IBM Software Group

IBM® WebSphere® Extended Deployment V6.1

WebSphere® eXtreme Scale

Formerly Data Grid

ObjectGrid programming concepts

business on demand.

© 2007 IBM Corporation

Updated June 12, 2008

This presentation will provide an introduction to basic ObjectGrid programming concepts.

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

- Programming concepts
- Creating an ObjectGrid
- Working with ObjectGrid data
- ObjectGrid query
- Summary

© 2007 IBM Corporation

This presentation will first cover general ObjectGrid programming concepts, then will describe how to create a simple ObjectGrid instance. It will then discuss ObjectGrid data access methods and some of the extensible ObjectGrid interfaces. Finally this presentation will provide an overview of the advanced query capability included in Extended Deployment version 6.1 ObjectGrid.

ObjectGrid overview

- An ObjectGrid is a transactional space for holding Java™ objects needed by an application
  - It can be used to cache frequently used objects in a memory replicated store before storing them in a database
- Customizable cache life cycle features
  - Declaration, configuration, invalidation, size management, cache loading
- Can be backed by hardened storage
  - Cache loader interface enables hardening using storage technology of your choice (such as a database)

© 2007 IBM Corporation

WebSphere Extended Deployment ObjectGrid provides a scalable, high performance, transactional cache for Java objects. An ObjectGrid is highly customizable – interfaces are provided for controlling all aspects of an object cache and the object stored in it. For instance, 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.

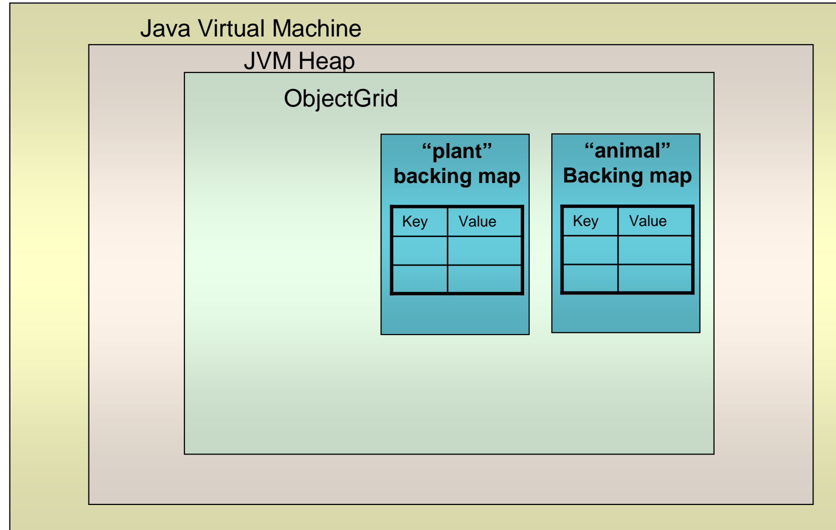
Section

Programming concepts

© 2007 IBM Corporation

This section will cover basic concepts used in ObjectGrid programming.

Backing maps

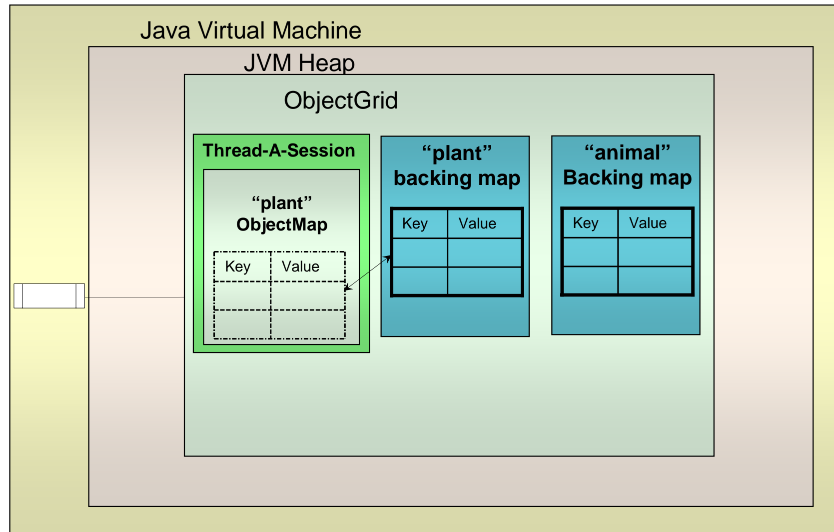


© 2007 IBM Corporation

Whether you are working with a local ObjectGrid cache within an application or a distributed ObjectGrid in a client-server environment, an ObjectGrid instance acts as a container for Java™ objects within a Java virtual machine. An ObjectGrid can be physically mapped to a single JVM or a thousand server grid spread over multiple data centers.

A backing map is a container within the ObjectGrid instance for key/value pairs. It allows an application to store an object indexed by a key. The backing maps are where the ObjectGrid stores its cached data, and all of the objects in a single backing map must be of the same type. Each ObjectGrid has one or more backing maps defined.

Sessions



© 2007 IBM Corporation

ObjectGrid provides transactional access to data in the backing maps. In order to do this you must create a session and access data from within the session. The session acts like a scratchpad: you 'get' data from the backing map into the session and work with it there, not in the backing map. When you 'commit' a transaction the working copy replaces the copy in the backing map; when you 'rollback' a transaction the working copy is discarded.

Copy modes

- Three different copy modes are supported
- Differing performance and data integrity traits:
  - COPY_ON_READ_AND_COMMIT
    - A copy of data is made on every read and commit action
    - Safest copy mode: thread never has direct reference to objects in map
  - COPY_ON_READ
    - Copy is not made when commit() is called
    - Applications must not reuse objects after commit() is called to ensure data integrity
    - Better performance than COPY_ON_READ_AND_COMMIT
  - COPY_ON_WRITE
    - Minimizes copying in read-most scenarios for best performance
    - Data must be accessed through a dynamic proxy

© 2007 IBM Corporation

The copy mode setting determines if and when copies are made of objects in the backing map and given to the application code, as opposed to when objects are passed by reference.

'COPY_ON_READ_AND_COMMIT' ensures that the application never has a direct reference to an object in the map. When the application calls the "get" method to retrieve an object, the session will create a copy of the object from the backing map. When the transaction is committed, any changes that are committed by the application are copied back to the backing map. This ensures the application can never corrupt the data in the backing map. This mode provides the best data integrity, but is the slowest because it makes a deep copy of the object every time a read or commit is performed.

'COPY_ON_READ' is similar to 'COPY_ON_READ_AND_COMMIT' in that data is copied from the backing map into the session. However when the transaction is committed the working copy of the object from the session is placed directly into the backing map, replacing the original. To ensure data integrity, this mode requires that the application guarantee that objects are not reused after a transaction is committed; doing so could corrupt data in the backing map. This mode provides better performance than 'COPY_ON_READ_AND_COMMIT', because data is not copied when the commit operation is performed.

The 'COPY_ON_WRITE' mode requires an application to access objects indirectly using a Java dynamic proxy. It provides the best performance in read-most scenarios, which are very common, while still ensuring data integrity. The proxy will create a copy of the object only if the application modifies it by calling a "set" method. Otherwise, "read" operations are deferred to the object instance in the backing map. When the transaction commits, the copy, if it was created, is placed in the backing map.

Locking strategies

- Optimistic locking
  - Locks are only acquired during the actual update action
    - Will throw an exception if two threads try to update the same data simultaneously
  - Most useful for “read mostly” Maps
- Pessimistic locking
  - Data is locked when a transaction “gets” data
    - High performance impact
  - Best used when optimistic locking results in frequent collisions
- None
  - ObjectGrid does not manage concurrency
  - Relies on EJB persistence manager or concurrency provided by a Loader

© 2007 IBM Corporation

Like most cache frameworks, you have a choice of data locking strategies when working with ObjectGrid. Optimistic locking is the most common, and only acquires exclusive locks when writing data to the map. If two threads try to get the same lock simultaneously, an exception will be thrown, and they will have to try again. For maps that only update data occasionally, this is the preferred mode. Pessimistic locking, on the other hand, assumes that data may be updated each time it is accessed. This means that data is locked for updating each time it is accessed. While this method is generally slower, it can be better than optimistic locking for write-heavy applications that generate collisions frequently. You can also choose to have ObjectGrid ignore concurrency issues entirely, and rely on a custom Loader or the EJB persistence manager for concurrency.

Section

Creating an ObjectGrid

© 2007 IBM Corporation

This section will cover creating a simple ObjectGrid instance.

Instantiating an ObjectGrid

- 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
- com.ibm.websphere.objectgrid.* package contains classes needed to use ObjectGrid
- To create an ObjectGrid using an XML file:

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

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

© 2007 IBM Corporation

To cache objects using ObjectGrid, you must create an ObjectGrid instance within your application. The instance can be configured programmatically, or created based on configuration data stored in an XML file. The code snippet shown here illustrates how to instantiate an ObjectGrid based on a configuration file, using the ObjectGridManager class. The file "newgrid.xml" fully defines the ObjectGrid configuration, including the contained backing maps and required copy modes. You can learn about ObjectGrid configuration files by exploring the samples provided in the "optionalLibraries" directory after installing WebSphere Extended Deployment Data Grid.

ObjectGrid sessions

- ObjectGrid operations can be performed within the scope of a one-phase commit transaction using an ObjectGrid session object
- Simple example:

```
BackingMap m = myObjectGrid.defineMap("testMap");
Session s = myObjectGrid.getSession();
ObjectMap testMap = s.getMap("testMap");
s.begin();
testMap.insert("Joe Employee", employeeRecord);
s.commit();
```

© 2007 IBM Corporation

After creating an ObjectGrid instance you must get a session object, then get access to the ObjectMap within the context of the session, as shown here. You can then perform actions against the map in between calls to the transactional “begin” and “commit” methods on the session. This basic example puts an employeeRecord object into the map using the string “Joe Employee” as a key. In this case the key is a simple string, but keys can be any type of object.

Section

Working with ObjectGrid data

© 2007 IBM Corporation

This section will cover the basics of working with an ObjectGrid data.

Working with ObjectGrid data

- An ObjectGrid contains one or more map-like objects (ObjectMaps)
  - ObjectMaps support all of the expected Map methods
    - Put(), get(), insert(), update(), …
- 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 custom loader objects

© 2007 IBM Corporation

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 using all of the usual Map-like methods, within the scope of a transaction. The Map can be solely populated by the application, or it can be loaded from a back-end store by implementing a custom cache loader.

Entities

- Alternative mechanism for interacting with distributed cache
- Retrieve/find POJO's using the EntityManager APIs
- Can store complex object graphs
  - Persist POJO's along with any relationships it may have in a transactional manner.
- Similar to Java Persistence API Entities

© 2007 IBM Corporation

In addition to map-like data access, ObjectGrid supports accessing data stored as entities. Entities use two 'tuple' objects, one for the key and another for the value. Each tuple is an array of attributes and foreign keys. The ObjectGrid provides an EntityManager API, which is similar to the Java Persistence API, for storing and retrieving entities. The EntityManager API uses the map-based infrastructure, but converts entity objects to and from tuples before storing or reading them from the Map. Entities can have relations to other Entities and the relationships are maintained automatically. ObjectGrid supports one-to-one, one-to-many, many-to-one and many-to-many relationships. This allows applications to easily have complex object graphs that span multiple maps. Each entity is physically mapped to a single backing map.

Loaders

- A Loader is an extensible object type used for associating an ObjectMap 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 be configured to preload data
- A Map also uses the Loader class to persist data back into the data store

© 2007 IBM Corporation

The cache loader interface allows you to implement a custom Java class to load cache data from a back-end data store, and 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.

Map eviction

- Cache size is controlled by evicting objects when space is needed
- An evictor is an extensible object type for creating custom eviction schemes
- Some evictors are provided with WebSphere extended deployment
  - LRU (least recently used)
  - LFU (least frequently used)
  - TTL (time to live)
    - Can be based on creation time or last used time

© 2007 IBM Corporation

Cache size control is also customizable. WebSphere Extended Deployment provides evictor classes that can remove objects from the cache using least-frequently used or least-recently used policies when the cache reaches a certain size. It also provides a time-to-live based evictor for invalidating entries that have existed for longer than a set period of time. You can also write your own evictor class to manage the cache size based on custom criteria.

Keyword based eviction

- Map entries (key, value pairs) can be associated with one or more keywords
  - Simple mechanism for grouping entries
- Keywords can be associated with other keywords
  - Implicitly associates map entries with the associated keyword
  - Useful for creating nested groupings
- Application can evict all entries associated with a keyword
  - Includes nested keywords

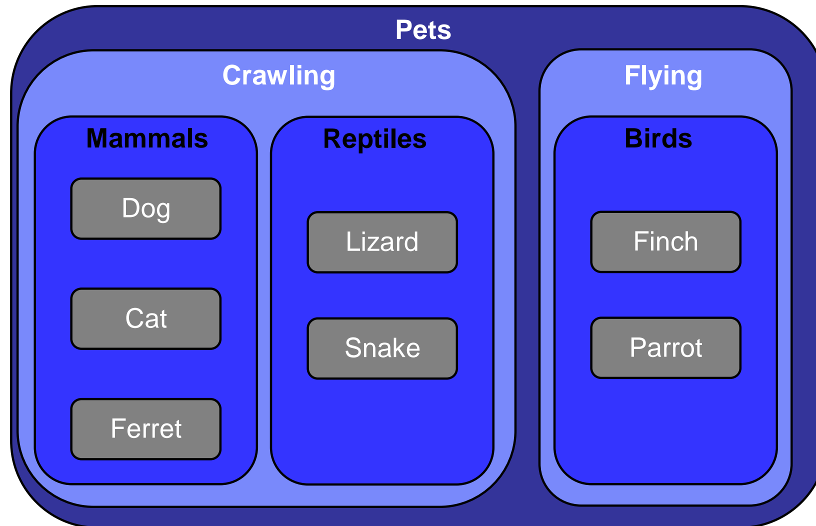
© 2007 IBM Corporation

Objects stored in an ObjectMap can be associated with one or more keywords, enabling simple grouping of entries within the cache. Keywords can also be associated with other keywords to create nested groups. These groups are useful because you can invalidate objects in the cache based on keyword. You can choose to invalidate all items associated with a particular keyword, which includes all nested keywords as well.

IBM Software Group

IBM

Nested keyword example



18

ObjectGrid programming concepts

© 2007 IBM Corporation

As an example, you might be storing pet records in an ObjectMap. You could group all of the pets according to their type, and then group the types by how they move about. In the example shown here, choosing to invalidate all records associated with the keyword "crawling" would invalidate five animals, because all of the animals in both the mammals and reptiles sets would be invalidated.

IBM

Section

ObjectGrid query

© 2007 IBM Corporation

ObjectGrid provides the ability to query data in the cache using traditional SQL queries, and the advanced capability of querying streaming data. This section will cover the query capabilities in ObjectGrid.

Single query

- Flexible query engine
  - Traditional ObjectMap POJO objects
  - Entity objects using the EntityManager APIs
- Select queries use same semantics as java persistence API's query API
  - Tailored for ObjectMaps
  - Named and positional parameters
  - First/max results
  - Single results
  - Iterate over multiple results

© 2007 IBM Corporation

ObjectGrid 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 same query language can be used to query both types of objects, but they are accessed using different API methods. In addition to the ability to access data stored in the cache the ObjectGrid query language provides advanced query capabilities such as data aggregation, sorting and grouping, and sub-expressions.

Programming model – Single query

- Select queries are created on the ObjectGrid session or EntityManager
  - `Session.createObjectQuery(String query)`
  - `EntityManager.createQuery(String query)`
- Query parameters (named and ordinal) are set with `setParameter()` methods
- Pagination support is available using the `setMaxResults` and `setFirstResult` methods
- Results can be retrieved using the `getSingleResult`, `getResultMap` and `getResultIterator` methods

© 2007 IBM Corporation

The query object is created through the session for object query or the EntityManager for entity query. The input is an SQL select statement. The select statement can contain replaceable parameters; before executing the query you must provide actual values for these parameters using the setParameter method. The query object provides pagination support by allowing you to specify which entry in the result set to return first, and the maximum number of results to return for this iteration. You can retrieve a single object, an ObjectMap containing just the results of the query, or an iterator over the result set.

Programming model – ObjectQuery example

```
// A transaction is required when the results are in an iterator or ObjectMap.
// The results are transaction-scoped.
session.begin();

// Use Join syntax to join resolve a relationship.
ObjectQuery q = session.createObjectQuery(
    "SELECT e.name, e.salary, d
          FROM EmpMap e JOIN e.deptid as d
          WHERE e.salary > :eSalary");

System.out.println("Rich Employees:");
q.setParameter("eSalary", new Double(20000));
Iterator iEmployees = q.getResultIterator();

while(iEmployees.hasNext()) {
    // Results are in an object array
    Object[] emp = (Object[]) iEmployees.next();
    String name = (String)emp[0];
    double salary = ((Double)emp[1]).doubleValue();
    DeptBean dept = (DeptBean)emp[2];
    System.out.println(name + ", " + salary + ", " + dept.name);
}

session.commit();
```


© 2007 IBM Corporation

This slide shows an example object query. The code creates a query object, sets replacement query parameters, and executes the query. The code then iterates over the result set. Note that the query results are only valid within the current transaction. EntityQuery is similar; the primary difference being that the query is created through the EntityManager rather than the session.

Stream query

- Allows querying over in-flight data in the ObjectMap
- Event driven
- Query results are updated continuously
- Supports ObjectMaps storing both POJO objects or Entity tuples
- Stream Processing Technology SQL
  - Similar to SQL92

© 2007 IBM Corporation

A stream query is a continuous query over data that is being continuously updated. Stream queries are similar to database queries in terms of how they analyze data. The difference is that stream queries are event driven, operating continuously on data as it arrives and updating the results in real-time.

The stream query engine used by ObjectGrid is called the **s**tream **p**rocessing **t**echnology (SPT) engine, and the SQL-like syntax that the stream query engine adopts is called **s**tream **p**rocessing **t**echnology **s**tructured **q**uery **l**anguage (SPTSQL). This is based upon the relational query language SQL. However, SQL was originally designed for snapshot queries and SPTSQL for continuous queries.

Usage examples

- Query the 50 stocks with the highest transaction volumes over the last 5 minutes
- Query the 10 stocks with the biggest gains in the past 30 seconds
- Query the average of 30 longest transaction time over the last 5 minutes
- Query the 10 longest average transaction time for each transaction type over the last hour

© 2007 IBM Corporation

When data involved in a stream query is inserted or updated into a map, the stream processing technology engine adds a time stamp to the data and inserts it into the associated stream. The stream contains historical data for the object, effectively creating a new map with the same key as the original map plus a timestamp. This allows you to create time-based queries and aggregate historical data for a single key.

Stream query concepts

- **Stream:** represents a stream of incoming data
  - An insertion or update to a stream ObjectMap generates a streaming message
  - A stream maps to an ObjectMap
- **View:** outputs of the SQL statements on streams
  - Refers to user interested stream queries or intermediate relations to support the queries
  - The view result is stored as a tuple in the ObjectMap
  - Results of intermediate views are not stored ObjectMap
- **Stream query set:** contains a set of streams and views
  - Serves as an containment object for the stream and view metadata
  - A stream query set is a deployment unit

© 2007 IBM Corporation

The stream processing technology engine introduces the concepts of **streams** and **views**. A stream represents a stream of raw incoming data and is the input to the stream query engine. There can be any number of streams and these streams can update with any frequency. Each stream is associated with an ObjectGrid stream map. Whenever there is an insertion or update to the ObjectGrid stream map, a stream event is generated.

The output from the SQL statements is called views. These SQL statements define the processing rules, and can be very complex. For example, you can join multiple streams, apply aggregation and computations, apply window operations, and group results.

A stream query set object serves as a containment object for the stream and view metadata. It consists of one or more streams and views, where views only use the streams and other views defined in this stream query set.

Example stream query

```
CREATE VIEW StockSummary AS SELECT * FROM
    (SELECT
      issue,
      AVG(price) AS avPrice,
      LATEST(price) AS currPrice,
      SUM(volume) AS totalVol
      FROM (SELECT * FROM StockTrades FETCH LATEST 2 MINUTES)
      GROUP BY issue)
    ORDER BY totalVol DESC FETCH FIRST 5;
```


© 2007 IBM Corporation

This example stream query defines a view named StockSummary derived from the StockTrades stream. At runtime, this view will track the average and current price, and the total volume of the five most actively traded stocks, based upon the activity during the most recent two minutes of trading.

Notice that the view specification applied three transformations to the stream StockTrades: Narrowing the original stream to a sliding window of the last two minutes of trades; computing the average price, current price, and total volume for each group of trades having the same issue; and selecting the five issues having the highest total volume.

Summary

- ObjectGrid instances are created programmatically and can be configured programmatically or using XML files
- ObjectMaps can be used like a standard map, with transaction support
- ObjectGrid features can be customized by implementing custom Java classes
  - Cache loading, invalidation, and more are extensible
- Advanced query support

 © 2007 IBM Corporation

In summary, ObjectGrid instances are created within your application code, optionally based on XML configuration files. ObjectMaps are used to hold cached objects, and work like a standard map, with the added benefit of transaction support. Many ObjectGrid features can be customized, and the ObjectGrid provides many advanced query capabilities.

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/

© 2007 IBM Corporation

This presentation only scratches the surface by introducing some of the more common top level programming concepts. To continue learning about writing ObjectGrid applications you should explore the ObjectGrid samples, which can be found in the “optionalLibraries” directory after installing WebSphere Extended Deployment data grid. The included samples guide will walk you through the provided sample code, which is well documented. Javadoc is also provided for the ObjectGrid interfaces in the “web” directory after installing WebSphere extended deployment. A comprehensive ObjectGrid programming guide is available on the DeveloperWorks Web site.

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_Programming_Concepts.ppt

© 2007 IBM Corporation

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

EJB, 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.

© Copyright International Business Machines Corporation 2007. All rights reserved.

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.