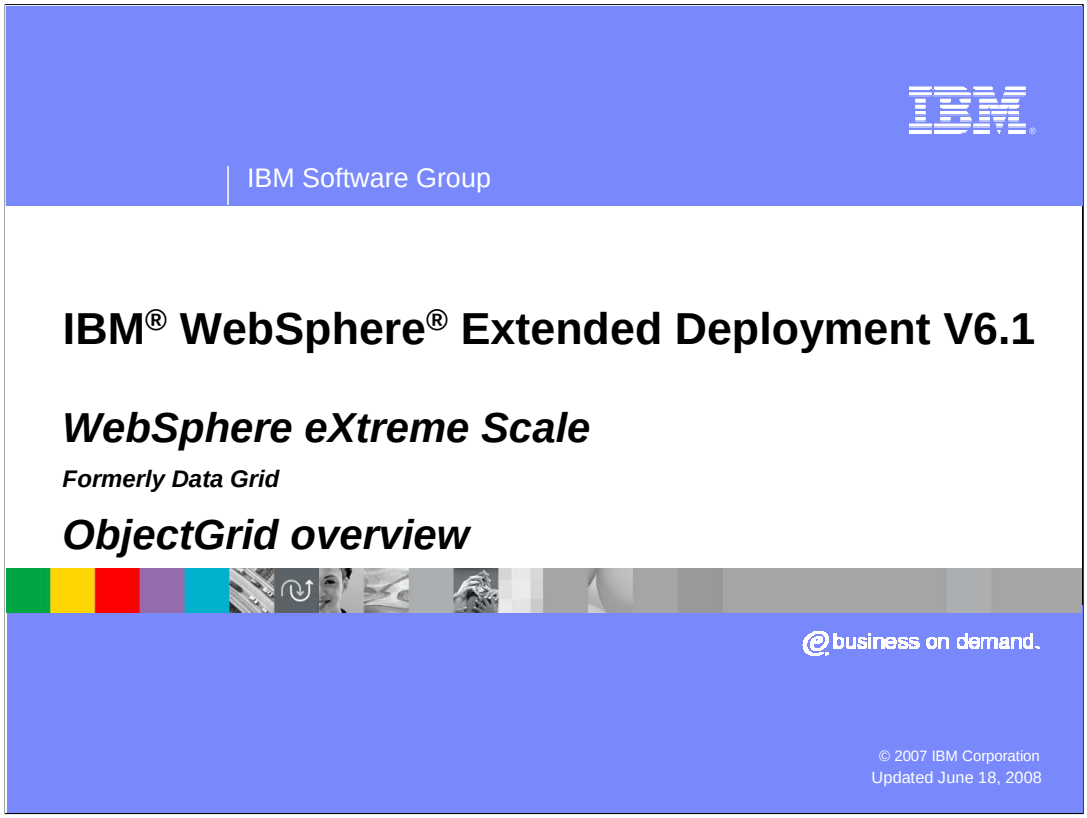


This presentation will cover ObjectGrid, a high performance cache framework in WebSphere Extended Deployment V6.1.

This module was originally recorded for WebSphere Extended Deployment Data Grid, which is now called WebSphere eXtreme Scale. Though the module uses the previous names, the technical material covered is still accurate.

Agenda

- ObjectGrid overview
- Configuring ObjectGrid
- ObjectGrid topology options
- Summary

The agenda is to first introduce ObjectGrid, then briefly cover ObjectGrid configuration and topology options.

Section

Overview

This section will provide an overview of ObjectGrid technology.

ObjectGrid overview

- ObjectGrid is a high performance cache framework for storing Java™ objects needed by an application
- Transaction support
  - Operations can take place within the scope of a transaction for consistency
  - 1-phase commit supported
- Customizable cache life cycle features
  - Declaration, configuration, invalidation, size management, cache loading

ObjectGrid is a high performance, transactional cache facility for Java objects.

An ObjectGrid is highly customizable. For example, interfaces are provided for implementing custom cache loading, size management, and invalidation schemes.

ObjectGrid overview

- Can be backed by hardened storage
  - Cache loader interface enables hardening using storage technology of your choice (like a database)
- Securable using Java Authentication and Authorization Service (JAAS) API
- Support for hierarchical, tag-based invalidation and custom eviction policies

The cache loader interface enables you to implement a class that uses the hardened storage technology of your choice, such as a database, as a backing store for the cache. An ObjectGrid instance is securable using the standard Java Authentication and Authorization Service API. Hierarchical, tag-based invalidation using keywords, is another feature of an ObjectGrid cache.

Scalability

- ObjectGrid can scale to hold hundreds of JVMs of data for very large data sets
  - ObjectGrid clusters can be replicated and partitioned for fault-tolerance and high performance
  - Scales nearly linearly with additional hardware
  - Supports thousands of concurrent clients
  - Supports 32-bit or 64-bit JVMs
- Cluster members communicate with each other using an ultra-high speed pub sub system
  - High performance and low processor overhead

An ObjectGrid is highly scalable, supporting a local cache within a single Java virtual machine, all the way up to a fully replicated cache distributed across numerous cache servers. In fact, ObjectGrid technology can scale a vast array of distributed and scalable data services spread across ObjectGrid clusters throughout an entire enterprise. ObjectGrid cluster members communicate with each other using a highly optimized, dedicated publish and subscribe messaging system.

HTTP session persistence

- HTTP Sessions can be replicated across servers using ObjectGrid
  - A servlet filter that enables session replication can be inserted into any Web application
  - Provides a session persistence approach that is independent of the WebSphere cell infrastructure
- Older versions of WebSphere products can use ObjectGrid as an upgraded session persistence mechanism
- Non-WebSphere servers (such as Geronimo or JBoss) can also use this servlet filter

Object grid can be used to persist and set up your HTTP session data. Basically, you can override the default session manager in the base application server to provide HTTP session management capabilities for an associated application. The ObjectGrid session manager can create HTTP sessions and manage the life cycle of HTTP sessions that belong to the application. This life cycle management includes the invalidation of sessions based on timeout or an explicit servlet or JavaServer Pages (JSP) call and the invocation of session listeners that are associated with the session or the Web application.

Runtime environment support - distributed platforms

- ObjectGrid caches are supported in
  - WebSphere Extended Deployment V6.0 (or greater)
  - WebSphere Application Server V5.1.1 (or greater) runtime environments
- ObjectGrid libraries are provided only with WebSphere Extended Deployment
  - Stand-alone ObjectGrid can be installed using the Compute Grid package
- ObjectGrid can also be run ‘stand-alone’
  - ObjectGrid runtime requires J2SE 1.4.2 or higher JVM

While ObjectGrid technology is a component of the WebSphere Extended Deployment data grid package, ObjectGrid caching is also supported in a WebSphere Application Server Network Deployment version 5.1 (or higher) server by including the ObjectGrid library, wsobjectgrid.jar, in your application classpath. In addition, ObjectGrid can run in a stand-alone Java virtual machine or other application server product by using the ObjectGrid.jar file. The client server object grid model employs a dedicated single-purpose object grid server. This server does not run in a WebSphere server, and is a scaled-down server in its own right. The stand-alone object grid server is available in WebSphere Extended Deployment V6.1. Object Grid requires a Java 1.4.2. Java virtual machine to run, but can also leverage Java 5 features such as annotations.

z/OS® ObjectGrid support

- WebSphere Extended Deployment V6.1 does not support ObjectGrid servers on the z/OS platform
- z/OS applications must use an ObjectGrid server running on a distributed platform

An application deployed to WebSphere Extended Deployment V6.1 for z/OS that requires access to an ObjectGrid server must use an off-platform ObjectGrid server such as Linux®.

Working with Object Grid data

- An ObjectGrid contains one or more map-like objects (ObjectMaps)
  - Interface similar to java.util.Map
  - ObjectMaps support all of the expected map methods
    - put(), get(), insert(), update(), and so on.
- Objects are stored as map entries (key value pairs)
  - Can be entered into the map by the application
  - Can be loaded from an external source using a custom loader class
- Can be indexed on key or attribute values

Java objects are stored in an ObjectGrid using key value pairs within map objects called ObjectMaps. Data can be put into and retrieved from an ObjectMap within the scope of a transaction using all of the usual map-like methods through an interface similar to java.util.Map. The map can be populated directly by the application, or it can be loaded from a back-end store by implementing a custom cache loader class.

Loaders

- A loader is an extensible object type used for associating an ObjectGrid map with a backing data store
- When requested data is not in the map, the request is passed to the data store using the loader
  - Maps can also be configured to preload data
    - In an ObjectGrid cluster, the preload operation can be restarted on another server if the primary server fails during preloading
- A map also uses the loader class to persist data back into the data store
  - An explicit call to flush() pushes the data to the data store, but does not commit a transaction

The cache loader interface allows you to implement a custom Java class to load cache data from a back-end data store and to persist changed values back to the hardened store, independent of the state of a transaction. Data can optionally be preloaded from the data store at server startup.

Query

- ObjectGrid query API
  - Flexible query engine for retrieving Entities and Java objects
- ObjectGrid stream query API
  - Continuous query over streaming data

In WebSphere Extended Deployment version 6.1, ObjectGrid includes two powerful query mechanisms for retrieving objects from the ObjectGrid cache using non-key attributes.

The ObjectGrid query API provides a flexible query engine for retrieving entities using the EntityManager API and Java objects using the ObjectQuery API. The query engine allows select type queries over an entity or Object-based schema using the ObjectGrid query language.

The ObjectGrid stream query is a continuous query over the in-flight data that has been stored in ObjectGrid maps.

Section

ObjectGrid configuration

This section will cover configuring an ObjectGrid instance.

Configuration options

- ObjectGrid instances can be configured programmatically or using Extensible Markup Language (XML) files
  - Sample configuration files are provided with WebSphere Extended Deployment installation
- XML file defines the Java implementations that should be used and how they are associated
- To create an ObjectGrid using an XML file:

```
ObjectGridManager myObjectGridManager =
  ObjectGridManagerFactory.getObjectGridManager();

ObjectGrid myObjectGrid = objectGridManager.createObjectGrid(“newGrid”,
  “newgrid.xml”);
```


To cache objects using ObjectGrid, you must first create an ObjectGrid instance within your application. The instance can be configured programmatically, or created based on configuration data stored in a descriptor XML file. The code snippet shown here illustrates how to instantiate the “newGrid” ObjectGrid based on a configuration file, “newgrid.xml”, using the ObjectGridManager class. You can learn about ObjectGrid configuration files by exploring the samples provided in the “optional libraries” directory after installing WebSphere Extended Deployment.

Configuration files

- ObjectGrid configuration file
  - Defines ObjectGrids, associated maps, and plug-ins
- Cluster configuration file
  - Maps ObjectGrids to a topology
- ObjectGrid servers can also bootstrap to a running server and read the configuration over TCP/IP

Configuration files are used for configuring the ObjectGrid itself, but they are also used to define the associated maps, or what kind of data is being stored in the ObjectGrid, and any plug-ins you might be using. If you plan to have a topology of distributed ObjectGrid servers, you would use a cluster configuration file. The configuration information can be read either from the local file system or you can bootstrap from a running server and read its configuration. This is a good way to ensure that all of your ObjectGrid servers use the same configuration.

Sample configuration

```
<objectGrids>
  <objectGrid name="ObjectGrid">

        <bean id="TransactionCallback“
        className="com.ibm.websphere.objectgrid.sample.HeapTransacti
        onCallback" />

        <backingMap name="employees" readOnly="false"
        pluginCollectionRef="employees" preloadMode="false"
        lockStrategy="optimistic"
        copyMode="COPY_ON_READ_AND_COMMIT" />

  </objectGrid>
</objectGrids>
```


This snippet shows what a simple ObjectGrid configuration looks like. The code specifies that the runtime use a custom TransactionCallback implementation by pointing to the implementation class. The grid will contain a Map named ‘employees’, and you can see that data preloading has been disabled and a locking strategy and copy mode have been set for the map.

Sample configuration

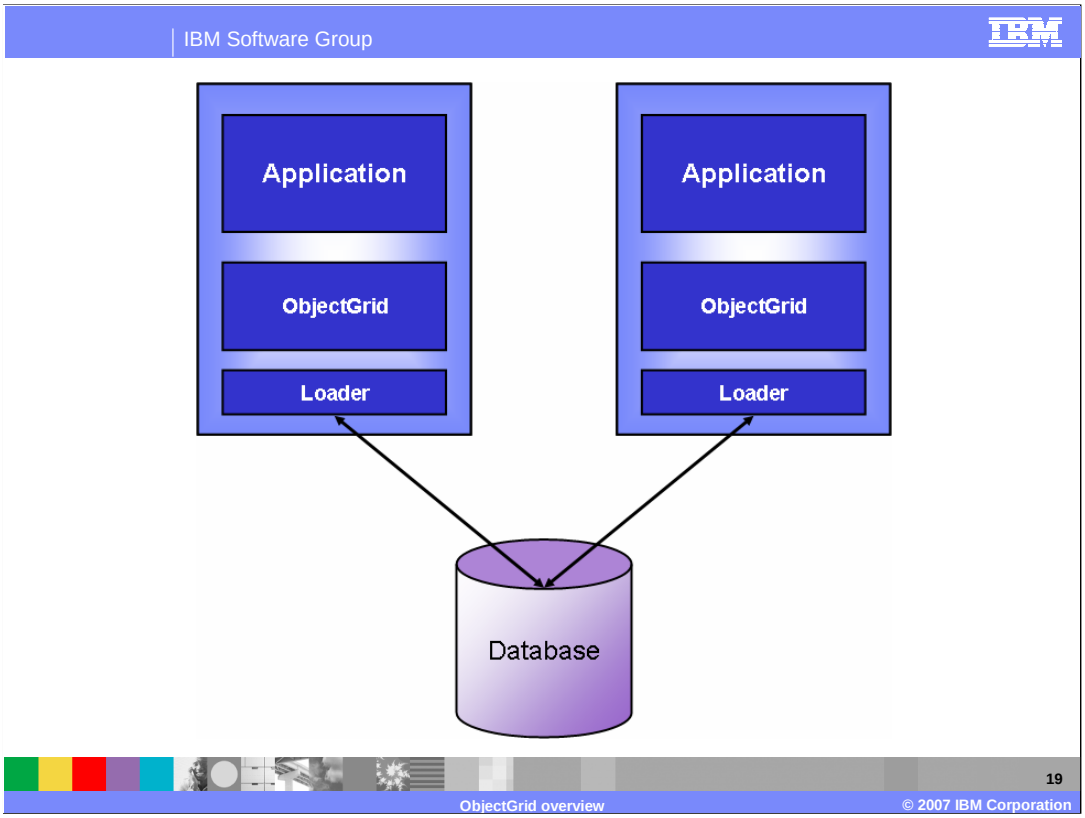
```
<backingMapPluginCollections>

  <backingMapPluginCollection id="employees">
    <bean id="Loader“
    className="com.ibm.websphere.objectgrid.sample.HeapCacheLoader">
      <property name="preLoadClassName" type="java.lang.String"
      value="com.ibm.websphere.objectgrid.sample.EmployeePreload"
      description="name of class that implements HeapPreloadData
      interface" />
    </bean>
    <bean id="ObjectTransformer"
    className="com.ibm.websphere.objectgrid.sample.EmployeeObjectTra
    nsformer" />
    <bean id="OptimisticCallback"
    className="com.ibm.websphere.objectgrid.sample.EmployeeOptimisti
    cCallbackImpl" />
  </backingMapPluginCollection>


</backingMapPluginCollections>
```


This snippet contains the remainder of the configuration that began on the previous page, and it defines the custom plug-ins that will be used on the map named ‘employees’. This particular map uses custom Loader, ObjectTransformer, and TransactionCallback implementations. These plug-ins are all associated with a map, so if you have multiple maps within an ObjectGrid instance, they can use different sets of custom plug-ins.

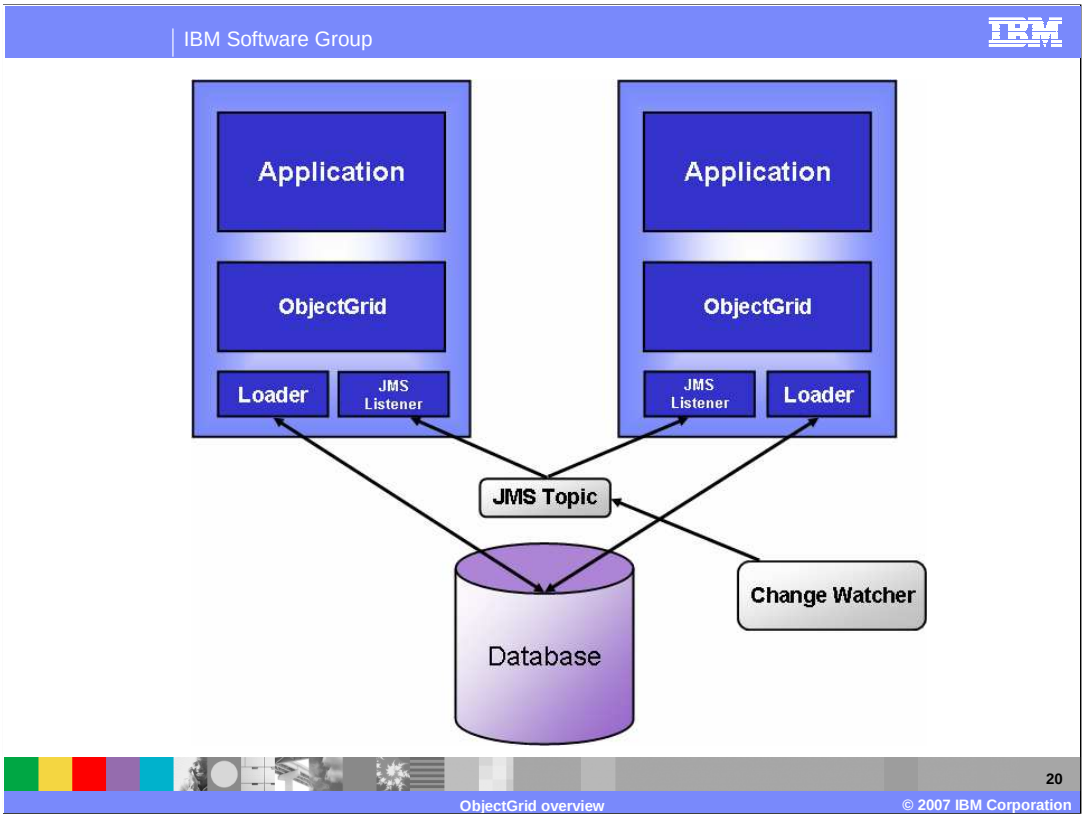
Section

ObjectGrid topology options

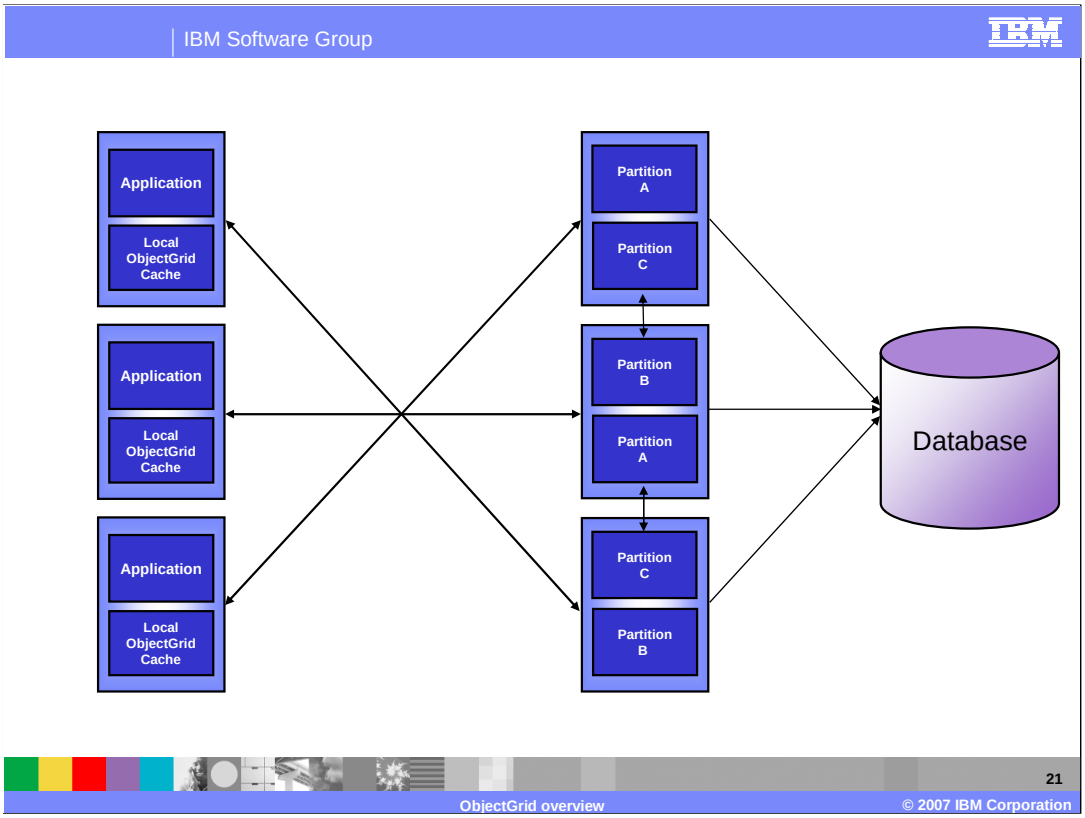
This section will discuss different ObjectGrid topology options.



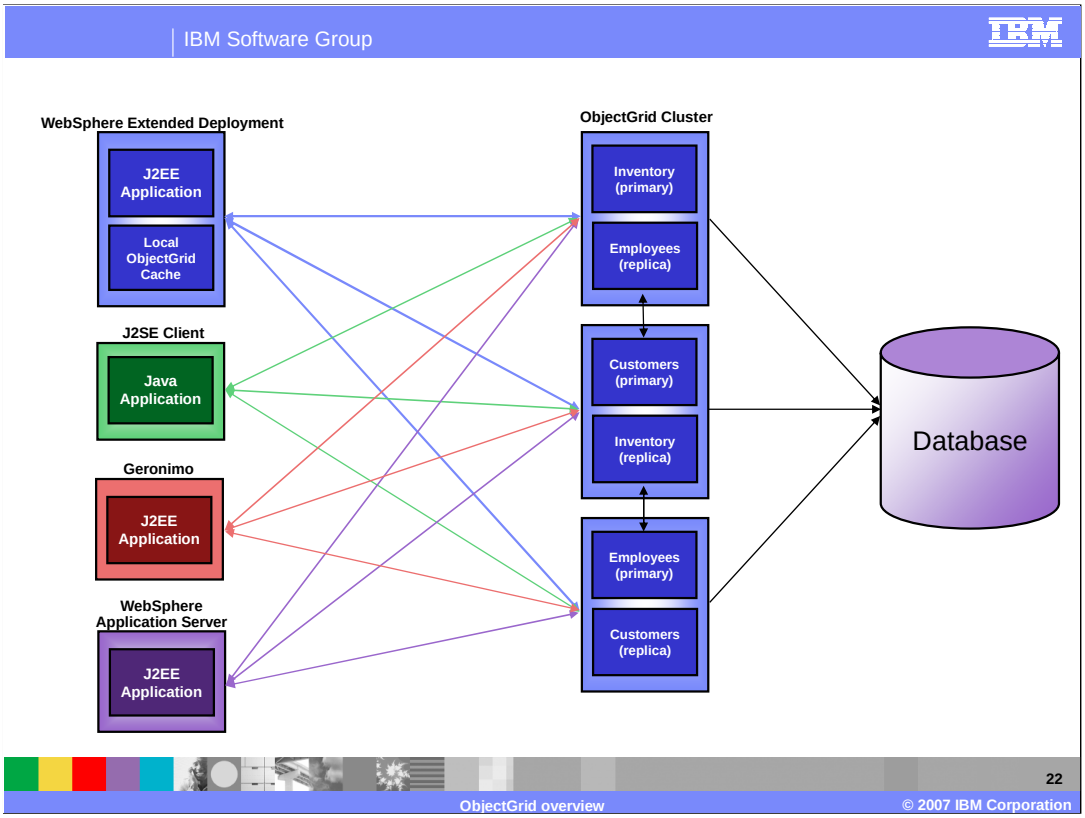
The first scenario shows multiple application servers loading data from the same database and caching it locally. In this scenario, each of the servers can have stale data in its local cache if the other server changes a value, but optimistic locking is used to ensure that stale data does not get written back to the database.



The scenario shown here is similar, except that a change watcher is employed to alert the servers whenever a change is made to the database by some other process. A lightweight JMS system is used for notification, in which both servers subscribe to a JMS topic that receives messages from the change watcher. When they receive messages on this topic, the servers invalidate the specified data. Again, optimistic locking is used to ensure that stale data is not written back to the database.



In this multi-tiered example, each application server contains an ObjectGrid instance that is used as a near cache. For objects that are not found in the local cache, a query can be made to the tier of ObjectGrid servers. This tier hosts a much larger set of data in memory, and objects can be retrieved from this tier much faster than by accessing the database. This example shows a replicated and partitioned tier of ObjectGrid servers. Partitioning enables higher performance access to the data, and replication ensures availability of each partition. The ObjectGrid tier can be configured to use a Loader that queries the database when the requested data is not found in the cache.



This example illustrates a topology in which several different client types have access to a set of data being hosted by an ObjectGrid cluster. The three maps – Inventory, Customers, and Employees – each have a primary instance and a replica. Data changes are periodically sent from the primaries to the replicas to ensure that the replica remains up to date. These replicas can serve client requests in case the primary fails, making the ObjectGrid highly available.

Section

Summary

This section will provide a summary and references for this presentation.

Summary

- ObjectGrid provides a high-performance, scalable object cache for applications running on WebSphere Extended Deployment
- ObjectGrid features can be customized by implementing custom Java classes
  - Cache loading, invalidation, and more are extensible

ObjectGrid provides a high-performance transactional cache technology for WebSphere Extended Deployment. Objects can be stored in and retrieved from the cache using map objects. The ObjectGrid cache is designed to be highly extensible, so that you can implement a cache system that meets the needs of your specific environment.

Getting started with ObjectGrid

- ObjectGrid samples guide
  - optionalLibraries/ObjectGrid/SamplesGuide.htm
- Comments in ObjectGrid sample code
  - optionalLibraries/ObjectGrid/objectGridSamples.jar
- ObjectGrid 6.1 documentation
  - http://www-03.ibm.com/developerworks/wikis/display/objectgrid/Getting+started
- ObjectGrid 6.1 user guide
  - http://www.ibm.com/developerworks/wikis/display/objectgridprog/

You can learn about ObjectGrid configuration files by exploring the samples provided in the “optional libraries” directory after installing WebSphere Extended Deployment, and by visiting the ObjectGrid documentation on the internet.

Feedback

Your feedback is valuable

You can help improve the quality of IBM Education Assistant content to better meet your needs by providing feedback.

- Did you find this module useful?
- Did it help you solve a problem or answer a question?
- Do you have suggestions for improvements?

Click to send e-mail feedback:

mailto:iea@us.ibm.com?subject= Feedback about XD61_ObjectGrid_Overview.ppt

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

Trademarks, copyrights, and disclaimers

The following terms are trademarks or registered trademarks of International Business Machines Corporation in the United States, other countries, or both:

IBM WebSphere z/OS

J2EE, J2SE, Java, JVM, and all Java-based trademarks are trademarks of Sun Microsystems, Inc. in the United States, other countries, or both.

Product data has been reviewed for accuracy as of the date of initial publication. Product data is subject to change without notice. This document could include technical inaccuracies or typographical errors. IBM may make improvements or changes in the products or programs described herein at any time without notice. Any statements regarding IBM's future direction and intent are subject to change or withdrawal without notice, and represent goals and objectives only. References in this document to IBM products, programs, or services does not imply that IBM intends to make such products, programs or services available in all countries in which IBM operates or does business. Any reference to an IBM Program Product in this document is not intended to state or imply that only that program product may be used. Any functionally equivalent program, that does not infringe IBM's intellectual property rights, may be used instead.

Information is provided "AS IS" without warranty of any kind. THE INFORMATION PROVIDED IN THIS DOCUMENT IS DISTRIBUTED "AS IS" WITHOUT ANY WARRANTY, EITHER EXPRESS OR IMPLIED. IBM EXPRESSLY DISCLAIMS ANY WARRANTIES OF MERCHANTABILITY, FITNESS FOR A PARTICULAR PURPOSE OR NONINFRINGEMENT. IBM shall have no responsibility to update this information. IBM products are warranted, if at all, according to the terms and conditions of the agreements (for example, IBM Customer Agreement, Statement of Limited Warranty, International Program License Agreement, etc.) under which they are provided. Information concerning non-IBM products was obtained from the suppliers of those products, their published announcements or other publicly available sources. IBM has not tested those products in connection with this publication and cannot confirm the accuracy of performance, compatibility or any other claims related to non-IBM products.

IBM makes no representations or warranties, express or implied, regarding non-IBM products and services.

The provision of the information contained herein is not intended to, and does not, grant any right or license under any IBM patents or copyrights. Inquiries regarding patent or copyright licenses should be made, in writing, to:

IBM Director of Licensing
IBM Corporation
North Castle Drive
Armonk, NY 10504-1785
U.S.A.

Performance is based on measurements and projections using standard IBM benchmarks in a controlled environment. All customer examples described are presented as illustrations of how those customers have used IBM products and the results they may have achieved. The actual throughput or performance that any user will experience will vary depending upon considerations such as the amount of multiprogramming in the user's job stream, the I/O configuration, the storage configuration, and the workload processed. Therefore, no assurance can be given that an individual user will achieve throughput or performance improvements equivalent to the ratios stated here.

Note to U.S. Government Users - Documentation related to restricted rights-Use, duplication or disclosure is subject to restrictions set forth in GSA ADP Schedule Contract and IBM Corp.