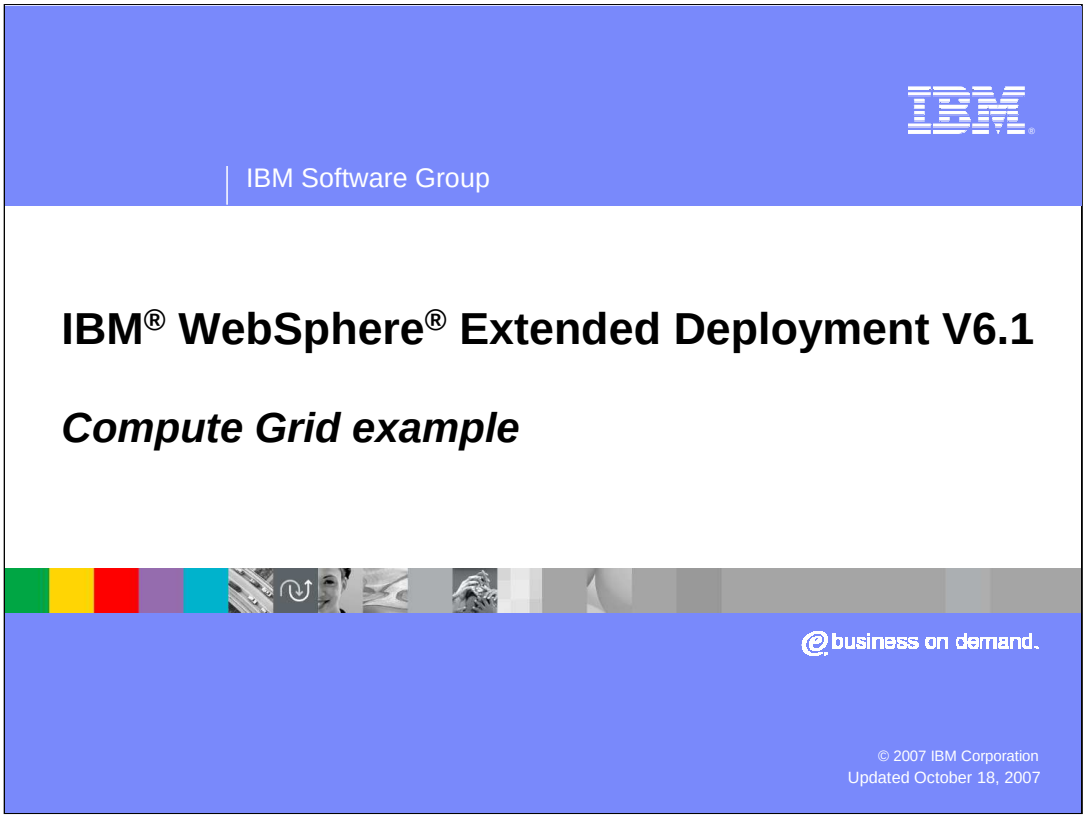


This presentation will provide an example of how to use the compute grid component offered in WebSphere Extended Deployment V6.1.

Agenda

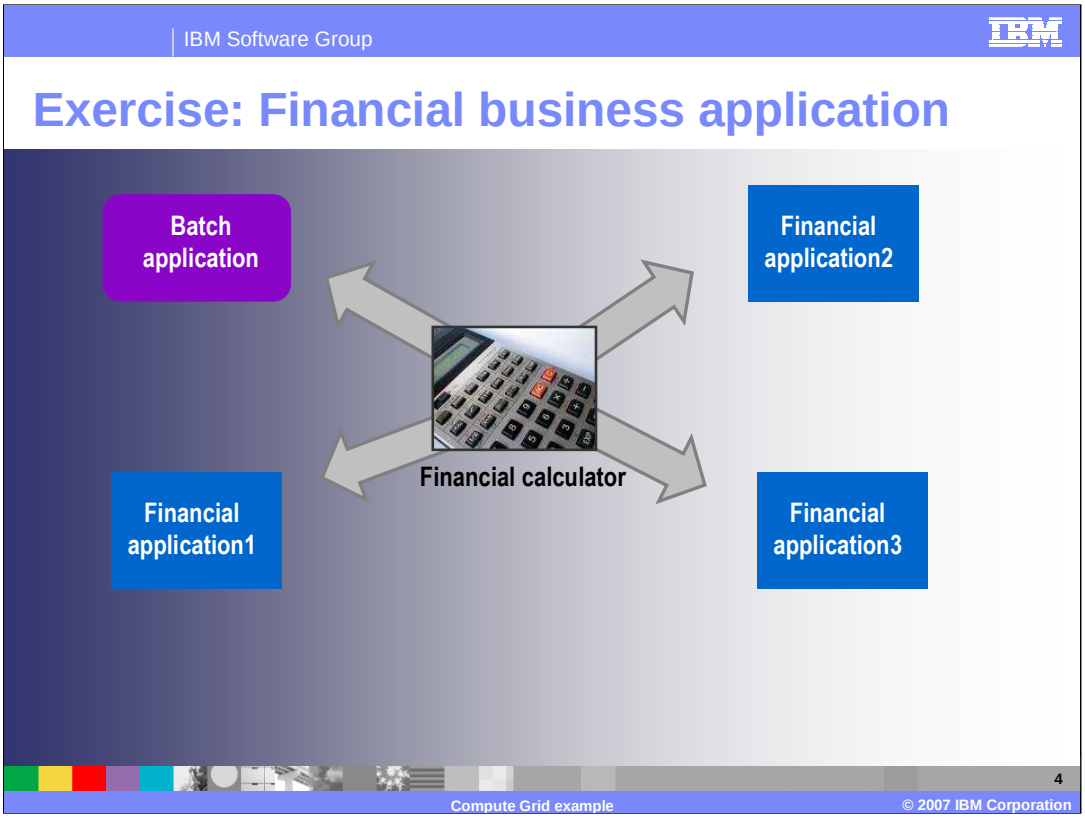
- Compute Grid exercise
- Implementation of the solution

This presentation will provide an example of how to configure your WebSphere environment to support the compute grid and explain how to deploy long-running applications to the environment.

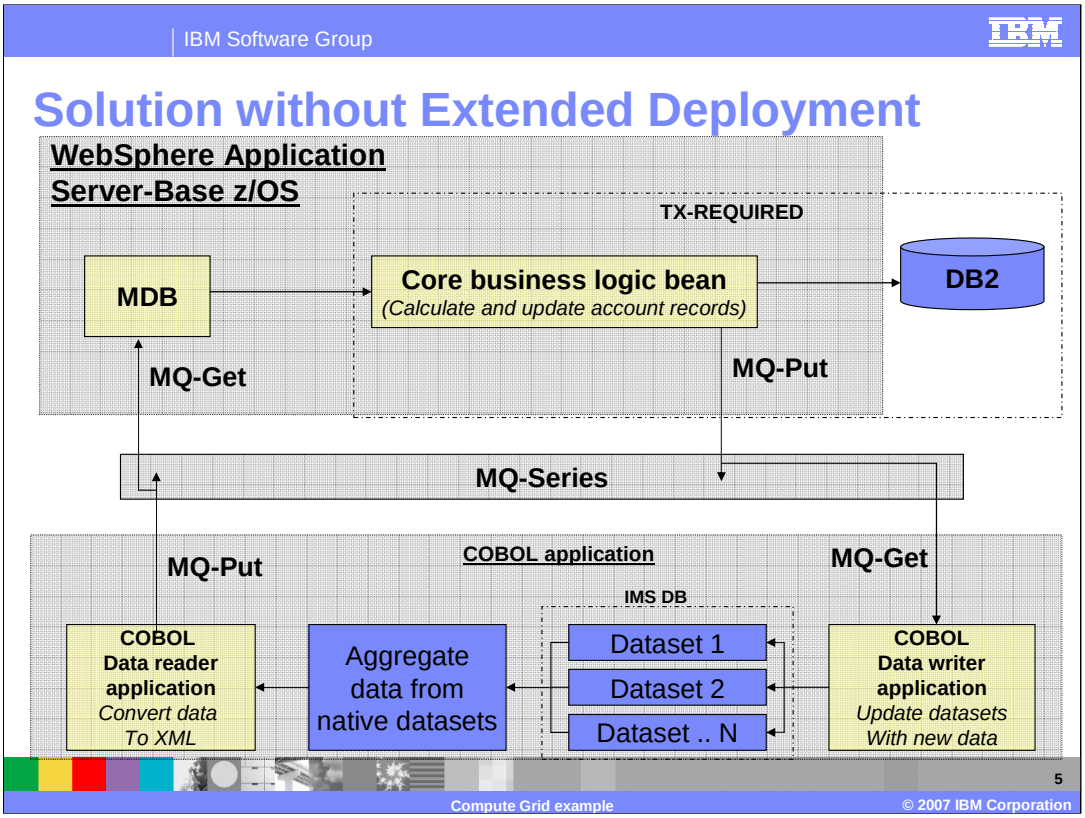
Section

Compute Grid exercise

The following slides contain an example to demonstrate an effective application of Compute Grid functionality.



The slide presents a problem you might want to solve on a z/OS® system. Several types of applications (retirement modeling, student loan forecasting, and others) access some common set of functions, for example a financial calculator libraries. In this example, the Financial Calculator application is a ‘Kernel’ application that must be available to numerous other banking applications, including asynchronous batch-type applications that perform tasks such as calculating interest and credit scores. The goal of the exercise is to use just one implementation of the financial calculator. The financial application is implemented as J2EE. However, it must also be accessible from existing batch applications. The data that the financial calculator must process asynchronously resides in EBCDIC in an IMS™ database on the mainframe. This data must remain on the mainframe to take advantage of the robust security, high-availability, and scalability features of z/OS and WebSphere on z/OS. How do you asynchronously process this native data while reusing the financial calculator application?



There are two ways to solve this problem. The first way is for you to generate code to connect WebSphere and classical applications. The second way is to take advantage of batch programming in WebSphere Extended Deployment. This presentation presents both approaches to highlight the differences.

This slide shows an example of the first solution: an elaborate design pattern based on WebSphere MQ that provides some batch-like behavior. The steps in this solution follow the arrows in a clockwise direction, beginning with the bottom center of this slide. First, aggregate the data to be processed from the various IMS datasets. This data then flows into a COBOL application which converts each record of the IMS data to ASCII and then to XML and places it on the message queue. This application submits the converted IMS data record to WebSphere using message-driven beans and WebSphere MQ messaging, as shown in the upper left of this slide. From here the WebSphere framework notifies a message-driven bean within WebSphere to retrieve that message from the message queue and passes that message on to the business logic. The business logic then converts the XML into a Java™ object and passes the Java object to a kernel bean and performs the financial calculations on it. After performing the calculation, the business logic converts the Java Object back to XML and pushes the XML back onto the message queue. The WebSphere framework then notifies the native application and persists the message back into the IMS database.

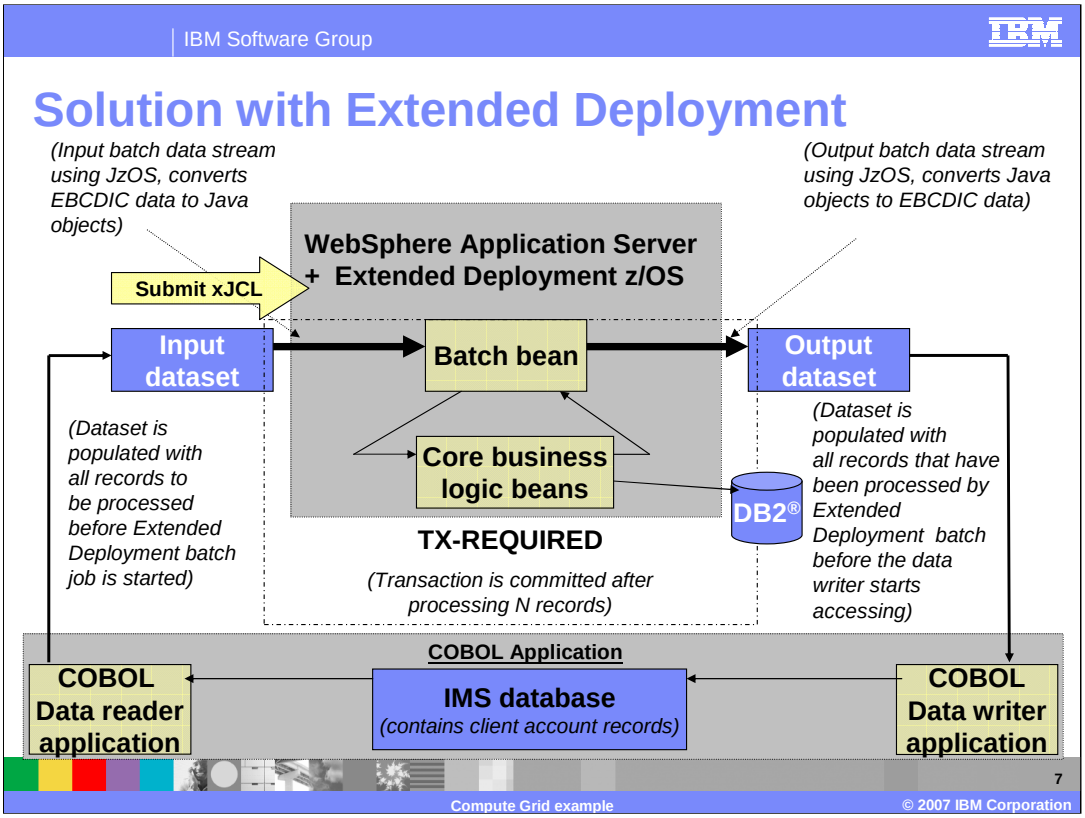
Overhead without Extended Deployment

- EBCDIC to ASCII to XML conversions
- Global transactions do not span the entire process
  - Requires processing a single record at a time
  - Checkpoint
- Complex code base that is not flexible

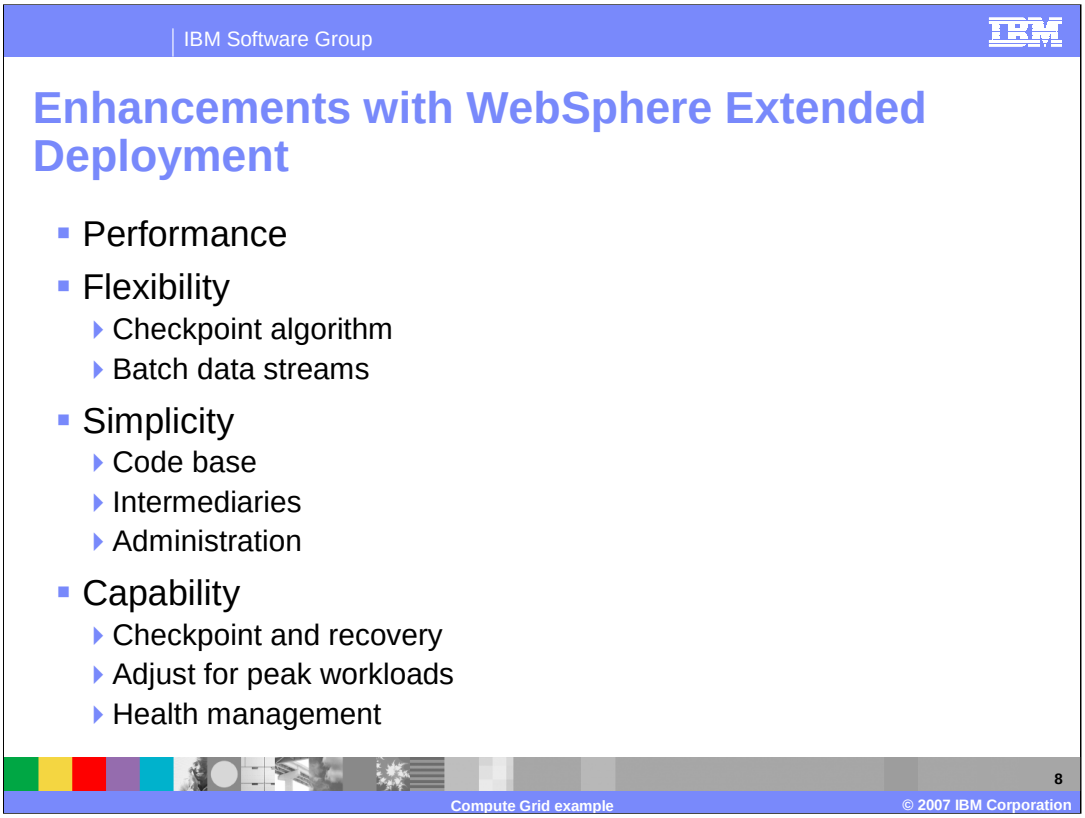
The solution presented on the previous slide is complicated and the overhead of the infrastructure is high, relative to the actual business processing for each record.

First, there are two conversions performed between EBCDIC, ASCII and XML and these types of conversions are very expensive. The initial conversion translated data from EBCDIC to ASCII to XML in order to pass the data into the WebSphere framework. The second conversion translated the processed data back into EBCDIC. In this example, these conversions are made in the COBOL application; however JzOS could offer a potential performance improvement by using Java to make the translation which could run on a zAAP – a zSeries application assist processor.

JzOS supports neither global transactions - which span the entire process - nor unified checkpoints. In particular, a global transaction is required to place a record onto the queue. Then, a separate global transaction retrieves the record from the queue by way of a message-driven bean. After processing, another global transaction is used to place the resulting data onto the queue and a process outside of WebSphere requires a global transaction to read the data from the queue. This process might initially be the most obvious approach, but it is complicated, inefficient and difficult to maintain.



WebSphere Extended Deployment batch provides a J2EE-centric batch processing environment. This environment allows the business logic to directly access native data and, therefore, eliminates much of the infrastructure overhead introduced in the WebSphere MQ-based solution. This solution requires you to create two JzOS Batch Data Streams: one for input, one for output, and a batch step bean with a processJobStep(). The method processJobStep() logic retrieves a record from the JzOS batch data stream and invokes the financial calculator with the retrieved record. The framework in WebSphere Extended Deployment repeatedly invokes the processJobStep() of the batch step bean in a loop. Upon each iteration of processJobStep(), the batch step bean creates a Java Object that represents the data retrieved from the input JzOS Batch Data Stream and passes that object to the financial calculator for processing. The processJobStep() logic takes the output of the financial calculator and persists the data to the output MVS dataset through the output JzOS batch data stream and returns to the Extended Deployment framework. The Extended Deployment framework continues this cycle until the processJobStep() method has processed each input record, the output has persisted, and the job has returned a process completed result to the framework.



The WebSphere Extended Deployment solution provides several enhancements over the previous solution.

Most of the MIPS consumed are zAAP-eligible therefore significantly reducing the overall cost of processing a single record. Performance data collected at a test site showed that this solution reduces both the MIPS and processing time roughly by half.

This solution allows more flexibility in adjusting the checkpoint algorithm without the need to modify any application code. This, in turn, affects the transaction scope and resource locking schemes. Additional flexibility is afforded through the ability to change data sources by defining new Batch Data Streams and through the isolation of application changes to specific sections of the code.

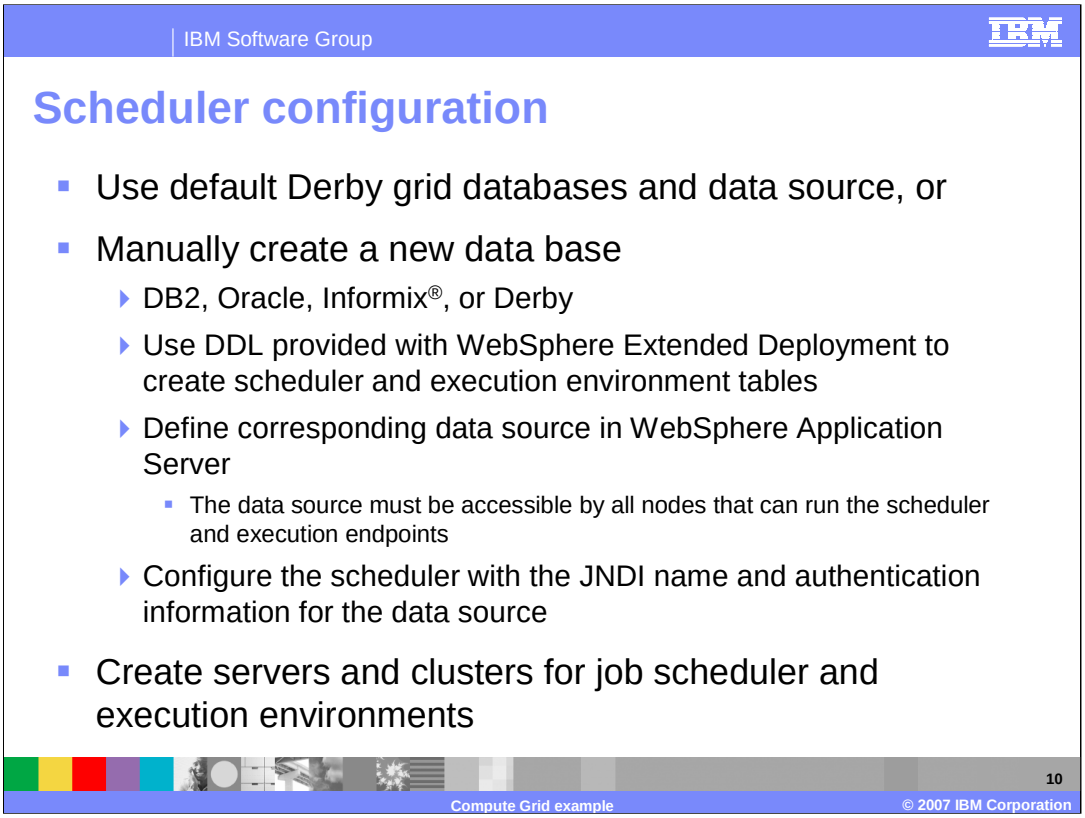
The solution also simplifies your implementation. The amount of code you have to produce and maintain is reduced by eliminating the need for intermediaries to massage the data format. Compute Grid is able to parse the raw data through the batch data stream definitions and convert that raw data straight to a Java-Object, as opposed to converting data to XML, to ASCII, and so on. Your system administration of the batch environment is simplified, since Compute Grid integrates with WebSphere Administration Console and Tivoli Process Manager to monitor batch jobs running in the system.

WebSphere Extended Deployment enhances this solution through the capability to recover from the last check-pointed position in case of system failures, such as temporary network or database failures. In the case of resource contention, you can temporarily suspend or cancel jobs. For example, if an unexpected peak in online workload occurs, Compute Grid can suspend work and use the server for online-transaction processing. You can assign service policies to jobs; for example give higher priority to platinum customer batch workloads. You can quickly create a conditional flow of job steps using xJCL. Finally, you

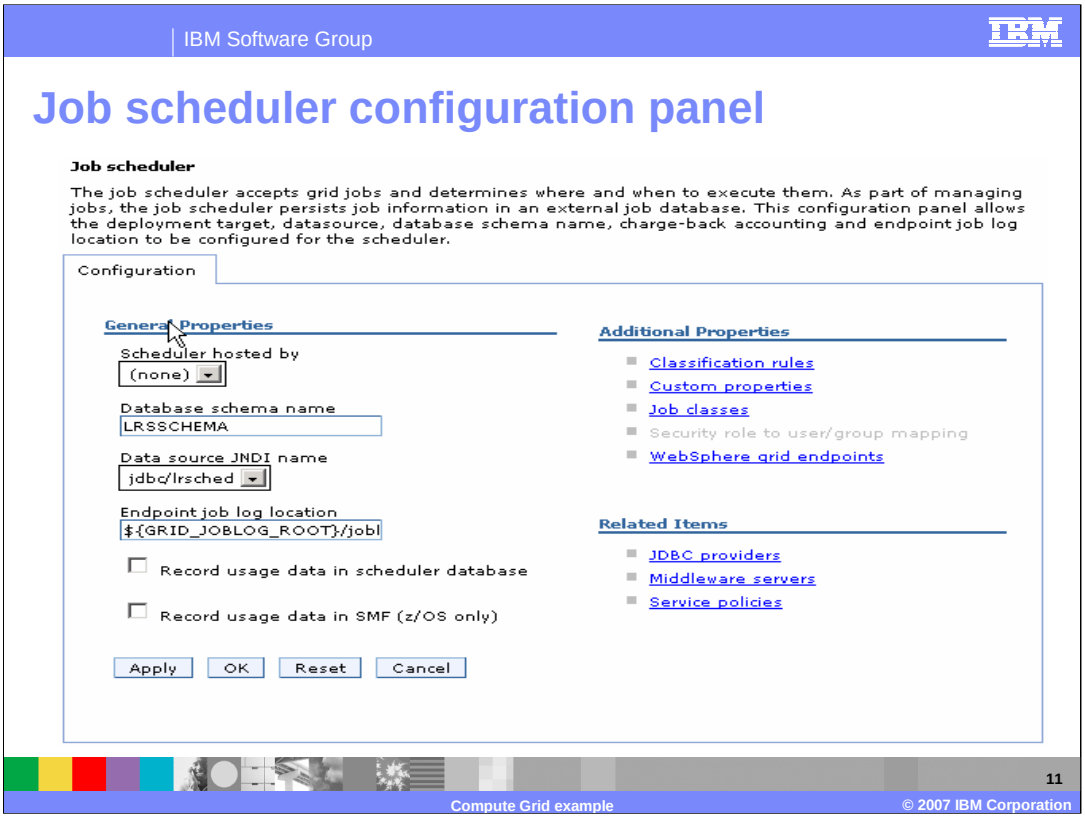
Section

Implementing the solution

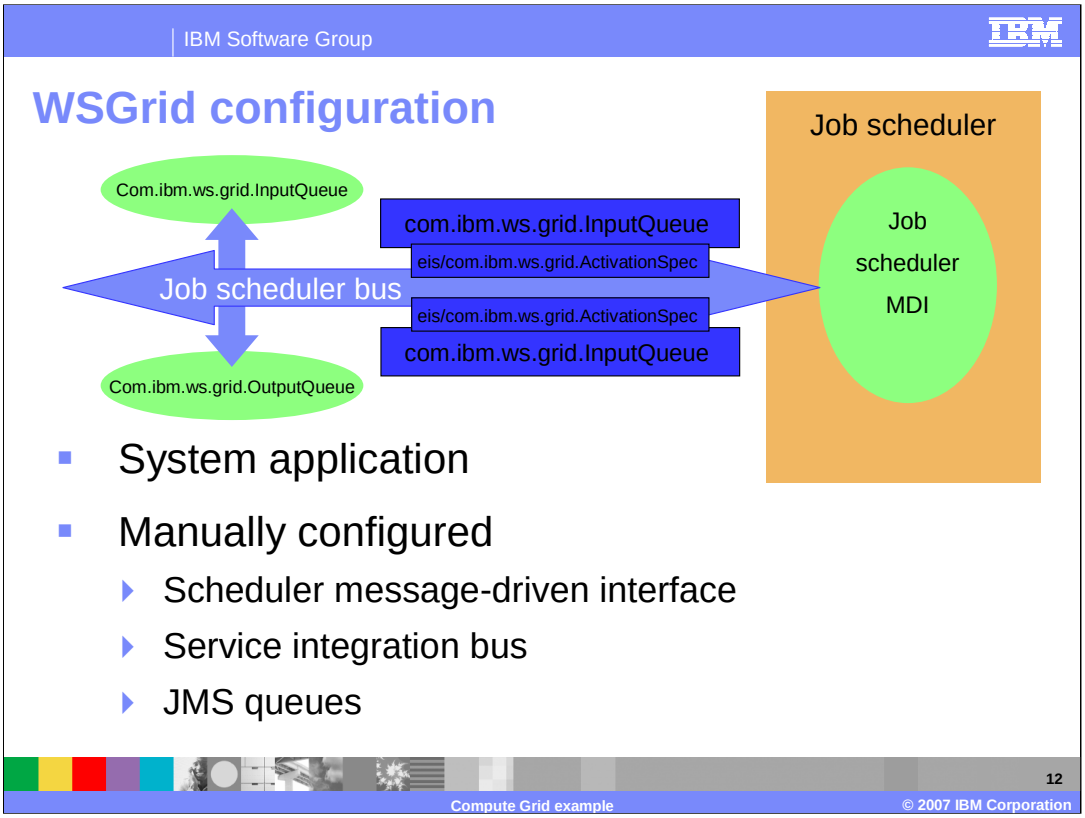
The remaining slides present many details of the various pieces required to implement a batch grid job like you just saw.



Several steps are required to create an environment that will support long-running applications. First you must create the databases for the scheduler component and for the grid execution environment; DDLs to accomplish this are provided with WebSphere Extended Deployment. If you choose use Derby, the database instances are created automatically when the runtime components are installed. Once the tables are created, you must define corresponding data sources in WebSphere. The data sources must be accessible to any nodes that will host either the scheduler or run long-running applications. The scheduler must then be configured with the JNDI name for the resource and any security information it may need to use the data source. The deployment of the job scheduler and the grid execution environment, if applicable, is automatic.



The Job Scheduler configuration panel is located under System Administration in the left frame of the administration console. Under the configuration panel for the scheduler you can configure where the scheduler will reside, schema used in the database, and the JNDI name for the data source. You can also specify classification rules and the Job Classes (resource restrictions) as discussed in the compute grid overview presentation.



The reason WSGrid and the Grid Scheduler are system applications is so that WebSphere service can be applied to them when necessary. WebSphere service cannot be applied to Enterprise application, since they are copied to locations outside of the WebSphere product directory. In the current release of WebSphere Extended Deployment, configuring WSGrid is optional and you have to deploy and configure the application. WSGrid uses the WebSphere enterprise service bus for communications. There are three steps you must complete to configure WSGrid as a participant of the service bus.

Developing long-running applications

- Develop long-running applications using normal development tools
  - J2EE
    - WebSphere Application Developer
    - IBM Rational® Application Developer
    - Eclipse
  - Java
  - External programs
    - Compiled (FORTRAN, COBOL, …)
- A J2EE or Java transactional batch long-running application can be packaged in an ordinary EAR file deployed to a WebSphere Application Server

Application developers create long-running applications based on either the computationally intensive or batch programming model using normal J2EE development tools. Long-running applications are packaged into normal ear files. The EJB jar file for the application contains some specific information for long-running applications, such as the deployment information for the controller bean and the actual bean implementations of the applications.

Deploying long-running applications

- WebSphere
  - Long-running applications are deployed as regular J2EE applications
  - When the application is deployed, WebSphere External Deployment automatically detects that it is a long-running application
  - Install process will silently install the grid execution environment
  - Can mix the transactional and grid applications in a dynamic cluster
  - The same placement controller is used for both types of workload
- Once the application is deployed, define service policies for the new long-running application

Long-running applications are deployed like any regular J2EE application. During the deployment process WebSphere Extended Deployment will detect that it is a long-running application and silently install the grid execution environment if it is not already installed. Once the application has been deployed an administrator can define service policies for the application in preparation for submitting a job. Long-running applications do not support the same service policies as OLTP application. The only metrics supported for long-running applications are completion time and discretionary.

Submit job

- Construct xJCL to submit job to application
- Use one of the interfaces provided by the scheduler to submit job
  - Job management console
  - Command line interface
  - Web service
  - EJB
  - WSGrid utility
- Note job ID assigned by scheduler
- Use job management console

Prior to submitting a job, an administrator must construct an xJCL document to describe the behavior of the application or an equivalent WSGRID properties file. For WebSphere applications the xJCL contains a JNDI name to identify which application should be used for a job step. For a non WebSphere application the xJCL specifies how to run the application, for example is it a program or a script; and parameters that are passed to the application. Then the administrator has a choice of interfaces to submit the job to the scheduler. Regardless of the interface used, the return value from the submission is the job ID assigned to the job by the scheduler. An administrator can then manage the job using panels in the job management console or any of the other available interfaces.

Submit job with WSGrid from USS

- Command line utility
  - Unix system services shell script
  - Three forms:
    - WSGrid.sh <job properties file>
    - WSGrid.sh <control properties file> <job properties file>
    - WSGrid.sh <control properties file> <xJCL file>

There are a few options when starting the WSGrid shell script. As shown here, the control file contains information about where the deployment manager is running, user ID and password. Besides the standard xJCL type of information, the job properties file can also contain a reference to a job stored in the WebSphere Extended Deployment job repository. The xJCL for the grid application can come from three places. First it can be stored in the job repository and called out by a ‘repository-job’ command in the job properties file. Second, the path to a file containing the xJCL can be passed as a parameter. Finally, the job properties file can contain the xJCL information. The format of both the control and job properties file is similar to standard UNIX, value pairs separated by an equal sign.

Submit job with WSGrid from MVS

- JCL

  //Job Card

  // MVS batch job steps

  // WSGrid step

  // MVS batch job steps

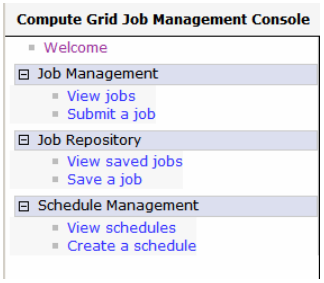
As mentioned in the grid programming model presentation for z/OS, WSGrid steps can be mingled with traditional batch steps in an MVS batch job. The data stream from any one step, traditional or transactional, can be passed into a follow on step.

Sample WSGrid job step

```
//* Use JzOS sample proc JVMPRC14 or JVMPRC50
//*
//JAVA EXEC PROC=JVMPRC50,
// JAVACLS='com.ibm.ws.bootstrap.WSLauncher'
//MAINARGS DD *
com.ibm.ws.grid.comm.WSGrid
//*******************************************************************************
//WGCNTL DD *
< WSGrid control properties >
//WGJOB DD *
< WSGrid job properties>
# specify optional batch data stream
bds.myinput=com.ibm.websphere.samples.TestBatchDataStream
# specify optional batch data stream properties
bds-prop.myinput.FILENAME=/batchsimulator/testdata/testbds.data
//STDENV DD *
<Java properties>
//
```


This slide presents the general layout of a WSGrid job step. This sample is using JzOS as indicated by the “exec” statement. You can also see how the WSGRID control and job properties are input. There are many job properties that you input to WSGrid, although this example only shows lines for a batch data stream. The Java properties shown here are basically the same as the samples delivered with JzOS.

Job management console



- Web interface
  - Job management
  - Job repository
  - Job scheduler

A key interface to manage grid jobs is the job management console. The job management console resides in the Job Scheduler system application, and has a browser interface which is independent of the WebSphere administrative console. The job management console has three sections.

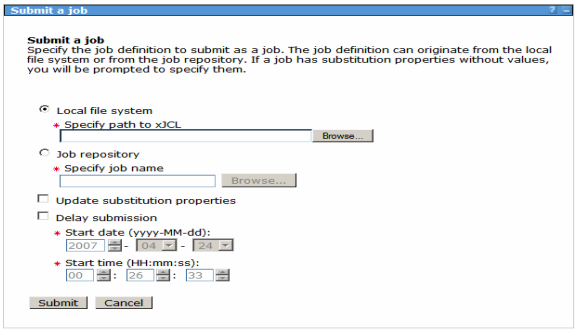
The Job Management section is used to submit a job to run or manage submitted jobs.

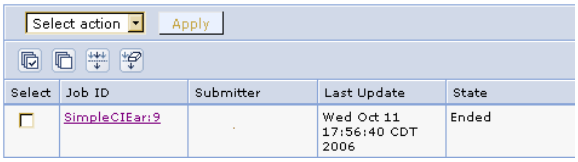
The Job Repository saves job definition xJCL into the Job Scheduler data base. Saved jobs can be viewed from the Job Repository panels.

Finally the Job Scheduler allows jobs to be scheduled for running at a future time or for running periodically.

Job management console: Job management

- Submit a job
  - xJCL
  - Parms
  - Schedule
- Manage submitted jobs
  - View job logs
  - Cancel
  - Resume
  - Restart
  - Stop
  - Suspend

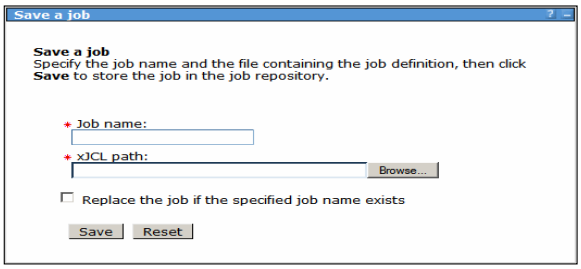




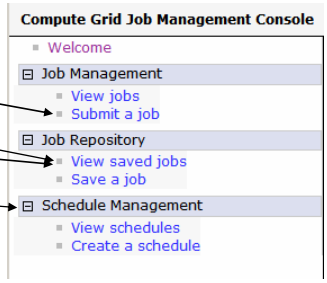
Job submission requires pointing to xJCL for the job. The xJCL contains information required to run the applications in the job. The xJCL also contains descriptions of parameters (for non WebSphere applications) or other environmental settings. You can optionally specify when the job will run, similar to the Job Schedule section of the job management console. Once a job is submitted, it can be viewed and managed in the "View jobs" panel. The state of a job is shown (submitted, running, paused, ended). Also shown is the node and application server where the job ran. Clicking on a job will show details of the job's execution and provide an opportunity to view the job logs. From this panel you can also manage the job during its life cycle.

Job management console: Job repository

- Save a job definition
  - xJCL



- Manage saved jobs
  - Submit
  - View xJCL
  - Delete definition
  - Schedule

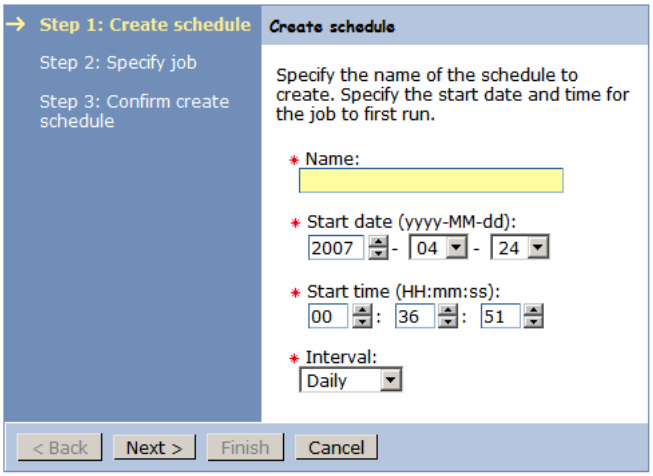


Saving a job to the job repository requires a name and a path to the defining xJCL. Once saved, the xJCL can be viewed by clicking on the job name. You can also replace or remove xJCL job definitions already in the job repository.

You can submit the saved job from the “Submit a job” panel, or schedule it for periodic submission on the “Create a schedule” panel.

Job management console: Job schedules

- Save a job definition
  - xJCL
  - Schedule
    - Date and time
    - Repeating
- Manage schedules
  - View details
  - Cancel



Scheduling a job is similar to submitting job by its xJCL or a job from the repository for delayed submission. You can schedule a job to be submitted once at a give date and time from the “Submit a job” panel or you can schedule the job to run on a periodically. From the “View schedules” panel you can view the details of the job schedule or cancel the schedule.

Summary

- WebSphere Extended Deployment provides an environment for managing and running batch-style and compute-intensive applications
  - Jobs are scheduled using the grid scheduler (LongRunningScheduler.ear)
  - Jobs are run in the long running execution environment (GEE.ear)
- A WebSphere Extended Deployment compute grid can dynamically balance the needs of long-running work against the needs of transactional applications within a cell

In summary, this presentation showed by example how to use the new compute grid component provided with WebSphere Extended Deployment V6.1.

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_XD61z_ComputeGrid_Example.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:

DB2 IBM IMS Informix Rational WebSphere z/OS

Rational is a trademark of International Business Machines Corporation and Rational Software Corporation in the United States, Other Countries, or both.

EJB, J2EE, Java, 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.