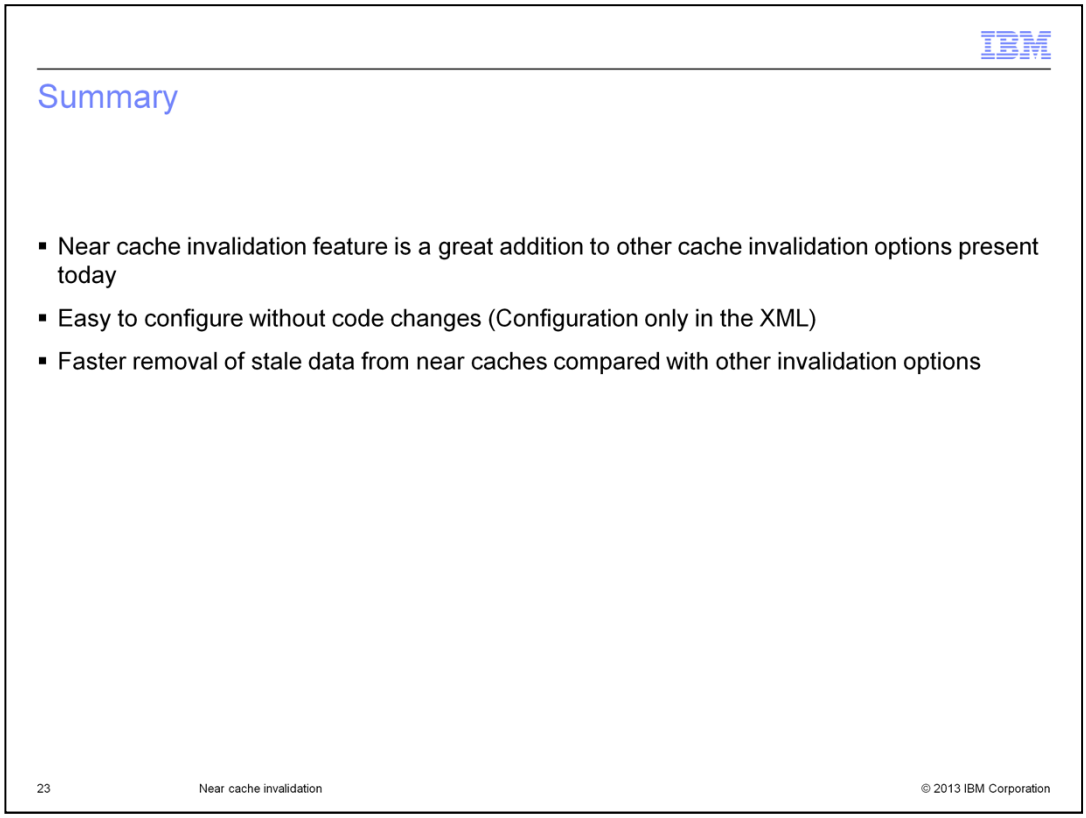
Also note that reads from near cache will drive a last access time update on the server. This prevents server side entries from hitting TTL expiration while being accessed by clients.



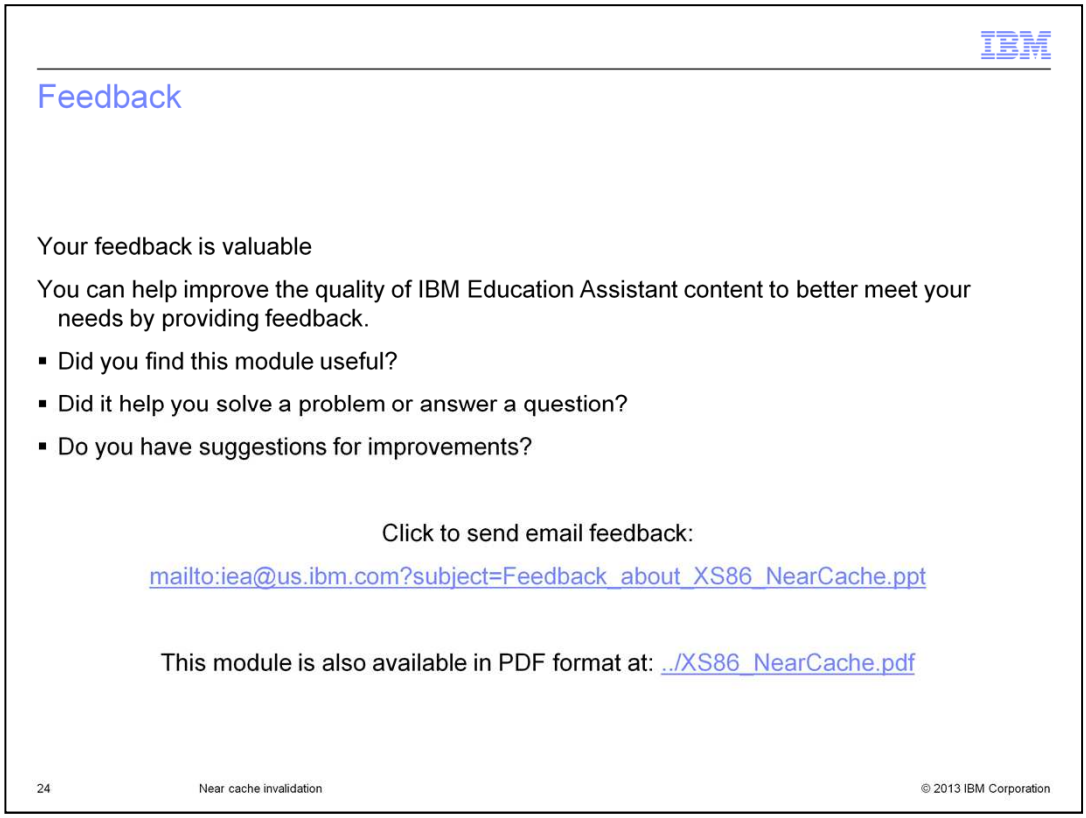
Trace can be configured for the near cache invalidation feature. This feature requires client and server trace to be turned on. To turn on trace, enter:

ObjectGridPubSub=all:ObjectGridCacheInvalidator=all

You should therefore enable trace before an issue arises. If you have a key in your near cache that did not get invalidated, it is probably too late to turn on trace and determine why it did not get invalidated if trace was not enabled. The best practice is to enable trace, then perform an update on the key in question to trigger an invalidation. This will show if any invalidation was sent and where it got lost.



In summary, you can configure near cache invalidation to remove stale data from the near cache as quickly as possible. No code changes are required. When an update, deletion, or invalidation operation is run against the remote data grid, an asynchronous invalidation is triggered in the near cache. This mechanism works more quickly than the other option of using time-to-live (TTL) eviction in the near cache. For troubleshooting information, see the WebSphere eXtreme Scale version 8.6 information center.



You can help improve the quality of IBM Education Assistant content by providing feedback.

Trademarks, disclaimer, and copyright information