A CICS to Linux Grid Implementation

This IBM® Redpaper describes an implementation that routes CPU-intensive work in an application from IBM CICS® on z/OS® to a grid Linux environment and returns the result back to CICS. Grid computing is a technique that attempts to boost the collective power of networked computers by distributing portions of complex operations.¹

We defined a grid of computing resources and ran part of a CICS application successfully on the grid. We thus showed a way to gain more flexible deployment and workload growth without a major application rewrite and were able to take advantage of capacity in the grid to manage system growth. In addition, CICS PRJVM significantly improved the performance of the application. (For information about PRJVM, see “Trusted middleware classes in persistent reusable JVM” on page 6.)

Background and objective

Our example features a Human Resources (HR) outsourcing and consultant firm. They provide human capital management services to their clients, who are companies from a wide range of industries. As part of this firm's pension management service, pension valuation is done through an application called CalcEngine.

CalcEngine valuates pension according to client constructs and policies defined by corporate and government boundaries. CalcEngine is implemented with the Smalltalk language. IBM supports the development and testing of this language with the VisualAge® Smalltalk product family. Client-exclusive interpretive language worksheets are created during CalcEngine execution.

Currently, multiple instances of CalcEngine run in multiple CICS Application Owning Regions (AORs) in z/OS. After Smalltalk is initialized, the valuation is immediately calculation-intensive, as the Smalltalk application evaluates all of the options in the Client's plan. Depending on the type of request, the processing can consume a large amount of zSeries® CPU resources. To reduce the cost of the pension valuation service and enable future business growth, the firm looks for a new, flexible way to deploy CalcEngine. That is the objective of this proof of concept (POC).

¹ American Banker, as quoted on http://www.datasynapse.com/kc/wp.jsp.
Grid computing

Before we describe our environment, let us define grid computing: It can be viewed as distributed computing taken to the next level. The goal is to create a simple yet large and powerful self-managing computer out of connected systems that share resources. The telecommunication philosophy that created the Internet explosion led to an emerging standardization for sharing resources. Along with that, the availability of higher bandwidth is now driving another large step called grid computing.

Some of the benefits that people receive from grid computing include:
- Exploiting underutilized resources
- Parallel CPU capacity
- Access to additional resources
- Resource balancing
- Reliability
- Manageability

Some of the components that make up a grid include:
- Computation
- Storage
- Communications
- Software and licenses
- Special equipment and architectures
- Jobs and applications
- Scheduling, reservations, and scavenging

These and other details about grid computing can be found in Fundamentals of Grid Computing, REDP-3613, at http://www.ibm.com/redbooks.

Architecture

Now we proceed with a description of the architecture we used in our example.

POC architecture

The POC architecture moves the CPU-bound Smalltalk CalcEngine from CICS to a Linux grid environment while leaving the rest of the functionality in the CICS region on z/OS. A request to CalcEngine is forwarded from CICS to the Linux grid environment. After calculation, the result is sent back to CICS. The original deployment is illustrated in Figure 1, and the new deployment is shown in Figure 2 on page 3.
The job-scheduling software that we chose was LiveCluster from DataSynapse (see http://www.datasynapse.com). This name has changed recently; the software is now GridServer by DataSynapse, but we will continue to use the names that were used when the project began. LiveCluster is a distributed computing platform for grid computing that creates a virtual pool of compute resources over an existing hardware and network infrastructure. Compute-intensive, data-intensive, or throughput-intensive applications can be distributed across the pool to achieve increased performance and reliability. Figure 3 on page 4 is a LiveCluster architecture chart from the DataSynapse LiveCluster 3G Administration Guide.

There are three cluster components in LiveCluster: drivers, engines, and servers.

► **Drivers** act as agents for client applications requesting work from the cluster. Client applications that integrate with LiveCluster on the application programming interface (API) level embed an instance of a driver in the application code. One type of client can be a Web services client.

► **Engines** are resources LiveCluster uses to perform work. LiveCluster Engines report to the Server when they are available. After logging in and updating code, they accept work assignments, run tasks, and notify the Server when results are ready. A single LiveCluster deployment can incorporate engines on multiple operating systems and with various hardware configurations. Each LiveCluster Engine consists of one Engine Daemon and one or more Engine Instances. The Engine Daemon manages Engine Instance runtime and maintains connectivity with the Server. Tasks are executed in Engine Instances.

► The **Server** is the central controlling mechanism of the cluster. A LiveCluster Server consists of one Director and one or more Brokers. Directors route Engines and Drivers to
Brokers, and manage Brokers and Engine Daemons. Directors balance the load among their Brokers and provide Broker fault tolerance. Brokers schedule work and manage state. Brokers monitor Engine Instances and re-queue work assigned to failed Engines. Brokers also maintain connectivity to Drivers and coordinate providing results to client applications. With the name changes from DataSynapse, this is the most confusing. The Server is now called the Manager, with the other components included.

![DataSynapse LiveCluster architecture chart](image)

We mapped our POC architecture (shown in Figure 2 on page 3) to these three cluster components:

- The **Driver** is the client code running inside CICS on z/OS.
- The **Engines** are multiple Linux images that execute the Smalltalk.
- The **LiveCluster Server** is a separate Linux system that includes a director and multiple brokers.

In LiveCluster, a server process (not to be confused with the Architecture component called the Server) that fulfills a client application’s request for work is called a Service. LiveCluster supports two types of Services, Jobs and Cluster Services. Job is appropriate for computation that can be logically broken down into many units of work and then the result is combined through these separate executions. Cluster Service is appropriate for requests that will benefit from distribution. For Job or Cluster Services, a unit of work is called a task.

### CICS to grid implementation

In our prototype, it is better to use a Cluster Service to implement DataSynapse service instead of breaking the CalcEngine application into several pieces. There are two main reasons for this: First, the current response time is acceptable by end user, and the focus of the new implementation is to resolve CPU growth issues. Second, the calculation depends on a Smalltalk interpretation of worksheets and serialized objects. Therefore, dividing the unit of work into smaller pieces and re-assembling the results can cause high processing overhead and require a significant rewrite of code.
Currently, CalcEngine data for pension calculation includes PersonData and Corporate EnvironmentData. PersonData is passed into CalcEngine as request data. Based on the request, CalcEngine looks up and loads specific EnvironmentData stored in VSAM. In the new architecture, EnvironmentData is stored in DB2® instead of VSAM. In either VSAM or DB2, EnvironmentData is updated by a separate application run as a job in z/OS. During the life cycle of a CalcEngine, EnvironmentData is cached after being loaded into CalcEngine unless an update notification is received. Notice of an update causes a cast-out of the specific EnvironmentData to occur.

As shown in the architecture chart in Figure 2 on page 3, we kept the TBA application in CICS and moved CalcEngine to Linux on xSeries® without too much code modification. Two pieces must be implemented: driver (also called client) and service.

**Driver implementation**

The client makes Web services requests to the LiveCluster director. The driver API is built on Apache application exchange for information systems (AXIS), which is an implementation of the Simple Object Access Protocol (SOAP). The driver runtime requires the AXIS library Java Archive (.jar) files. In the proof of concept, the first step is to build the Java SOAP client interface for the director. This is done with the Apache AXIS WSDL2Java tool, which translates a Web Services Description Language (WSDL) of a Web service's interface into Java code. The WSDL for LiveCluster is obtained from http://[server]:[port]/livecluster/webservice/ClusterService?wsdl. The command to generate code is java org.apache.axis.wsdl.WSDL2Java <wsdl_filename>. Eleven Java classes in two different Java packages are generated through this step and they are specific to our broker host system: hostname etpserv26 and port 8000.

The Driver runs in CICS and makes service requests to LiveCluster. The Drive code is developed using JCICS API, LiveCluster API, and Java classes generated by WSDL. The code fragment is shown in Example 1. Packages clusterServiceInterface and clusterServiceStub are generated from WSDL. ServiceData, LoginReply, ClusterServiceLocator, and ClusterService are Java classes generated from WSDL.

PersonData is obtained from the CICS CommArea as a byte array. Its codepage is EBCDIC and it contains both text and binary data. The driver code and the service code do not handle the codepage conversion because CalcEngine has the conversion and parsing logic built in. Through Smalltalk services, the codepage casting can be accomplished through the language. Application extensions were created to make the common source handle the appropriate byte array, based on location of execution.

To access the Cluster Service, the client first must create an instance of ClusterService and call the login method to establish a session. Then the request is driven by the execute method with a session ID, a service method, and input data as arguments. For login, a Secure Sockets Layer (SSL) can be used to encrypt user ID and password. DataSynapse also supports using the SSL for data transport. But SSL was not used in this project.

**Example 1  CalClnt.java (Java code in CICS)**

```java
import clusterServiceInterface.*;
import clusterServiceStub.*;
import com.ibm.cics.server.*;
public class CalClnt {
    public static void main(CommAreaHolder cah) throws Exception {
        byte[] personData = cah.value;
        ...
        try {
            // set service name, user ID and password
```
String uid = "prod";
String password = "password";
String svcName = "RealCalcService.java";
ServiceData input = new ServiceData();
ClusterService service = (new
ClusterServiceServiceLocator()).getClusterService();
LoginReply reply = service.login(uid, password, null);
ServiceData initialData = new ServiceData();
initialData.setStr("foo");
String serviceId = service.createService(reply.getSessionId(),
"RealCalcService.java", initialData, null,
null);

// set input
input.setBytes(personData);
// execute calc, execute getResult() method of RealCalcService class
ServiceData response = service.execute(reply.getSessionId(),serviceId,
"getResult", input,);
byte[] respFromvast = response.getBytes();
cah.value = respFromvast;
...
} catch (Exception e) {
  e.printStackTrace();
}

Trusted middleware classes in persistent reusable JVM

The Cluster Service login and session creation for each transaction degrades performance. In this application, it does not cause concern to reuse sessions so we can take advantage of the Trusted Middleware Class (TMC) feature of the persistent reusable Java virtual machine (PRJVM).

PRJVM is a technology that is mostly used in a transaction environment. Multiple JVMs share the same address space, but each runs only one transaction at a time to ensure transaction isolation. The set of JVMs in the same address space, called a JVMSet, shares a common system heap of system classes and other shareable classes. This significantly reduces the time needed to start a new JVM in the JVMSet, because the majority of system classes are already loaded in the system heap. It also reduces the overall memory footprint for these classes because they are loaded only once per JVMSet and not once per JVM. Java classes loaded in PRJVM include system classes, TMCs, and application classes. Trusted middleware is trusted by the JVM to manage its own state across a JVM reset by using the Tidy-Up method at the end of each transaction. References from middleware classes to application classes are nulled out to ensure that the JVM reset does not fall into expensive garbage collection operations.

To significantly improve transaction throughput, we can separate the service session creation and service execution operations as shown in Example 2 and Example 3 on page 7. The RealCalcClusterService class creates a static instance of ClusterService. The CalcClnt class reuses the static ClusterService instance and keeps it in trusted classpath. For detailed setup information, see Note 4 in “References” on page 21. PRJVM is also supported in IMS™ Version 7.1 and DB2 Version 7.1. Similar functionality should be available but was not verified during this project.

Example 2  RealCalcClusterService

...
public static ClusterService realCalcClService;
public static String serviceId;
public static LoginReply reply;
public static synchronized void createRealCalcClusterService() throws Exception {
    if (realCalcClService == null) {
        try {
            realCalcClService = (new ClusterServiceServiceLocator()).getClusterService();
            reply = realCalcClService.login("prod", "password", null);
            ServiceData initialData = new ServiceData();
            initialData.setString("foo");
            serviceId = realCalcClService.createService(reply.getSessionId(),
                "RealCalcService.java", initialData, null, null);
        }
        catch (Exception e) {
            throw (new Exception("RealCalcClusterService create exception:"+e.getMessage()));
        }
    } // end if
}

public void destroyRealCalcClusterService() throws Exception {
    try {
        realCalcClService.destroy(reply.getSessionId(), serviceId);
        realCalcClService= null;
    }
    catch (Exception e) {
        throw (new Exception("RealCalcClusterService destroy exception:"
            + e.getMessage()));
    }
}

Example 3  CalClnt.java (modified)
import RealCalcClusterService;
public class CalClnt {
    public static void main(CommAreaHolder cah) throws Exception {
        byte[] personData =  cah.value;
        ...
        try {
            if (RealCalcClusterService.realCalcClService == null)
                RealCalcClusterService.createRealCalcClusterService();
            ServiceData input = new ServiceData();
            input.setBytes(personData);
            ServiceData response =
                RealCalcClusterService.realCalcClService.execute(RealCalcClusterService.reply.getSessionId(),
                    RealCalcClusterService.serviceId, "getResult",
                        input);
            byte[] respFromvast = response.getBytes();
            cah.value = respFromvast;
            input.setBytes(null);
            response.setBytes(null);
        }
        catch (Exception e) {
            e.printStackTrace();
        }
    }
}
A later version of ClusterService uses a similar approach. **PersistentClusterServiceFacade**, an API, provides access to static ClusterService interfaces in order to maintain these service instances across a JVM reset. It is more compatible with the PRJVM, provides better performance and enables us to keep the LiveCluster API in the trusted middleware class. Another difference with this version is that it is built on a Java Web service's client interface and takes a broker Web services uniform resource locator (URL) as input. Also, the development procedure is simplified without the need to use wsdl2java. Driver code using this new API is shown in Example 4.

**Example 4 CalClnt.java (with persistentclusterservicefacade)**

```java
import com.ibm.cics.server.*;
import com.livecluster.webservice.clusterservice.*;
import com.datasynapse.livecluster.driver.*;
public class CalClnt {
    public static void main(CommAreaHolder cah) throws Exception {
        byte[] personData = cah.value;
        ...
        try {
            // set service url, name, userID and password
            String uid = "prod";
            String password = "password";
            String primUrl = "http://etpserv26:8000/livecluster/webservice/ClusterService";
            String svcName = "RealCalcService.java";
            ServiceData input = new ServiceData();
            // create clusterservice
            ClusterService rcService = PersistentClusterServiceFacade.instance(uid, password, primUrl, secondUrl).getClusterService(svcName);
            // set input
            input.setBytes(personData);
            // execute calc, execute getResult() method of RealCalcService class
            ServiceData response = rcService.execute("getResult", input);
            byte[] respFromvast = response.getBytes();
            cah.value = respFromvast;
            ...
        } catch (Exception e) {
            e.printStackTrace();
        }
    }
    ...
}
```

**CalcEngine service implementation**

Each of the DataSynapse engines has three distinct functions that execute:

**Service Implementation** This receives calculation requests and interfaces with the Smalltalk application.

**Update Environment** This receives notification of state changes in the grid when they occur and before any additional units of (queued) work are processed.

**Engine Hook** This provides Service-specific processing at initiation and termination of the Engine on this machine. Its main purpose is to synchronize with the Smalltalk application.
Service implementation

The main service function executed on the LiveCluster Engines is CalcEngine pension valuation. The Smalltalk CalcEngine is ported from CICS to Linux, but it cannot be driven or managed by LiveCluster directly. Instead, we need a hub on the Engine that is managed by the LiveCluster broker. This hub gets the request from the LiveCluster Server, forwards it to CalcEngine, and passes back the response. The hub code is implemented in Java and uses a local socket to communicate with the Smalltalk CalcEngine. CalcEngine itself is modified to listen on an open server socket and wait for requests.

As shown in Example 1 on page 5, the client makes a request to the getResult method of service RealCalcService.java. RealCalcService.java is the name of the service registered with the LiveCluster Director. The logic for this service is the hub code. getResult is the method that opens a socket to CalcEngine, sends a request with PersonData, and waits for a response. In our case, if CalcEngine fails to respond to the socket for a period of time, an error message is sent back for the client to handle. Other options that LiveCluster provides are throwing an exception and letting the broker reschedule the request, or throwing an exception and letting the engine restart.

The InitCalcService method loads systems properties and application-specific properties (for example, the socket port number). It is registered as an initialization method that is executed when a physical instance of a service is started on a grid engine. DestroyCalcService is registered as a destroy method that does necessary cleanup and is called just before the service is destroyed. The code fragment is shown in Example 5. The service is deployed to LiveCluster as shown in Figure 4 on page 10.

Example 5  RealCalcService.java (fragments)

```java
...  // RealCalc Service
public class RealCalcService {
    int execCount = 0;
    byte[] updateCmd = new byte[15];
    int base_port = 1230;
    // init() method
    public void initCalcService() {
        Properties myProps = new Properties();
        try {
            myProps.load(new FileInputStream("./resources/shared/properties/" + EngineSession.getProperties().getProperty("BROKER") + ".properties"));
            base_port = Integer.parseInt(myProps.getProperty("CalEngine_PORT"));
        } catch (Exception e1) {
            System.out.println("error getting properties");
            ...
        }
    }
    // destroy service
    public void destroyCalcService() {}
    public void updateEnv (byte[] updateCmd1) {}
    ...  // getResult
    // input : person data from mainframe
    // open socket to Smalltalk and send bytes, wait response from Smalltalk
    // get response from Smalltalk, close socket and return result to mainframe
    // output: result from CalcEngine or error message
    public byte[] getResult(byte[] input) {
        String respmsg = "error: from socket to Smalltalk app"; // response holder
        String reqmsg = " ";
        Socket clientSocket = null;
```
String host = "127.0.0.1";
int port = base_port +
Integer.parseInt(EngineSession.getProperties().getProperty(EngineProperties.INSTANCE));
try {
    clientSocket = new Socket(host, port); }
catch (IOException ioe) {
    respmsg = "error: failed to bind port " + Integer.toString(port); }
if (clientSocket == null) return respmsg.getBytes();
else
    try
    {
        byte[] result ;
        // open socket and send request data
        OutputStream os = (clientSocket.getOutputStream());
        BufferedOutputStream buffOut = new BufferedOutputStream (os);
        DataOutputStream dos = new DataOutputStream (buffOut);
        dos.write(input,0, Array.getLength(input));
        dos.flush();
        System.out.println("Array=" + Array.getLength(input) + ", sent= " + dos.size());
        InputStream is = (clientSocket.getInputStream());
        DataInputStream dis = new DataInputStream(is);
        // wait for response
        int av = 0 ;
        System.out.println("available=", av);
        int times = 0 ;
        while ((av == 0) || (times < 36000))
        {
            java.lang.Thread.sleep(5);
            times ++;
            av = dis.available();
            System.out.println("available=", av);
        }
        if (av == 0)
            result = (new String("error: time out waiting for Smalltalk")).getBytes();
        else
            result = new byte[av];

---

<table>
<thead>
<tr>
<th>Property</th>
<th>Value</th>
<th>Desc</th>
</tr>
</thead>
<tbody>
<tr>
<td>NAME</td>
<td>RealCalcService.java</td>
<td></td>
</tr>
<tr>
<td>TYPE</td>
<td>java</td>
<td></td>
</tr>
<tr>
<td>Provider</td>
<td>Services implemented in Java.</td>
<td></td>
</tr>
<tr>
<td>className</td>
<td>RealCalcService</td>
<td>The Java fully-qualified class name</td>
</tr>
<tr>
<td>Options</td>
<td>ClusterService Options.</td>
<td></td>
</tr>
<tr>
<td>initMethod</td>
<td>initCalcService</td>
<td>Initialization method.</td>
</tr>
<tr>
<td>cancelMethod</td>
<td></td>
<td>Cancel method, called when a service</td>
</tr>
<tr>
<td>destroyMethod</td>
<td>destroyCalcService</td>
<td>Destroy method, called when a service</td>
</tr>
<tr>
<td>email</td>
<td></td>
<td>The optional email address, for notify created and destroyed.</td>
</tr>
<tr>
<td>priority</td>
<td></td>
<td>The priority of this service instance.</td>
</tr>
</tbody>
</table>

Figure 4   Service code deployment to LiveCluster
**Updating environmental and state data**

The `updateEnv` method in `RealCalcService.java` is used for updating state data. As we mentioned earlier, `EnvironmentData` is loaded from DB2 and cached in `CalcEngine`. We treat `EnvironmentData` as part of the service state and `CalcEngine` state. When environmental data is updated, `CalcEngine` needs to cast out the cached data and reload the new version if another request comes in.

In LiveCluster, the `updateState` method of `ClusterService` class behaves quite differently from execute and submit methods. All three transmit a request to the Server, but the latter two result in the request being sent to an Engine as soon as possible (subject to Engine availability and scheduler configuration); state updates remain on the Server for the life of the service instance. When an Engine is given a non-update request to process, it downloads all state updates that it has not already processed, executes them in order, and then finally processes the non-update request. To send an update request, the Driver calls the `updateState` method of the `ClusterService`, passes in the service method name and the new service data. The new service data can be new state information or a command to tell service what to do. In our case, the method for update state is `updateEnv`; the new service data is a command to tell `CalcEngine` to cast out old `EnvironmentData`. `CalcEngine` reloads specific `EnvironmentData` later only when a new request comes in for it. The implementation for Method `updateEnv` is similar to the `getResult` method, so it is not shown in Example 5 on page 9. It connects to `CalcEngine` through the same port and sends a byte array `updateCmd1` as a command to tell `CalcEngine` to cast out certain pieces of cached environmental data.

**Engine Hook**

When `CalcEngine` starts, it opens a port and listens to requests coming in. Because it is a single-threaded application, the best way to utilize hardware resources is to start the same number of `CalcEngine` instances as the number of CPUs on a multiprocessor machine. Also, the number of CPUs determines the default number of LiveCluster engine instances started by the LiveCluster Server. One `CalcEngine` instance should run in one LiveCluster engine instance, and it should be started as soon as the LiveCluster engine is started.

LiveCluster has a feature called engine hook that enables users to perform operations upon login of the Engine (that is, to “hook” in customer-defined initialization code) and upon termination of the Engine. We adapted an EngineHook sample from DataSynapse (see code fragment in Example 7 on page 12 and full source in “Appendix” on page 15). It starts the `CalcEngine` by executing a command from a customized engine properties file and adds extra properties to the existing engine properties. Which port the `CalcEngine` listens to is based on engine instance number, so that each `CalcEngine` within the same Linux image has a different port.

Another engine hook function tells `CalcEngine` to close the port before engine shutdown. The code sample is shown in Example 8 on page 13. There can be as many as hooks as needed. The `initialized()` method of all engine hooks is executed upon engine startup, and the `terminated()` method of all hooks is executed before engine shutdown. All hook Jar files need to be deployed to the server and populated to engines. All hooks are configured in an Extensible Markup Language (XML) file that is deployed to the server and distributed to the engines. The sample .xml file is shown in Example 6.

---

**Example 6 processhook.xml**

```xml
<hookcontainer class="HookContainer">
  <hook class="ProcessHook">
  </hook>
  <hook class="VastShutDownHook">
  </hook>
</hookcontainer>
```
public class ProcessHook extends EngineHook {
  private static final String BROKER = "BROKER";
  private static final String PROCESS_NAME = "PROCESS_NAME";
  private static final String PROCESS_ARGS = "PROCESS_ARGS";
  private static final String PROCESS_STDOUT = "PROCESS_STDOUT";
  private static final String PROCESS_STDERR = "PROCESS_STDERR";
  private static final String PROCESS_STDIN = "PROCESS_STDIN";
  private static final String HEWITT_PORT = "HEWITT_PORT";
  private static Object _olock = new Object();
  private static ProcessHook _instance = null;
  private ICProcessRunner _processRunner;
  private Exception _exc;
  private Process _process;
  private String _cmd;
  private String[] _cmd_args;
  private String _stdin;
  private String _stdout;
  private String _stderr;
  ...

  public void initialized() {
    System.out.println("ProcessHook: initialized() entry");
    // get our Broker EngineSession URL
    com.livecluster.server.plugin.LoginPlugin lp =
      (com.livecluster.server.plugin.LoginPlugin)
        (com.livecluster.server.MessageServer.instance(0).findPluginByClass(com.livecluster.server.plugin.LoginPlugin.class));
    String url = com.livecluster.server.MessageServer.instance(0).getConnection(lp.getPrivateConnectionName()).getSender().toString();
    // parse URL for name of broker and set as EngineSession property
    String broker = url;
    try {
      java.net.URL brokerURL = new java.net.URL(url);
      broker = brokerURL.getHost() + "-" + brokerURL.getPort();
    } catch (java.net.MalformedURLException ex) {
    }
    EngineSession.setProperty(BROKER, broker);
    // read in properties file keyed off broker name for process name, stdin, stdout, and stderr and start process
    _exc = null;
    if (!broker.equals("")) {
      try {
        Properties myProps = new Properties();
        myProps.load(new FileInputStream("./resources/shared/properties/" +
          EngineSession.getProperties().getProperty(BROKER) + ".properties"));
        Enumeration enum = myProps.keys();
        while (enum.hasMoreElements()) {
          String key = (String) enum.nextElement();
          EngineSession.setProperty(key, myProps.getProperty(key));
        }
        _cmd = myProps.getProperty(PROCESS_NAME);
        String _args = myProps.getProperty(PROCESS_ARGS, "");
        StringTokenizer st = new StringTokenizer(_args, ",");
        _cmd_args = new String[st.countTokens()+1];
        int i = 0;
        while (st.hasMoreTokens()) {
          _cmd_args[i++] = new String(st.nextToken());
        }
        // port number
A CICS to Linux Grid Implementation

Example 8   VastShutDownHook.java

```java
public class VastShutDownHook extends EngineHook {
    public void initialized() {
        System.out.println("VastShutDownHook: initialized ");
    }
    public void terminated() {
        System.out.println("VastShutDownHook: terminated is called ");
        String command = "@ÈäãÄÖæÕ"; // "SHUTDOWN" in EBCDIC
        Socket clientSocket = null;
        String host = "127.0.0.1"; // localhost
        int base_port = 1230;
        int port = base_port +
        Integer.parseInt(EngineSession.getProperties().getProperty(EngineProperties.INSTANCE));
        try {
            clientSocket = new Socket(host, port);
        } catch (IOException ioe) {
            System.out.println("error:failed to bind CalcEngine port ");
        }
        if (clientSocket != null) {
            try {
                byte[] input = command.getBytes();
                OutputStream os = (clientSocket.getOutputStream());
                BufferedOutputStream buffOut = new BufferedOutputStream (os);
                DataOutputStream dos = new DataOutputStream (buffOut);
                dos.write(input,0, Array.getLength(input));
                dos.flush();
                dos.close();
                clientSocket.close();
            }
        }
    }
}
```
Summary

By moving the calculation-intensive part of the application from a resource-constrained environment to an open Linux grid environment, we were able to achieve more flexible deployment and workload growth without a major application rewrite. DataSynapse provides the deployment infrastructure and management features. Taking advantage of the PRJVM features in CICS significantly improved the performance of the implementation.
Appendix

Sample Java source code: ProcessHook.java

```java
package examples.hook;
import java.io.*;
import java.util.*;
import com.datasynapse.livecluster.engine.*;
/*
  ProcessHook is an example demonstrating how to run an external process from the EngineHook interface.
  This example defaults to a Unix-style environment (with "/dev/null") but presumably could be adapted
  for other uses. You set the command to be run and the in/out/err files via Engine properties.
  Don't forget to create an XML per the EngineHook docs, e.g.:
  <hookcontainer class="HookContainer">
    <hook class="examples.hook.ProcessHook">
  </hook>
</hookcontainer>
  You'll also want to adapt the error handling to your preferences. In the example, tasklets or
  service methods can call ProcessHook.instance().except() to get any Exception that was
  thrown.
  Null indicates a good status.
  Thread safety is always a problem with Java's process management methods, so calling
  methods should be prepared for NullPointerExceptions and other race condition byproducts.
*/
public class ProcessHook
  extends EngineHook {
  public ProcessHook() throws Exception {
    synchronized (_olock) {
      if (_instance != null) {
        throw new Exception("Only one ProcessHook may be created");
      }
      _instance = this;
    }
  }
  public static ProcessHook instance() {
    synchronized (_olock) {
      return _instance;
    }
  }
  public void initialized() {
    System.out.println("ProcessHook: initialized() entry");
    // get our Broker EngineSession URL
    com.livecluster.server.plugin.LoginPlugin lp =
    (com.livecluster.server.plugin.LoginPlugin)
    plugin.LoginPlugin.class));
    String url =
    com.livecluster.server.MessageServer.instance(0).getConnection(lp.getPrivateConnectionName( )
    ).getSender().toString();
    // parse URL for name of broker and set as EngineSession property
    String broker = url;
    try {
      java.net.URL brokerURL = new java.net.URL(url);
    }
  }
```

Example 9  ProcessHook.java
broker = brokerURL.getHost() + "-" + brokerURL.getPort();
} catch (java.net.MalformedURLException ex) {}

EngineSession.setProperty(BROKER, broker);
// read in properties file keyed off broker name for process name, stdin, stdout, and stderr and start process
_exc = null;
if (!broker.equals("")) {
    try {
        Properties myProps = new Properties();
        myProps.load(new FileInputStream("./resources/shared/properties/" +
        EngineSession.getProperties().getProperty(BROKER) + ".properties"));
        Enumeration enum = myProps.keys();
        cycle_count = 0; // July15POK
        v_cycle = new Vector(); // July15POK
        while (enum.hasMoreElements()) {
            String key = (String) enum.nextElement();
            EngineSession.setProperty(key, myProps.getProperty(key));
            if (key.substring(0, 5).equalsIgnoreCase("CYCLE")) { // July15POK
                cycle_count ++; // July15POK
                v_cycle.addElement(myProps.getProperty(key)); // July15POK
            }
        }
        if (cycle_count > 0) { // July15POK
            EngineSession.setProperty("CYCLE_COUNT", (new Integer(cycle_count)).toString()); // July15POK
            _cmd_p = myProps.getProperty(PROCESS_NAME);
            String _args = myProps.getProperty(PROCESS_ARGS, "");
            StringTokenizer st = new StringTokenizer(_args, ",");
            _cmd_args = new String[st.countTokens()];
            _cmd_args_p = new String[st.countTokens()+1];
            java.lang.System.out.println("Engine Instance=+
            EngineSession.getProperties().getProperty(EngineProperties.INSTANCE));
            int i = 0;
            while (st.hasMoreTokens()) {
                _cmd_args_p[i++] = new String(st.nextToken());
            }
            // port number
            _cmd_args_p[0] =
            Integer.toString(Integer.parseInt(myProps.getProperty(HEWITT_PORT)) +
            // Integer.parseInt(
            // EngineSession.getProperties().getProperty(EngineProperties.INSTANCE));
            _stdin_p = myProps.getProperty(PROCESS_STDIN, "/dev/null");
            _stdout_p = myProps.getProperty(PROCESS_STDOUT, "/dev/null");
            _stderr_p = myProps.getProperty(PROCESS_STDERR, "/dev/null");
            if (_cmd_p == null) {
                System.out.println("ProcessHook: no runtime command specified");
                _exc = new Exception("No runtime command specified");
                return;
            }
            String cmd_str = null;
            String stdout_str = null;
            String stdin_str = null;
            String stderr_str = null;
            for (int j =0; j< cycle_count; j++) {
                cmd_str = _cmd_p + " " +
(Integer.toString(Integer.parseInt(myProps.getProperty(HEWITT_PORT)) +
  Integer.parseInt(
    EngineSession.getProperties().getProperty(EngineProperties.INSTANCE)))) +
  (String) v_cycle.elementAt(j) ;
stdout_str = _stdout_p + "\n" +
EngineSession.getProperties().getProperty(EngineProperties.INSTANCE) +
  "\n" + (String) v_cycle.elementAt(j); //July15POK
stdin_str = _stdin_p ; //July15POK
stderr_str = _stderr_p + "\n" +
EngineSession.getProperties().getProperty(EngineProperties.INSTANCE) +
  "\n" + (String) v_cycle.elementAt(j); //July15POK
  _processRunner = new ICProcessRunner( );
  _processRunner = new ICProcessRunner(cmd_str,
stdout_str,stdin_str,stderr_str  );//July15POK
  _processRunner.start();
}   // end for loop
} catch (Exception e) {
  _exc = e;
}
}
public void terminated() {
  _processRunner.halt();
}
public Exception status() {
  return _exc;
}
private ICProcessRunner _processRunner;
private Exception _exc;
//    private Process _process;
//    private String _cmd;
//    private String[] _cmd_args;
//    private String _stdin;
//    private String _stdout;
//    private String _stderr;
private Vector v_cycle; //July15POK
private int cycle_count;  //July15POK
private String _cmd_p;   //July15POK
private String[] _cmd_args_p;  //July15POK
private String _stdin_p;  //July15POK
private String _stdout_p;  //July15POK
private String _stderr_p;   //July15POK
private class ICProcessRunner
  extends Thread {
    public ICProcessRunner() {//July15POK
    }//July15POK
    public ICProcessRunner(String cmd,String stdout,String stdin,String stderr)
      {//July15POK
      _cmd=cmd;  //July15POK
      _stdout=stdout; / /July15POK
      _stdin=stdin; //July15POK
      _stderr=stderr; //July15POK
      }//July15POK
    public void run() {
      try {
        System.out.println("ProcessHook$ICProcessRunner: running");
        _process = Runtime.getRuntime().exec(_cmd, _cmd_args);//July15POK
System.out.println("cmd="+_cmd);
_process = Runtime.getRuntime().exec(_cmd);//July15POK
_outputCopier = new ICStreamCopyRunner(new FileInputStream(_stdin),
_process.getOutputStream(), true);
_inputCopier = new ICStreamCopyRunner(_process.getInputStream(), new
FileOutputStream(_stdout), true);
_errorCopier = new ICStreamCopyRunner(_process.getErrorStream(), new
FileOutputStream(_stderr), true);
_outputCopier.start();
_inputCopier.start();
_errorCopier.start();
_outputCopier.join();
_inputCopier.join();
_errorCopier.join();
_exitcode = _process.waitFor();
if (_exitcode != 0) {
    ProcessExitedException pxe = new ProcessExitedException("Process exited
with non-zero exit code " + _exitcode);
    pxe.setExitcode(_exitcode);
    throw pxe;
}
if (_inputCopier.getException() != null) {
    throw _inputCopier.getException();
}
if (_outputCopier.getException() != null) {
    throw _outputCopier.getException();
}
if (_errorCopier.getException() != null) {
    throw _errorCopier.getException();
}
} catch (Exception e) {
    System.out.println("ProcessHook$ICProcessRunner: exception thrown");
    e.printStackTrace();
    _exc = e;
} finally {
    _process = null;
}
}

public void halt() {
    if (_process != null) {
        _process.destroy();
    }
}

private ICStreamCopyRunner _inputCopier;
private ICStreamCopyRunner _outputCopier;
private ICStreamCopyRunner _errorCopier;
private int _exitcode = 0;
private Process _process;  //July15POK
private String _cmd;  //July15POK
private String[] _cmd_args;  //July15POK
private String _stdin;  //July15POK
private String _stdout;  //July15POK
private String _stderr;  //July15POK
}

private class ICStreamCopyRunner
    extends Thread {
    public ICStreamCopyRunner(InputStream is, OutputStream os) {
        _is = is;
    }
public class ProcessExitedException extends Exception {
    public ProcessExitedException(String msg) {
        super(msg);
    }
}

private long copy(InputStream in, OutputStream out) throws IOException {
    byte[] buf = new byte[4096];
    int totalBytes = 0;
    while (true) {
        int bytesRead = in.read(buf);
        if (bytesRead == -1) {
            break;
        }
        totalBytes += bytesRead;
        out.write(buf, 0, bytesRead);
    }
    return totalBytes;
}

private InputStream _is;
private OutputStream _os;
private Exception _runException;
private boolean _closeOutput;
private long _nbytes;
};

public class ProcessExitedException extends Exception {
    public ProcessExitedException(String msg) {
        super(msg);
    }
}
private int _exitcode = 0;
public void setExitcode(int exitcode) {
        _exitcode = exitcode;
    }

    public int getExitcode() {
        return _exitcode;
    }

private static final String BROKER = "BROKER";
private static final String PROCESS_NAME = "PROCESS_NAME";
private static final String PROCESS_ARGS = "PROCESS_ARGS";
private static final String PROCESS_STDOUT = "PROCESS_STDOUT";
private static final String PROCESS_STDERR = "PROCESS_STDERR";
private static final String PROCESS_STDIN = "PROCESS_STDIN";
private static final String HEWITT_PORT = "HEWITT_PORT";
private static Object _olock = new Object();
private static ProcessHook _instance = null;
References

- IBM Design Center
  
  http://www-1.ibm.com/servers/eserver/design_center/

- IBM Developer Kit for OS/390, Java 2 Technology Edition
  
  New IBM technology featuring Persistent Reusable Java Virtual Machines

- IBM Redpaper: CICS TS 2.2 with Persistent Reusable JVM In a Grid Environment (to be published on http://www.ibm.com/redbooks)

- IBM Smalltalk User's Guide, version 6.0

- VisualAge Smalltalk Database Guide, version 6.0

- Apache Ant Project
  
  http://ant.apache.org/

- Apache Web Services Project
  
  http://ws.apache.org/axis/

- DataSynapse home page
  
  http://www.datasynapse.com/

- DataSynapse LiveCluster 3G Administration Guide

- DataSynapse LiveCluster 3G Developer's Guide

The team who wrote this Redpaper

This Redpaper was produced by a team of specialists from around the world working at the IBM Design Center, Poughkeepsie.

Hong Min is a software engineer at the IBM Design Center for e-business on demand™, Poughkeepsie. She has more than six years of experience in e-business technical support, customer application design, and prototyping. Her current job focuses are WebSphere®, J2EE applications, and Grid computing.

Ken Muckenaupt is a Senior Software Engineer at the IBM Design Center for e-business on demand in Poughkeepsie and has more than 25 years of experience in IBM hardware and software development. He currently works as a Grid technology and WebSphere consultant.

Paul Wanish is a Senior Technical Staff Member with the IBM Design Center for e-business on demand in Poughkeepsie. He provides leadership to customer IT organizations in solving key business problems through the use of IBM products, such as WebSphere, Linux, and IBM @server technologies. Paul joined IBM in 1970 with a BS in Computer Science, and he holds several patents and many invention disclosures.

The ITSO project leader for this Redpaper:
Mike Ebbers, International Technical Support Organization (ITSO), Poughkeepsie
Acknowledgements

Thanks to the following people for their contributions to this project:

Ira Chavis
xSeries Benchmark Center and OnDemand Infrastructure, IBM Poughkeepsie

Steve Carbaugh
Lynn Winkelbaurer
Design Center for e-business on demand, IBM Poughkeepsie

Mark Cocker
CICS TS, IBM Hursley

Ginny Ghezzo
Gryan Hogan
Rick Trotter
Rob Pungello
Larry Hamann
Greg Curfman
IBM VisualAge Smalltalk Development, IBM

John Skovron
Anthony Fox
DataSynapse, Inc.
Notices

This information was developed for products and services offered in the U.S.A.

IBM may not offer the products, services, or features discussed in this document in other countries. Consult your local IBM representative for information on the products and services currently available in your area. Any reference to an IBM product, program, or service is not intended to state or imply that only that IBM product, program, or service may be used. Any functionally equivalent product, program, or service that does not infringe any IBM intellectual property right may be used instead. However, it is the user's responsibility to evaluate and verify the operation of any non-IBM product, program, or service.

IBM may have patents or pending patent applications covering subject matter described in this document. The furnishing of this document does not give you any license to these patents. You can send license inquiries, in writing, to:
IBM Director of Licensing, IBM Corporation, North Castle Drive Armonk, NY 10504-1785 U.S.A.

The following paragraph does not apply to the United Kingdom or any other country where such provisions are inconsistent with local law: INTERNATIONAL BUSINESS MACHINES CORPORATION PROVIDES THIS PUBLICATION "AS IS" WITHOUT WARRANTY OF ANY KIND, EITHER EXPRESS OR IMPLIED, INCLUDING, BUT NOT LIMITED TO, THE IMPLIED WARRANTIES OF NON-INFRINGEMENT, MERCHANTABILITY OR FITNESS FOR A PARTICULAR PURPOSE. Some states do not allow disclaimer of express or implied warranties in certain transactions, therefore, this statement may not apply to you.

This information could include technical inaccuracies or typographical errors. Changes are periodically made to the information herein; these changes will be incorporated in new editions of the publication. IBM may make improvements and/or changes in the product(s) and/or the program(s) described in this publication at any time without notice.

Any references in this information to non-IBM Web sites are provided for convenience only and do not in any manner serve as an endorsement of those Web sites. The materials at those Web sites are not part of the materials for this IBM product and use of those Web sites is at your own risk.

IBM may use or distribute any of the information you supply in any way it believes appropriate without incurring any obligation to you.

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 and cannot confirm the accuracy of performance, compatibility or any other claims related to non-IBM products. Questions on the capabilities of non-IBM products should be addressed to the suppliers of those products.

This information contains examples of data and reports used in daily business operations. To illustrate them as completely as possible, the examples include the names of individuals, companies, brands, and products. All of these names are fictitious and any similarity to the names and addresses used by an actual business enterprise is entirely coincidental.

COPYRIGHT LICENSE:
This information contains sample application programs in source language, which illustrates programming techniques on various operating platforms. You may copy, modify, and distribute these sample programs in any form without payment to IBM, for the purposes of developing, using, marketing or distributing application programs conforming to the application programming interface for the operating platform for which the sample programs are written. These examples have not been thoroughly tested under all conditions. IBM, therefore, cannot guarantee or imply reliability, serviceability, or function of these programs. You may copy, modify, and distribute these sample programs in any form without payment to IBM for the purposes of developing, using, marketing, or distributing application programs conforming to IBM's application programming interfaces.
This document created or updated on October 10, 2003.

Send us your comments in one of the following ways:

- Use the online Contact us review redbook form found at:
  ibm.com/redbooks
- Send your comments in an Internet note to:
  redbook@us.ibm.com
- Mail your comments to:
  IBM Corporation, International Technical Support Organization
  Dept. HYJ Mail Station P099
  2455 South Road
  Poughkeepsie, NY 12601-5400 U.S.A.

Trademarks

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

- CICS®
- DB2®
- e-business on demand™
- IBM®
- ibm.com®
- IMS™
- Redbooks™
- Redbooks(logo)™
- VisualAge®
- WebSphere®
- xSeries®
- z/OS®
- zSeries®

The following terms are trademarks of other companies:

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

UNIX is a registered trademark of The Open Group in the United States and other countries.

Other company, product, and service names may be trademarks or service marks of others.