# Step By Step

Windows HPC Server 2008 R2

Microsoft

This document is provided “as-is”. Information and views expressed in this document, including URL and other Internet Web site references, may change without notice. You bear the risk of using it.

Some examples depicted herein are provided for illustration only and are fictitious. No real association or connection is intended or should be inferred.
This document does not provide you with any legal rights to any intellectual property in any Microsoft product. You may copy and use this document for your internal, reference purposes.

© 2010 Microsoft Corporation. All rights reserved.

Microsoft, Active Directory, Excel, Outlook, SQL Server, Visual Basic, Visual C#, Visual Studio, Windows, Windows PowerShell, and Windows Server are trademarks of the Microsoft group of companies. All other trademarks are property of their respective owners.

# Table of Contents


Chapter 1 The Beginning: Building a Windows HPC cluster ..... 1

1.1 Windows HPC Server Cluster Architecture ..... 1

1.2 Create High Performance Computing Cluster ..... 2

1.3 Assign Node Template ..... 22

1.4 Try the HPC Application ..... 25

Chapter 2 Introduction of Windows HPC Server 2008 R2 ..... 29

2.1 Windows HPC Server 2008 R2 ..... 29

2.2 MS-MPI ..... 33

2.3 Job Scheduler System ..... 47

2.4 Cluster Manager ..... 58

2.5 Security ..... 74

2.6 Summary ..... 74

Chapter 3 Job Scheduler ..... 75

3.1 Concepts ..... 75

3.2 Application install ..... 75

3.3 Submit Job ..... 76

3.4 Tune Resource Allocation ..... 97

3.5 Appendix ..... 108

3.6 Working with Job Files and Submitting Jobs ..... 113

3.7 Controlling Jobs Using Filters ..... 113

3.8 Security Considerations for Jobs and Tasks ..... 113

Chapter 4 Developing HPC SOA Applications ..... 115

4.1 Sequential and Parallel Monte Carlo Simulation for Asian Option Pricing ..... 115

4.2 Create the HPC SOA Service ..... 120

4.3 Create a Fire and Recollect Client Application ..... 135

Chapter 5 Programming MPI-based HPC Applications ..... 149

5.1 Problem Domain ..... 149

5.2 Digital Representation of an Image ..... 150

5.3 Data Parallelism and Contrast Stretching ..... 152

5.4 A Sequential Version of the Contrast Stretching Application ..... 155

5.5 Running MPI Applications on Windows HPC Server 2008 R2 ..... 174
Chapter 6 Manage the Windows HPC Server 2008 R2 Cluster ..... 181
6.1 Introduction ..... 181
6.2 Cluster Deployment ..... 182
6.3 Configuration ..... 192
6.4 Monitor the Cluster ..... 213
6.5 Diagnostics ..... 221
6.6 Reporting-Based Decisions ..... 224
Chapter 7 Building VBA Applications and Workbooks for a Windows HPC Server 2008 R2 Cluster ..... 239
7.1 Structure and Formatting ..... 239
7.2 Overview ..... 240
7.3 Prerequisites and Requirements ..... 243
7.4 Part I: Building a Workbook from Scratch ..... 245
7.5 Part II: Converting a Desktop Workbook to Run on the Windows HPC Server Cluster 263
7.6 Best Practices for Cluster-Enabled Excel Workbooks ..... 283
7.7 Final Notes ..... 286
Chapter 8 User Experience: SAIC (Shanghai Automotive Industry Corporation) ..... 287
8.1 Introduction to SAIC and its business requirements ..... 287
8.2 The Solution of Windows HPC Server and its Advantages ..... 288
8.3 Computer Aided Engineering (CAE) Application and Challenges in High-Performance Computing ..... 288
8.4 SimCloud: The portal solution, which is based on Windows HPC Server ..... 290
8.5 Application Integration Case ..... 293
8.6 License Management ..... 302

# Chapter 1 The Beginning: Building a Windows HPC cluster

Building a cluster is a complex and error-prone process. The administrator needs to install software onto each machine and configure the cluster network topology so that machines that consist of multiple networks can be managed as a resource pool. The administrator also needs to manage images and setup software upgrade strategy for daily maintenance. To verify that the cluster has been successfully built, the administrator needs to run built-in diagnostic tests to confirm that the hardware, network, and software can be properly used in the production environment. This chapter describes how to build a Windows® HPC Server 2008 R2 cluster and shows how to simplify installation procedures and mitigate common errors.

Chapter Highlights:

- HPC architecture and how it integrates into the enterprise environment
- Head node installation, and cluster configuration based on a to-do list
- Cluster deployment based on node template
- Microsoft design of simplified cluster management
- Run an Asian option pricing application to test the cluster

## 1.1 Windows HPC Server Cluster Architecture

Windows HPC Server 2008 R2 consists of a head node and multiple computing nodes and/or workstations (see Figure 1-1). The head node can provide Windows Server 2008 R2 Enterprise high-availability failover services and the SQL Server® cluster service. It decides and grants access to the cluster resource; thus, this is the only node to manage, deploy, and schedule cluster jobs. Windows HPC Server 2008 R2 can provide security and account management by using the existing Active Directory® infrastructure. The head node can also use build-in system management tools to execute cluster-wide operations.

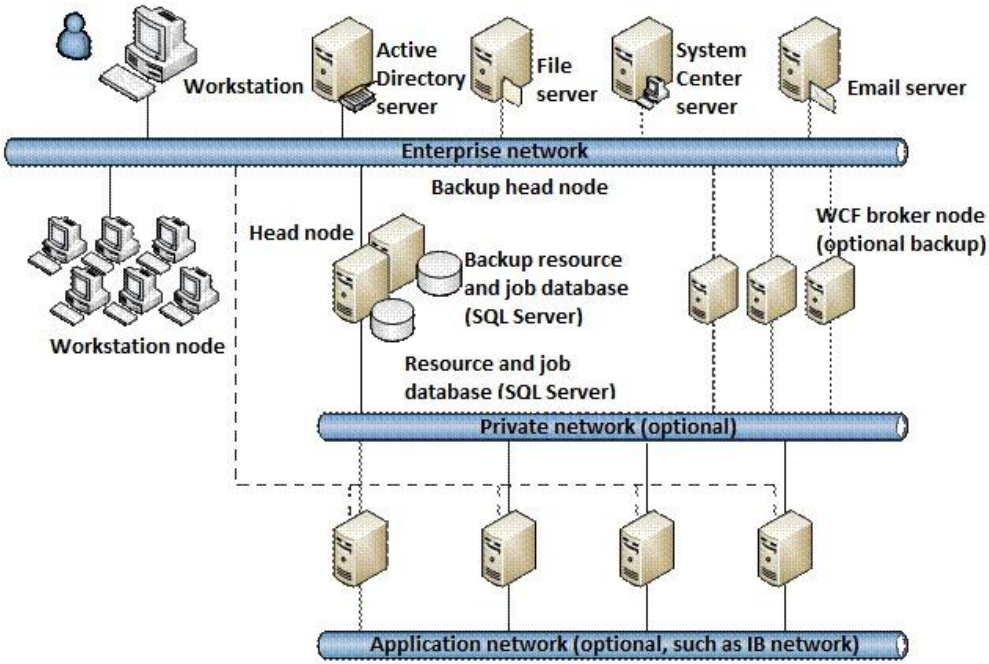



*Figure 1-1 Typical Windows HPC Server 2008 R2 network*

Linux system administrator shortcut: Active Directory

If you use NIS or /etc/passwd to manage user accounts or machines, you can replace it with Active Directory in a Windows system. The Active Directory service is a core component in Windows Server. It is an efficient way for users to manage the relationship of each key component in the same network. Administrators can easily search and use network object information stored in Active Directory.

In 2004, to provide enterprise users with a transparent and secure user experience, the Microsoft HPC team decided to add all cluster nodes to Active Directory. The goal is to provide a seamless experience for Active Directory users to access network resources such as data files, databases, and Web services. They can use the same account to access the compute nodes, while a Linux cluster with a basic installation cannot use enterprise users' directory accounts. It needs complicated configuration to sync an Active Directory account and NIS account to achieve this. Most users will simply create another NIS account. Therefore, the administrator will need to manage and maintain two sets of accounts.

## 1.2 Create High Performance Computing Cluster

Deploying Windows HPC Server 2008 R2 involves three steps, as shown in Figure 1-2.

*Figure 1-2 Steps to deploy an HPC Cluster*

1. Create a cluster: Install the operating system on the head node, join the head node into the Active Directory domain, and then install HPC Pack 2008 R2.
2. Configure the cluster: navigation panel contains a to-do list for setting cluster configuration. It includes the network configuration wizard, the node template creation wizard, and other tools. The network configuration wizard helps you to configure a cluster network easily. The node template creation wizard instructs the cluster administrator to create a node template to deploy all the compute nodes in a unified way, including installing the operating system, running the operating system update, and installing applications.
3. Deploy the compute nodes: Install the operating system, install HPC Pack 2008 R2 on the compute nodes, and add the compute nodes to the cluster.

### 1.2.1 Create a Cluster

Creating a cluster involves three steps: installing the Windows HPC Server 2008 R2 operating system on the head node machine, joining the Active Directory domain from the head node, and installing HPC Pack 2008 R2.

#### 1.2.1.1 Install the Windows HPC Server 2008 R2 Operating System

Insert the Windows HPC Server 2008 R2 DVD into the DVD-ROM drive, and follow the Windows Server installation instructions to install the operating system.

### 1.2.1.2 Join the Active Directory Domain

**Tip**

You need to have "Creation of Computer objects" permission to proceed. For more information, contact your Active Directory domain administrator. If you do not have such permission, you will get an "Access Denied" error.

To join the head node machine to the Active Directory domain, do the following:

1. On the **Start** menu, right-click **Computer** and select **Properties**.
2. Click **Change settings**.
3. Click the **Computer Name** tab and click **Change ...**.
4. Enter the domain name in the Domain text box.
5. Click **OK**, and then reboot the system.

### 1.2.1.3 Install HPC Pack 2008 R2

1. Insert the HPC Pack 2008 R2 DVD into the DVD-ROM drive and double-click "setup.exe". The installation window shown in Figure 1-3 is displayed.

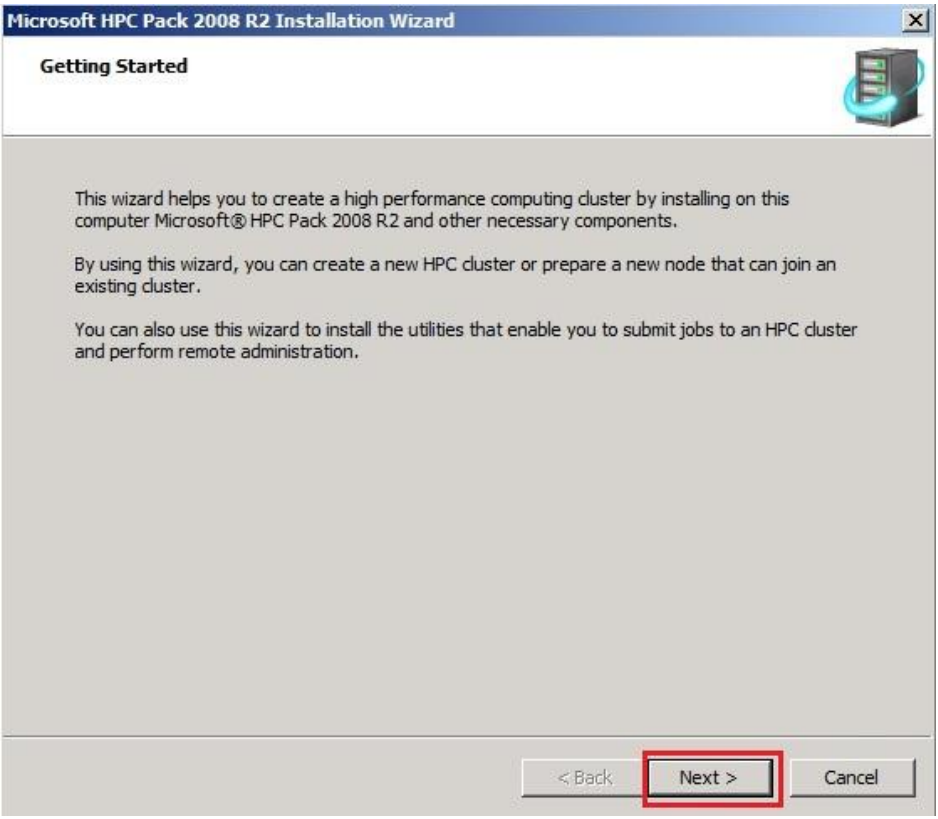



*Figure 1-3 Installation Wizard – Step 1*

2. Click **Next**, and the installation window shown in Figure 1-4 is displayed. Select the "HPC Pack 2008 R2 Enterprise and HPC Pack 2008 R2 for Workstation" check box.

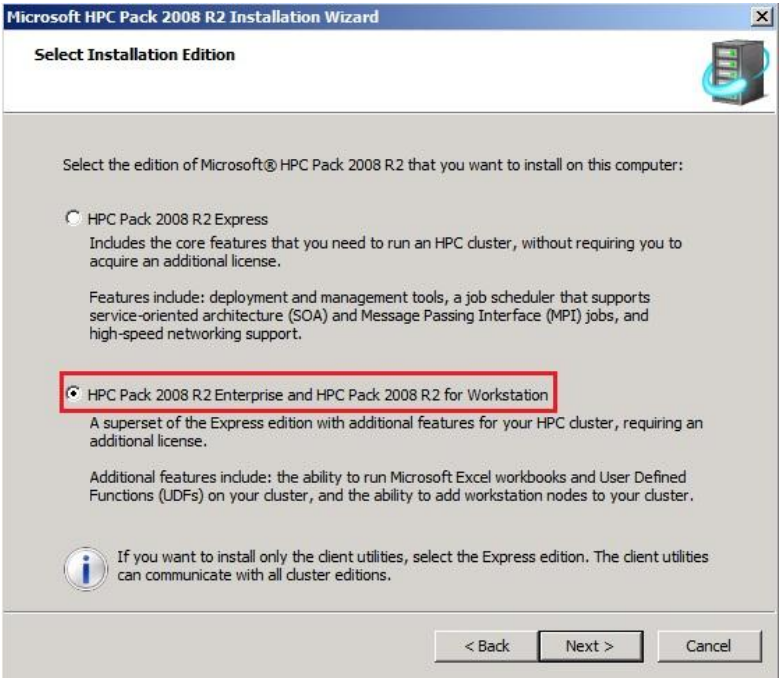



*Figure 1-4 Select Installation Edition*

3. Click **Next**. The installation window shown in Figure 1-5 is displayed. Read the Microsoft Software License Terms. If you agree, select the "I accept the terms in the license agreement" check box.

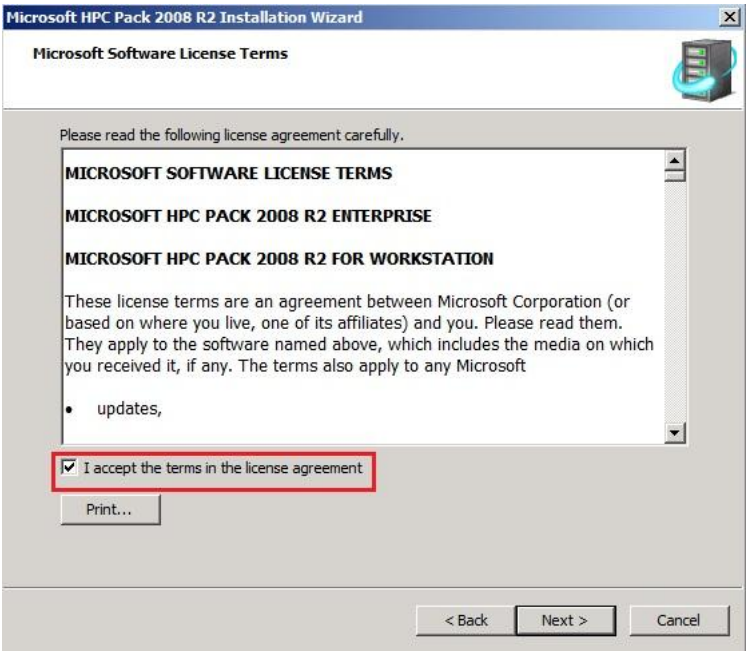



*Figure 1-5 Installation Wizard — Software License Window*

4. Click **Next**. The installation window shown in Figure 1-6 is displayed. This page is for selecting the type of installation. Select "Create a new HPC cluster by creating a head node".

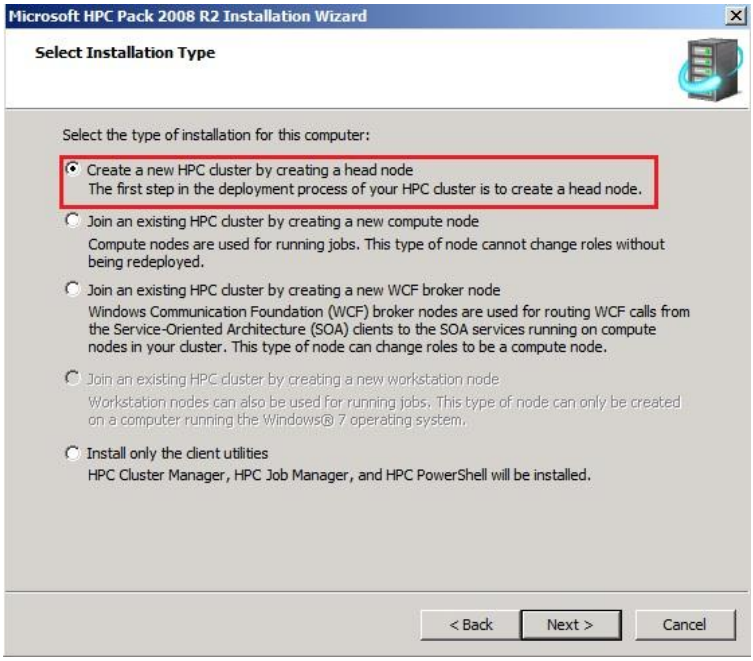



*Figure 1-6 Installation Wizard — Select Installation Type*

5. Click **Next**. The installation window shown in Figure 1-7 is displayed. This page is for selecting the HPC database location. Keep the default settings on this page.

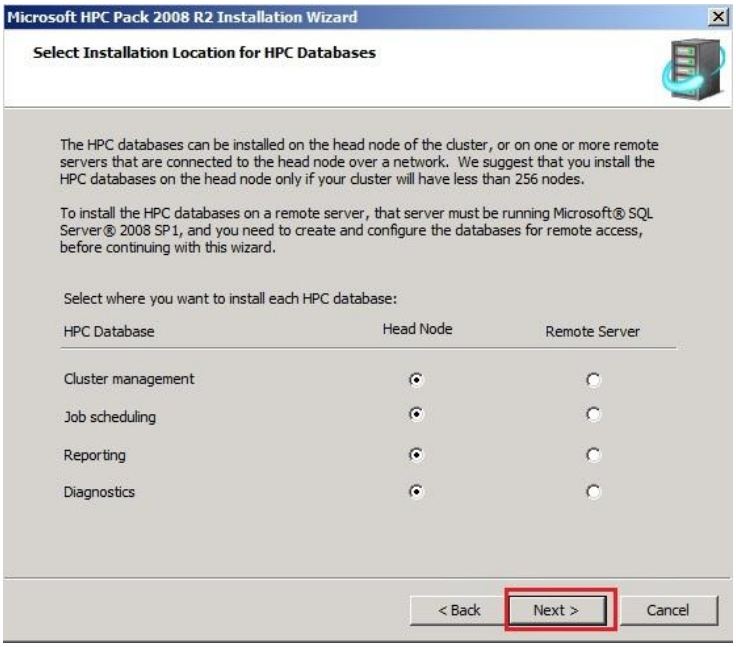



*Figure 1-7 Installation Wizard – Select Installation Location for HPC Databases*

6. Click **Next**. The "Instance for Local HPC Database" page as shown in Figure 1-8 is displayed. If there are SQL Server 2008 SP1 instances installed on the machine, a new page will pop up to let you select to use one SQL server instance, so that you don't need to install a new SQL server instance.

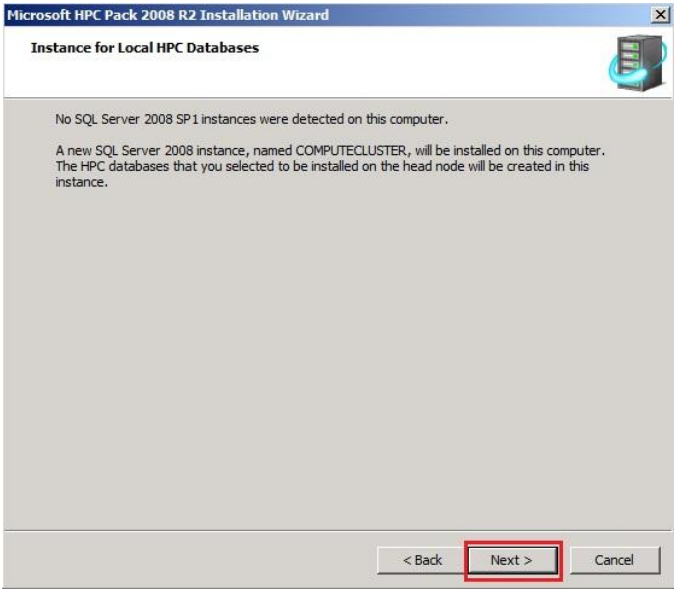



*Figure 1-8 Installation Wizard — Local HPC Database Instances*

7. Click **Next**. The page shown in Figure 1-9 is displayed. Select "Install all databases in the same folders".

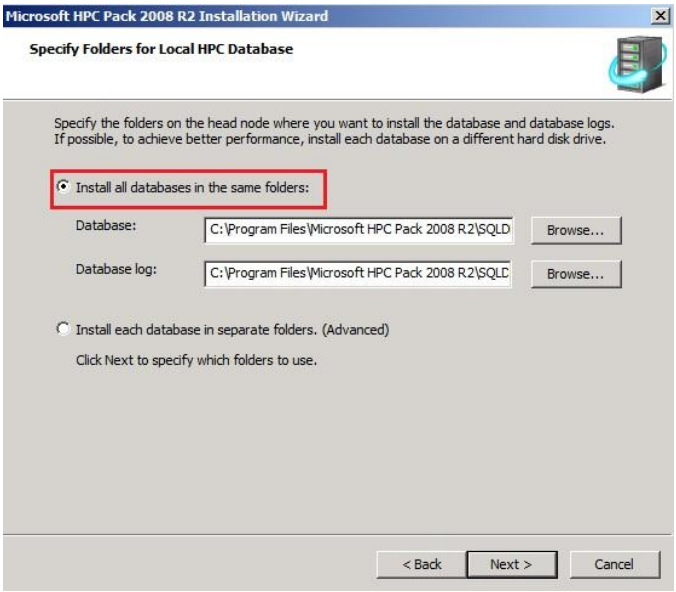



*Figure 1-9 Installation Wizard – Specify Installation Folders*

8. Click **Next**. The installation page shown in Figure 1-10 is displayed. Keep the default installation locations for this page.

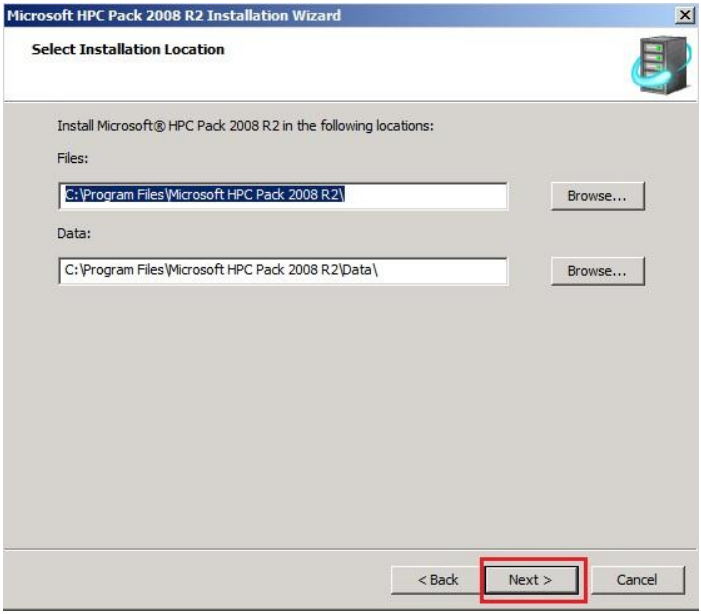



*Figure 1-10 Installation Wizard — Select Installation Location*

9. Click —. The installation page shown in Figure 1-11 is displayed. This page is for selecting whether or not to turn on Microsoft Update. Select "Use Microsoft Update when I check for updates (recommended)".

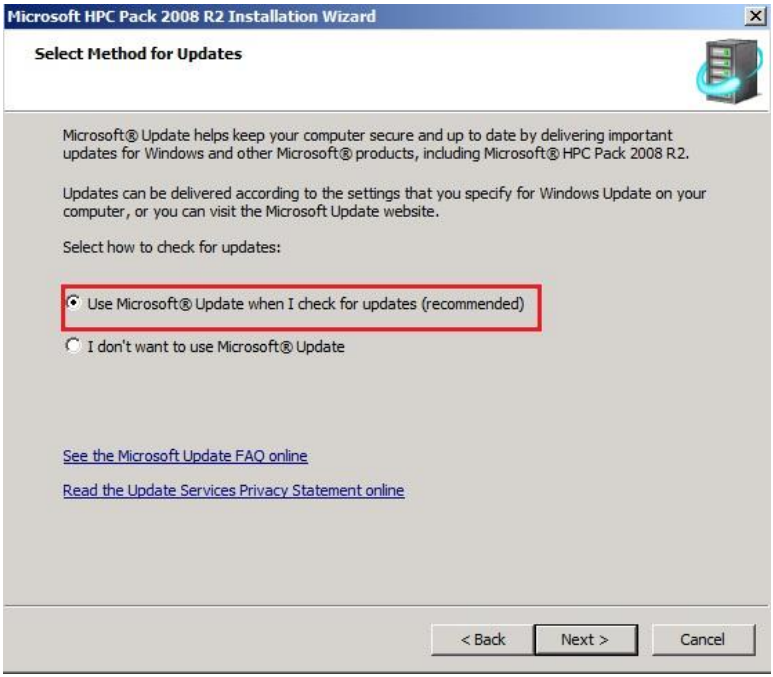



*Figure 1-11 Installation Wizard – Select Methods for Updates*

10. Click **Next**. The installation page shown in Figure 1-12 is displayed. This page is about the "Customer Experience Improvement Program" setting. With the user's permission, anonymous information will be sent to Microsoft for the purpose of improving the quality, reliability, and performance. Select the "Yes, I want to anonymously participate in the program" option.

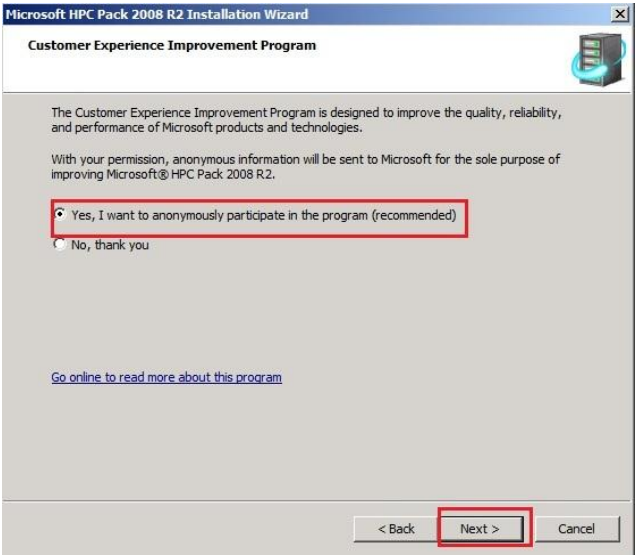



*Figure 1-12 Installation Wizard – Customer Experience Improvement Program*

11. Click **Next**. The installation page shown in Figure 1-13 is displayed. Click **Install** and the setup program will install all prerequisite components one by one, including .NET Framework 3.5 SP1 and Windows PowerShell v2.

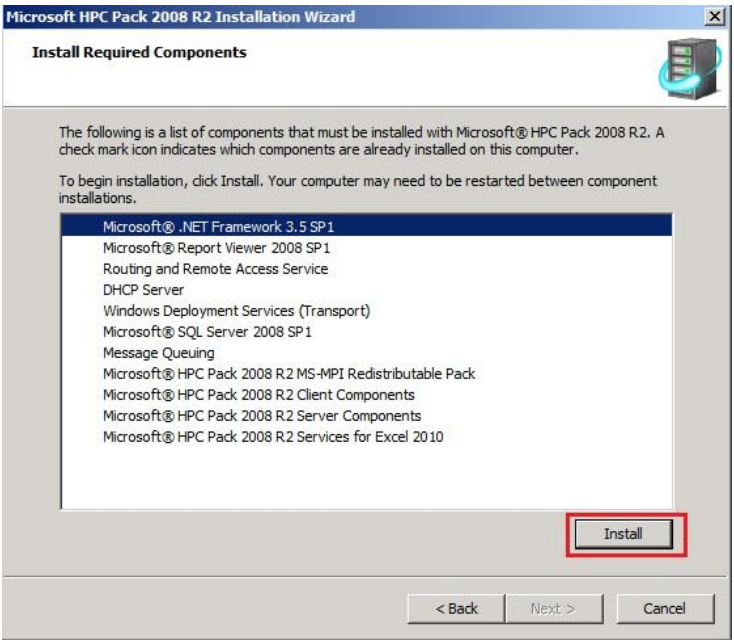



*Figure 1-13 Installation Wizard – Install Required Components*

12. After all prerequisites are installed, click **Install**. The dialog box shown in Figure 1-14 is displayed to show the progress of the installation.

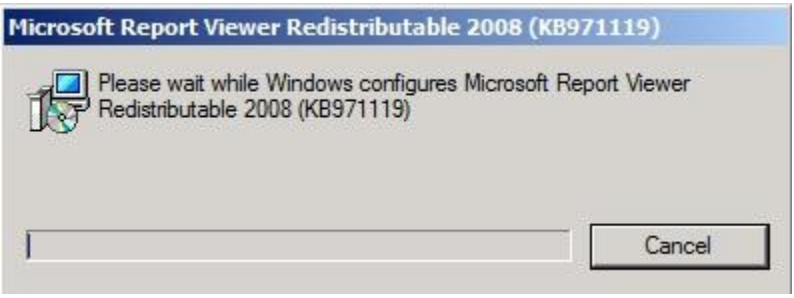



*Figure 1-14 Installation Progress*

13. When installation completes, the finish page shown in Figure 1-15 is displayed. Select "Start HPC Cluster Manager", then click **Finish**.

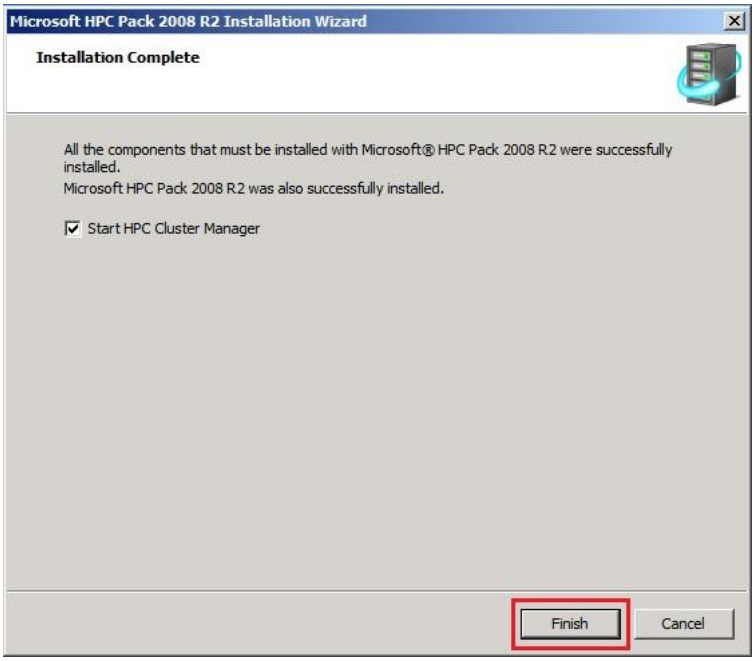



*Figure 1-15 Installation Wizard — Installation Complete*

The next step is to configure the cluster.

## 1.2.2 Configure the Cluster

When the HPC Cluster Manager is launched for the first time, a to-do list will show, including all steps that are necessary for configuring a cluster, as shown in Figure 1-16. The steps include configuring the network topology, configuring automatic deployment, and adding the compute node into the cluster. Windows HPC Server 2008 R2 introduces the node template, which is a convenient way to define the node configuration. The node template comes with an easy-to-use user interface. It leverages the Windows deployment service interface in Windows Server 2008 R2, and it can deploy the operating system and applications to all nodes or a set of nodes in the cluster. Also, it supports trace during deployment. The Windows HPC Server 2008 R2 administration console contains diagnostics test, which allows administrators to test node-to-node connectivity, node workload, and job status.

**Product Design Story: To-do list**

Why pop up the to-do list instead of popping up the administration console? It relates to one principle of Microsoft product design: understand the persona. To really push High Performance Computing into the mainstream, we expect cluster administrators to know how to administrate Windows instead of network configuration. Without such a to-do list, administrators may feel confused when facing our administration console directly and may not know what to configure first to get their cluster running. For example, there is dependency between different configurations. Deployment setting depends on network topology configuration. Some configurations are fundamental for creating a cluster, in which network topology configuration is important.

The to-do list design was applied in Windows Small Business Server successfully and received positive user feedback.

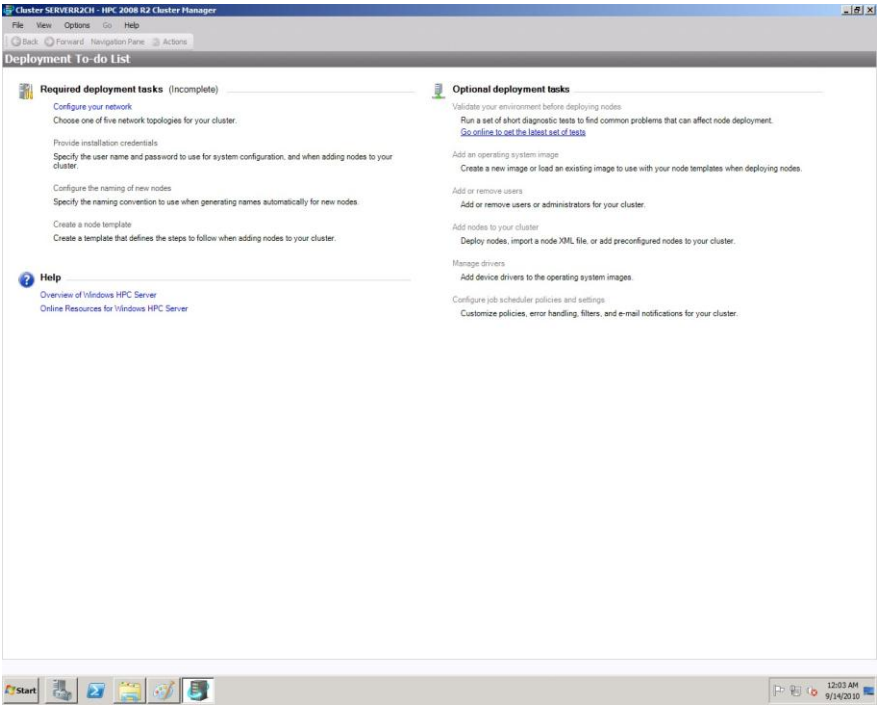



*Figure 1-16 Cluster Manager To-do List*

#### 1.2.2.1 Configure the Network

**Product Design Story: Network Topology**

It is always hard to do network configuration on a server with multiple network cards installed, which involves configuring IP address assignment and resolution, Network Address Translation (NAT), and firewall. As Linux administrators know, many configuration files need to be updated in the process — /etc/dhcp.conf, /etc/dnsd.conf, /etc/resolv.conf, /etc/natd.conf, and /etc/rc.firewall. Many errors come from configuring the wrong network card. How can you enable administrators with less network configuration experience to configure network topology easily and correctly? The HPC group worked hard to get the design right.

Do the following to configure the network using a wizard.

1. Select "Configure your network" and choose "5. All Nodes only on an enterprise network".
   Windows HPC Server 2008 R2 supports five types of different network topologies (as shown in Figure 1-17). Users can configure a network based on their hardware environment, infrastructure, and cluster requirement. Each cluster node is equipped with one to three network interface cards in these five network topologies. The network configuration wizard (as shown in Figure 1-17) simplifies the configuration process.

   *Table 1-1 Cluster network topology options description*

| Network topology | Network topology description |
|---|---|
| Compute nodes isolated on a private network | The head node is equipped with two network cards, one that connects to a private network and the other that connects to an enterprise network. The head node provides NAT service to compute nodes. |
| All nodes on enterprise and private networks | One network card connects to an enterprise network, and the other one connects to a private network. |
| Compute nodes isolated on private and application networks | The head node is equipped with three network cards: one connects to an enterprise network, another one connects to a private network, and another connects to an application network. The head node may provide NAT service in-between the private network and the enterprise network. Each compute node connects to the private network and an application network that supports a high-speed network protocol, such as the Message Passing Interface (MPI). |
| All nodes on enterprise, private, and application networks | One network card connects to an enterprise network, another connects to a private management network, and another connects to a high-speed application network. |
| All nodes only on enterprise network | In this restricted network topology, only one network card is installed on each node. Deploying the compute node with Windows Deployment Service is not supported. The user needs to install each compute node manually and activate them. |

> **Note**
>
> In Topology #1 and Topology #3, compute nodes are isolated on a private network, in which case NAT service in-between the private network and the enterprise network could be served on the head node or on a separate server.

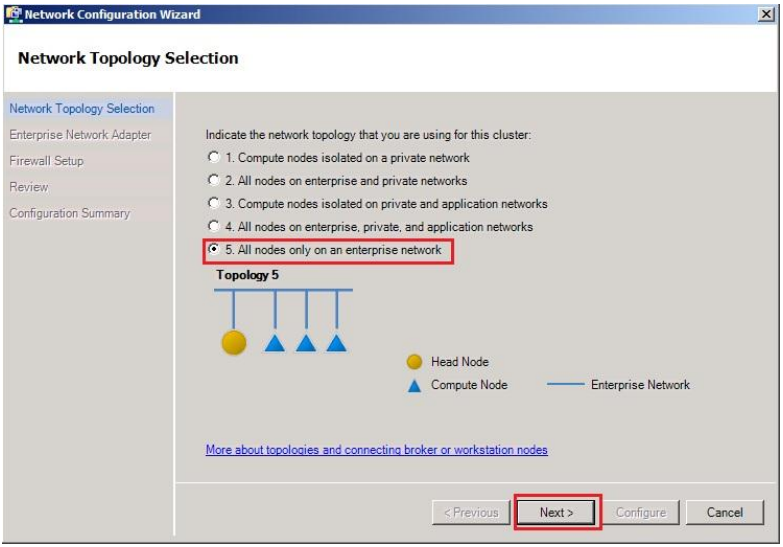



*Figure 1-17 Network Topology Selection*

2. Click **Next** to enter the **Firewall Setup** page, as shown in Figure 1-18. Select "Automatically apply firewall settings to the networks and nodes in the cluster" and select "ON" to the right of the "Enterprise network" label.

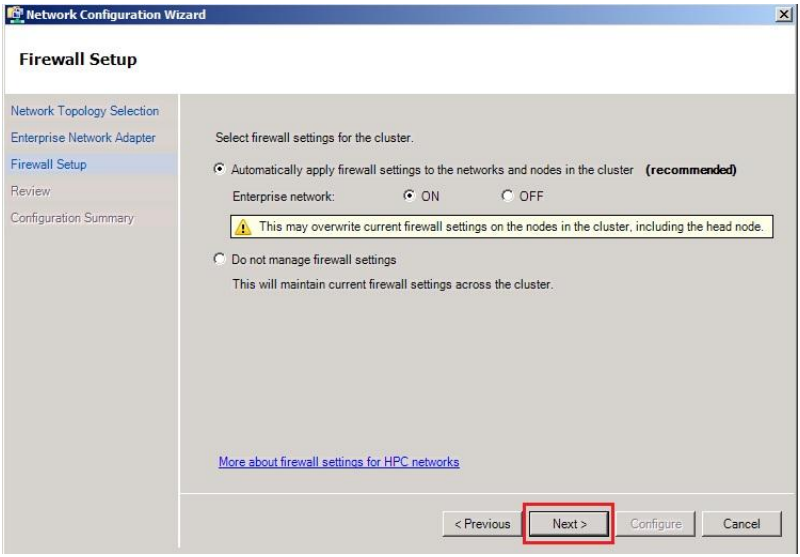



*Figure 1-18 Firewall Settings*

3. Click **Next**. The "Review" page shown in Figure 1-19 is displayed. Click **Configure** on the page, and the "Network Configuration Progress" page shown in Figure 1-20 is displayed.

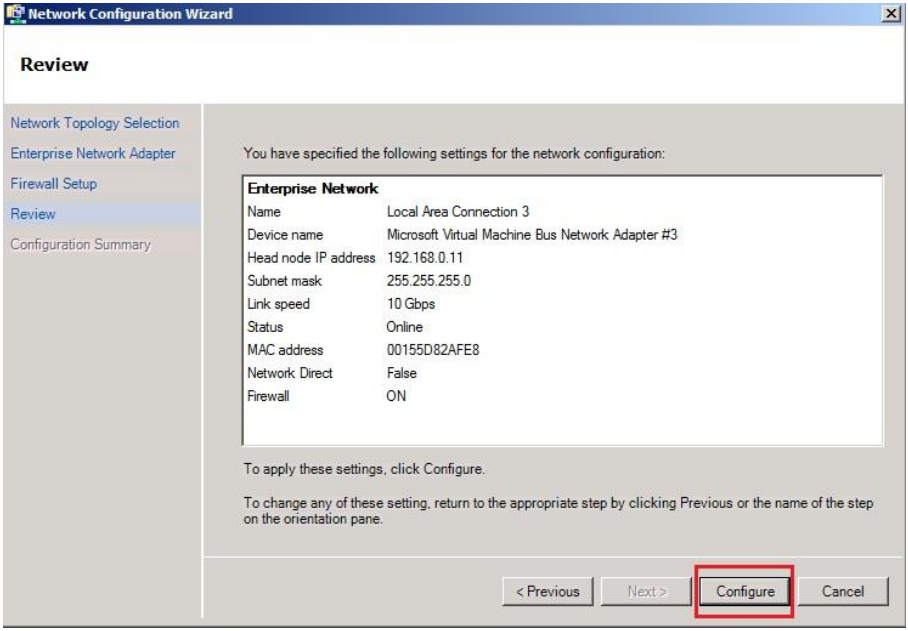



*Figure 1-19 Review the Network Configuration*

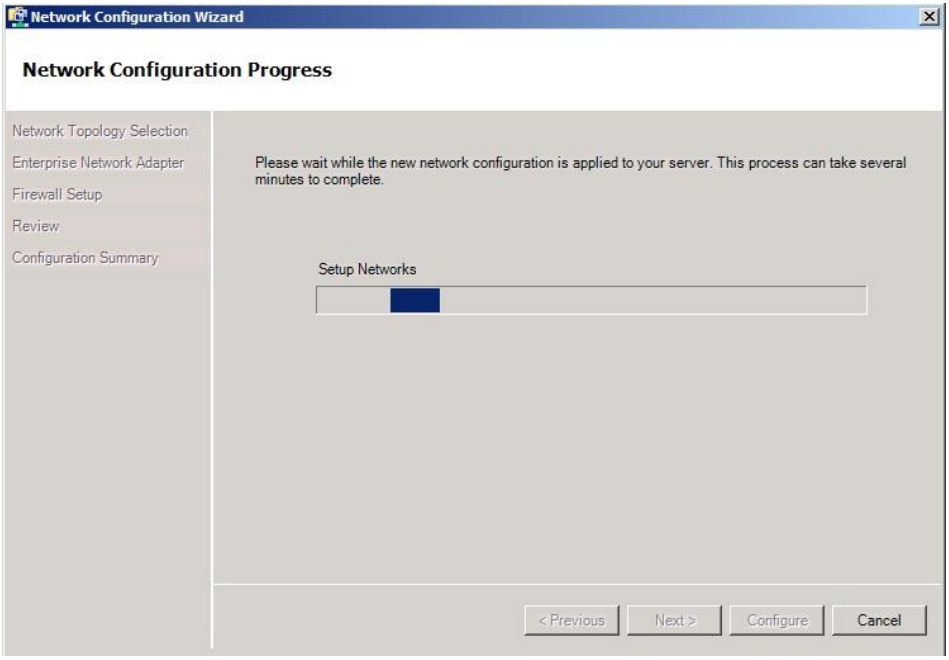



*Figure 1-20 Network Configuration Progress*

4. After configuration completes, the "Configuration Summary" page shown in Figure 1-21 is displayed. In this "Configuration Summary" page, ensure that the result you get is the same as in the figure: setup firewall configuration and setup networks tasks are successfully executed. Then click **Finish** to finish network configuration.

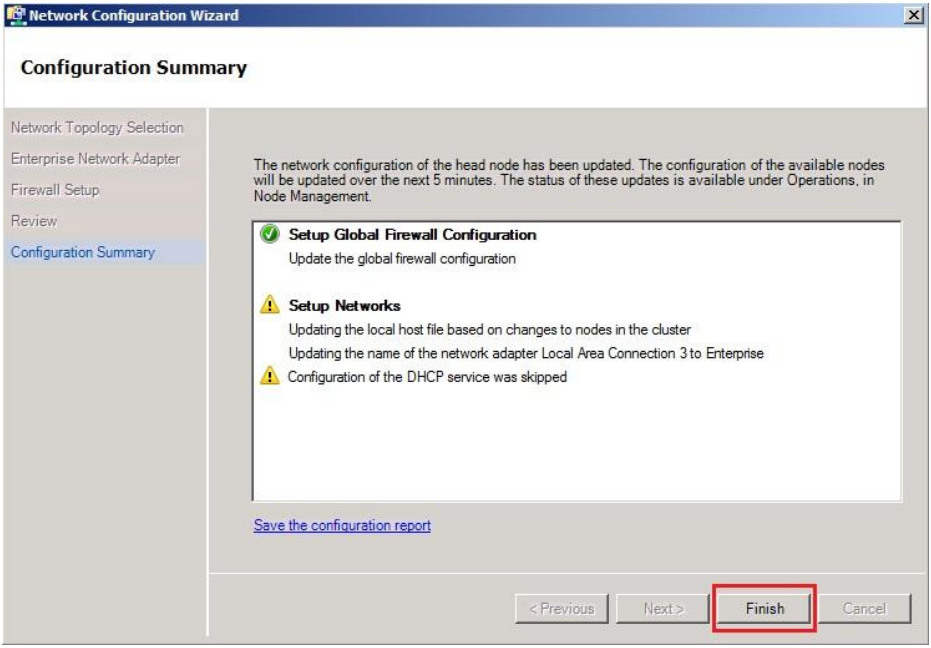



*Figure 1-21 Configuration Summary*

### 1.2.2.2 Provide Installation Credentials

Some cluster operations are conducted in the background, such as system configuration, joining a node into the domain, and diagnostics tests. The Cluster system needs to use the system administrator's credentials to get the required permission and conduct background operations on compute nodes. The second step of the to-do list is to collect the system administrator's credentials.

After setting up the network, the system user interface looks like that shown in Figure 1-22. Click **Provide installation credentials**, and the **Installation Credentials** dialog box shown in Figure 1-23 is displayed.

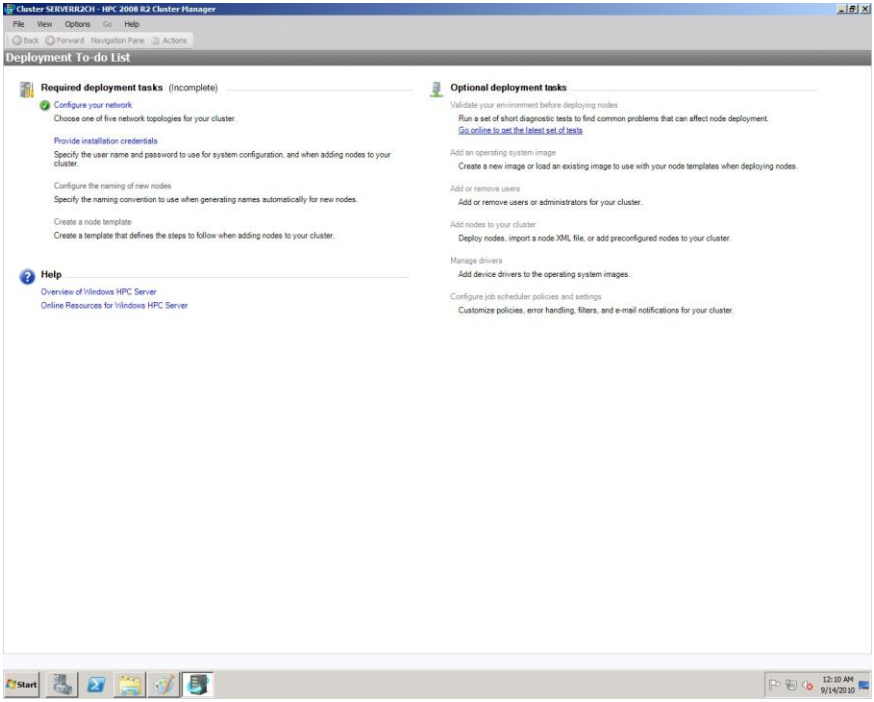



*Figure 1-22 Deployment To-do List*

*Figure 1-23 Installation Credentials*

Fill in the user name and password in the dialog box and click **OK** to finish this step.

### 1.2.2.3 Configure the Naming of New Nodes

Windows HPC Server supports thousands of nodes in a cluster. To name each node one by one when installing nodes would be tedious. Windows HPC Server provides node naming series, which is used to generate names automatically for new nodes added to a cluster. Its format is *prefix%STARTNUM%*. For example, SHPC%100% defines a series of SHPC100, SHPC101, SHPC102, and continuing up to SHPC999.

In this chapter, we will install compute nodes manually, so any naming series works for us. As shown in Figure 1-24, click **OK**.

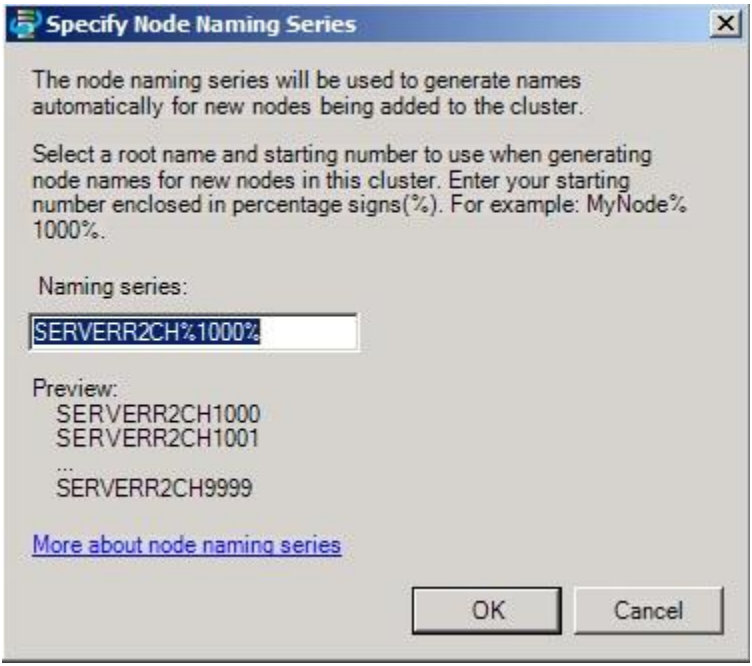



*Figure 1-24 Specify Node Naming Series*

### 1.2.2.4 Create Node Template

To install and configure compute nodes involves several steps, such as creating a computer account, partitioning a hard disk, installing the operating system, and joining the domain. A node template is for defining these operations. Administrators can use the node template creation wizard to create node templates. For advanced configuration settings, the node template editor provides extra functions, including configuring the node template used for automated deployment. The Windows HPC Pack 2008 R2 node template creation wizard also supports injecting drivers into the operating system image.

The Windows Deployment Service uses the Windows Imaging Format (WIM) file and multicast function to deploy nodes quickly and in parallel. Using node template, administrators can create node templates to deploy different types of nodes. Node templates contain basic operating system image, drivers, configurations, and extra applications.

The following procedure describes how to create a node template.

1. On the "Choose Node Template Type" page, select "Compute node template", as shown in Figure 1-24.

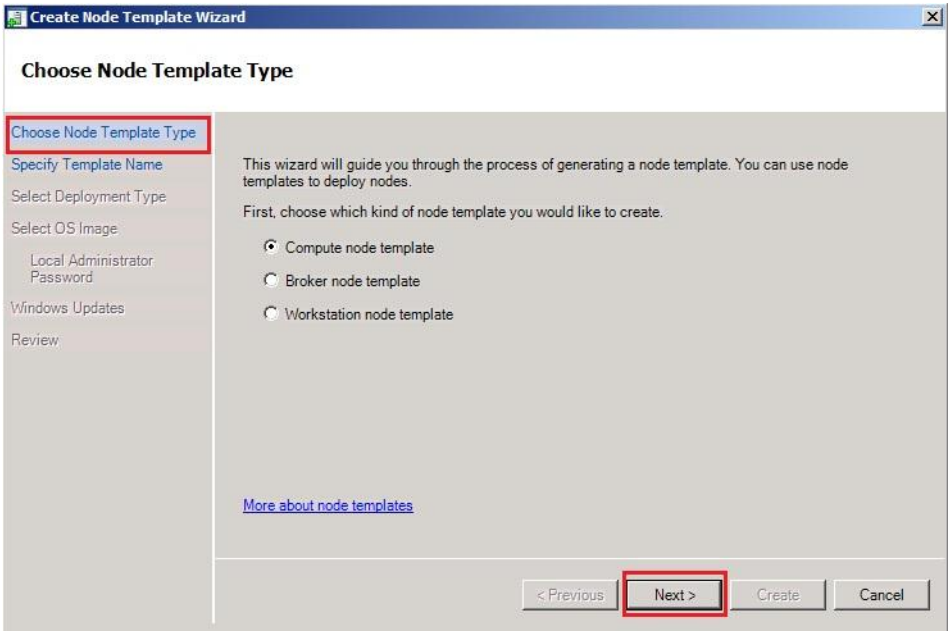



*Figure 1-25 Choose Node Template Type*

2. Click **Next**. The "Specify Template Name" page shown in Figure 1-26 is displayed. Keep the default name as "Default ComputeNode Template".

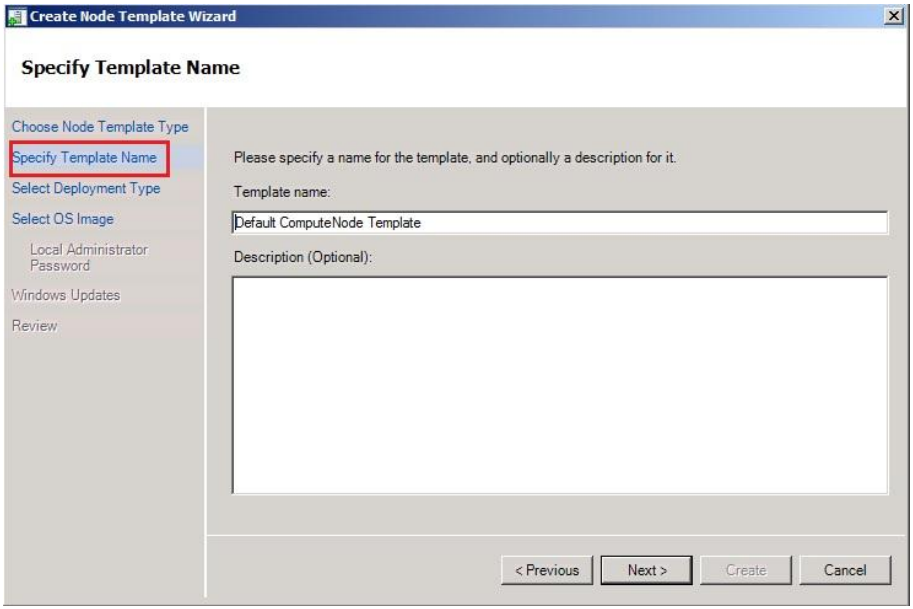



*Figure 1-26 Specify Template Name*

3. Click **Next**. The "Select Deployment Type" page shown in Figure 1-27 is displayed. We will install the operating system manually in this chapter, so select "Without operating system" on this page. To understand how to use bare metal deployment, see Chapter 6, "Manage Windows HPC Cluster".

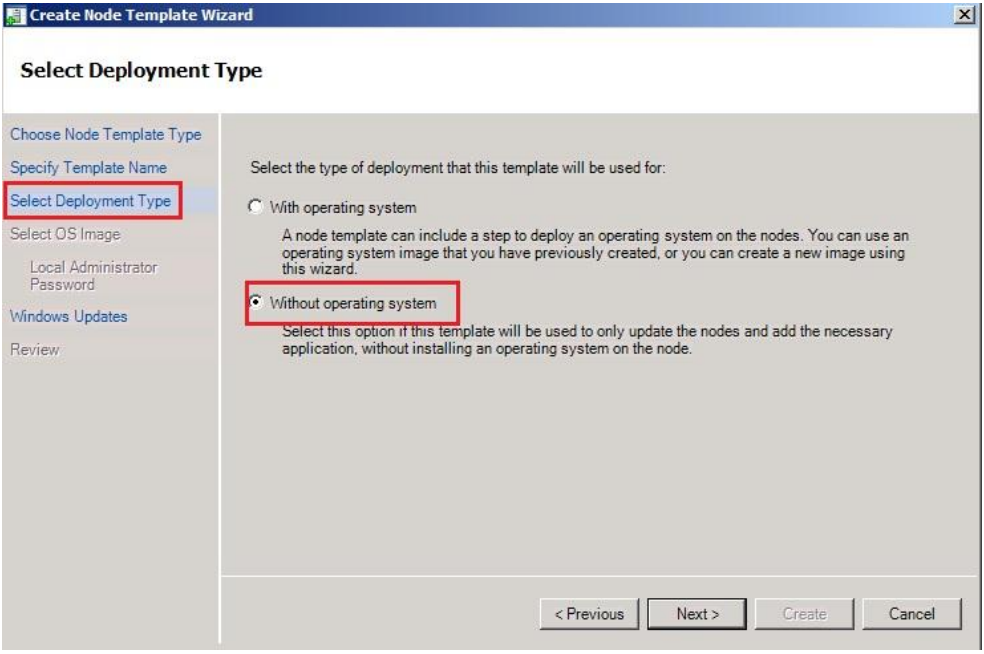



*Figure 1-27 Select Deployment Type*

4. Click **Next**. The "Specify Windows Updates" page shown in Figure 1-28 is displayed. Select "Include a step in the template to download and install updates for my cluster using Microsoft Update" and select "Install only critical and security updates".

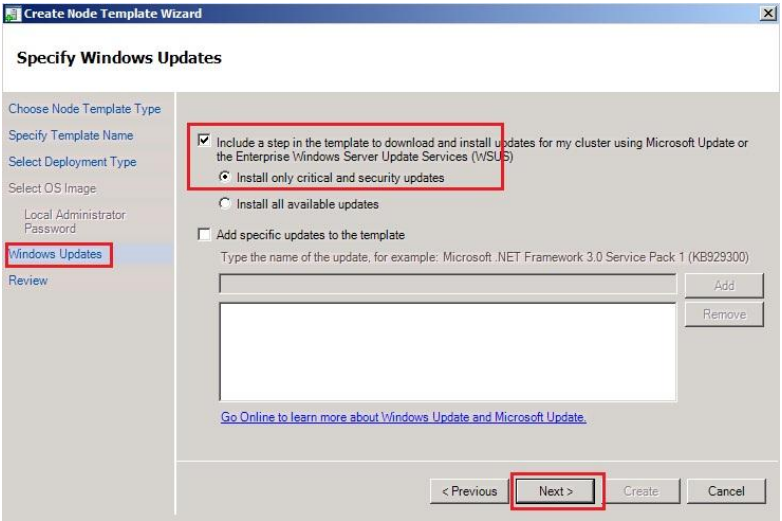



*Figure 1-28 Specify Windows Updates*

5. Click **Next**. The "Review" page shown in Figure 1-29 is displayed. The settings to review are shown in Figure 1-29. Click **Create** to proceed to node template creation.

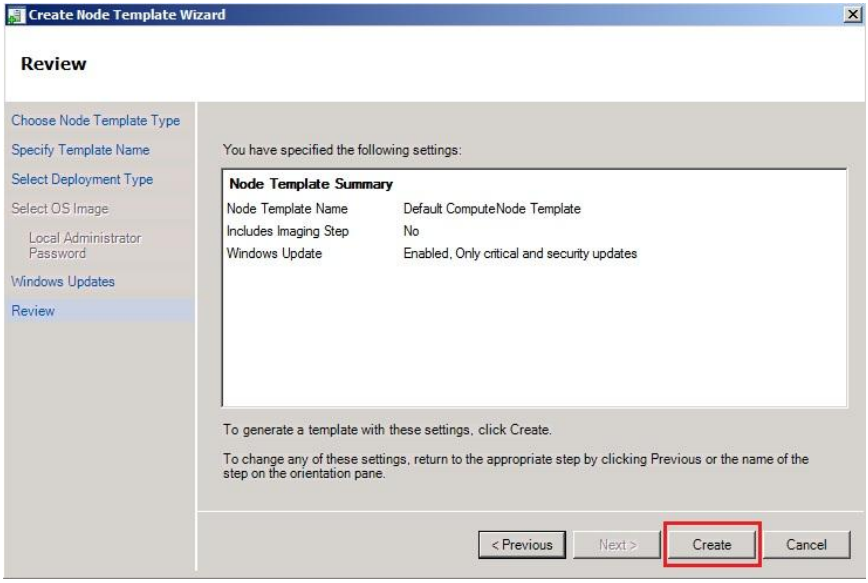



*Figure 1-29 Review*

When creation finishes, the bottom left side of the cluster management console will show a Configuration, Node Management, Job Management, Diagnostics, and Chart and Reports wunderbar, as shown in Figure 1-30.

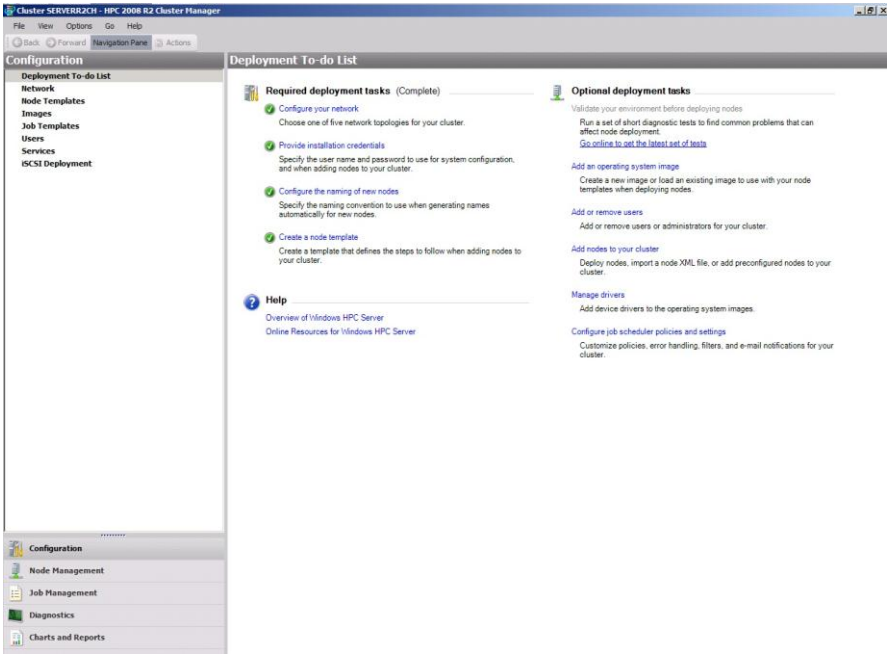



*Figure 1-30 Cluster Management Console User Interface Wunderbars*

### 1.2.3 Deploy the Compute Nodes

Before doing the following steps, manually install the Windows HPC Server 2008 R2 operating system on each compute node.

1. Insert the Windows HPC Pack 2008 R2 DVD into your DVD-ROM drive and double-click the "setup.exe" icon. Click **Next**, and the page shown in Figure 1-31 is displayed. Select "Join an existing HPC cluster by creating a new compute node".

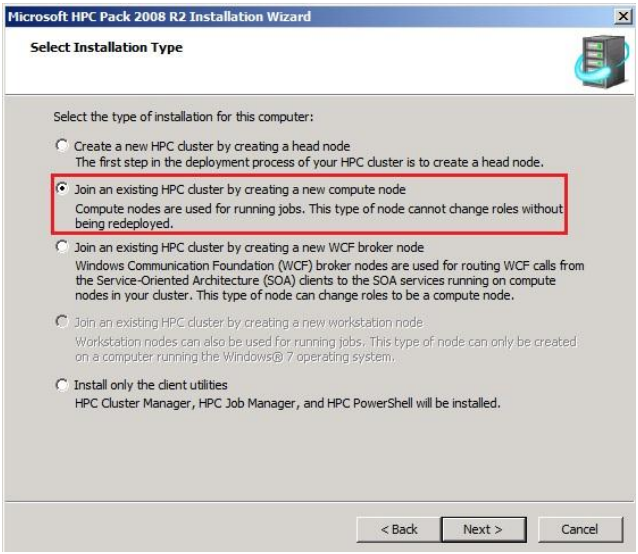



*Figure 1-31 Compute Node Installation Wizard*

2. Click **Next**. The "Join Cluster" page shown in Figure 1-32 is displayed. Enter your head node name.

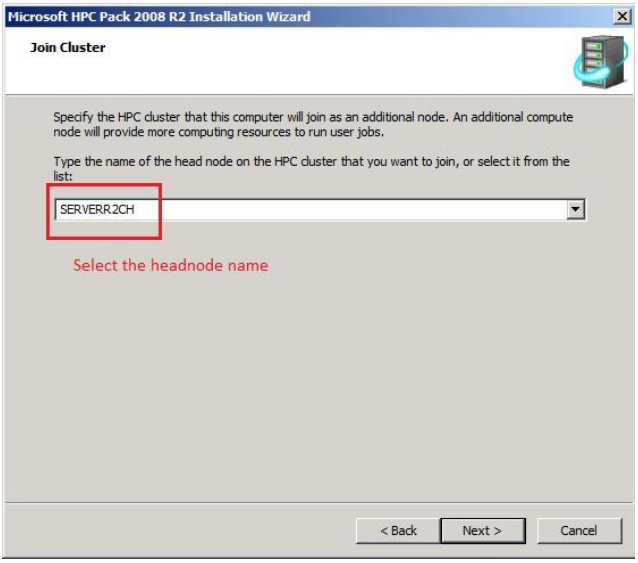



*Figure 1-32 Select Target Head Node Name*

3. Click **Next**. The "Select Installation Location" page shown in Figure 1-33 is displayed. Keep the default installation locations in this page.

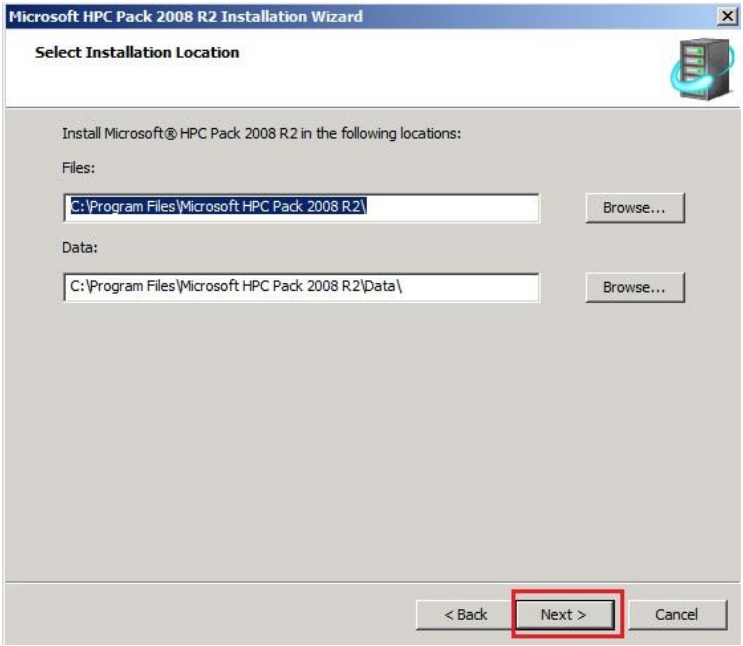



*Figure 1-33 Target Installation Locations*

4. Click **Next**. The "Install Required Components" page shown in Figure 1-34 is displayed. Installation prompts to install the listed components. Click **Install**.

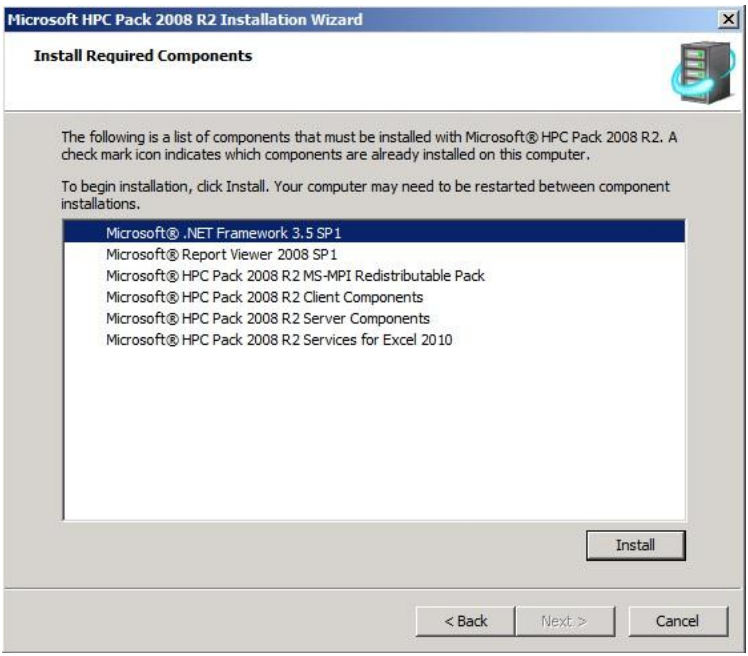



*Figure 1-34 Compute Node Required Components*

5. After installation completes, the "Installation Complete" page shown in Figure 1-35 is displayed. Click **Finish** to finish compute node deployment.

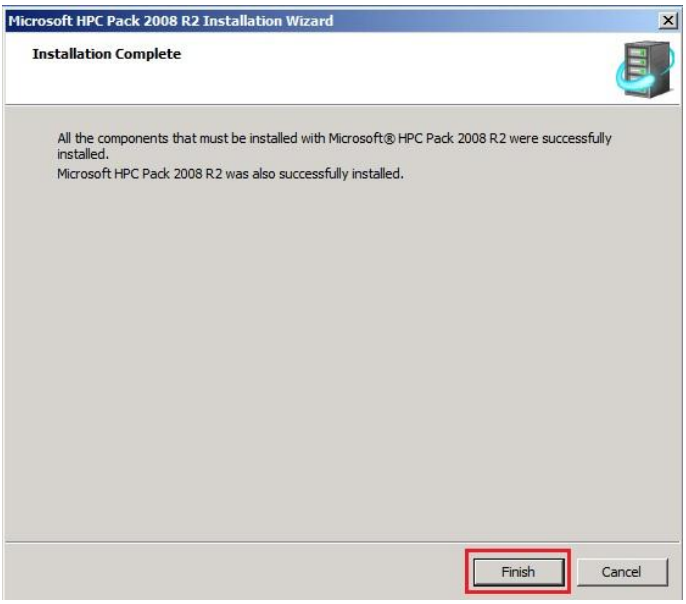



*Figure 1-35 Installation Complete*

Repeat step 1 to step 5 in this section to deploy other compute nodes.

## 1.3 Assign Node Template

Is the cluster ready after installing those compute nodes? No, we have to assign the node template. As stated in 0, the node template defines operations required by our management system. Compute nodes will be accepted by the cluster after applying those operations.

There are two "unknown" compute nodes in the "Cluster Manager" Console as shown in Figure 1-36.

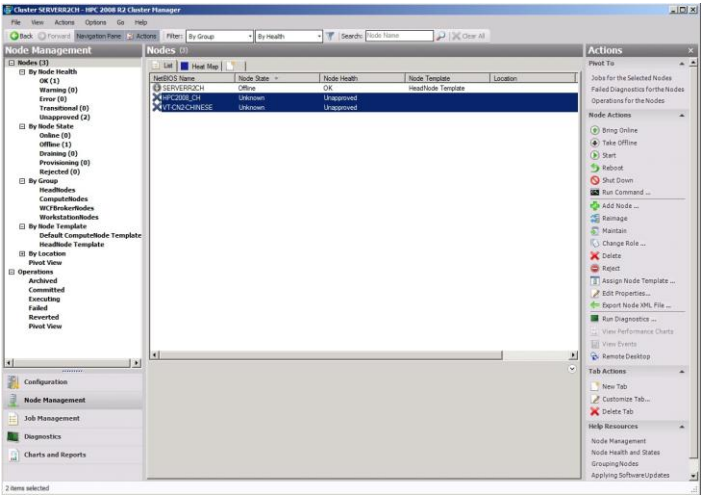



*Figure 1-36 Select Multiple Compute Nodes*

Let's join the "unknown" nodes to the cluster.

1. Select all compute nodes with an "unknown" state, right-click, and select "Assign Node Template …" from the context menu as shown in Figure 1-37.

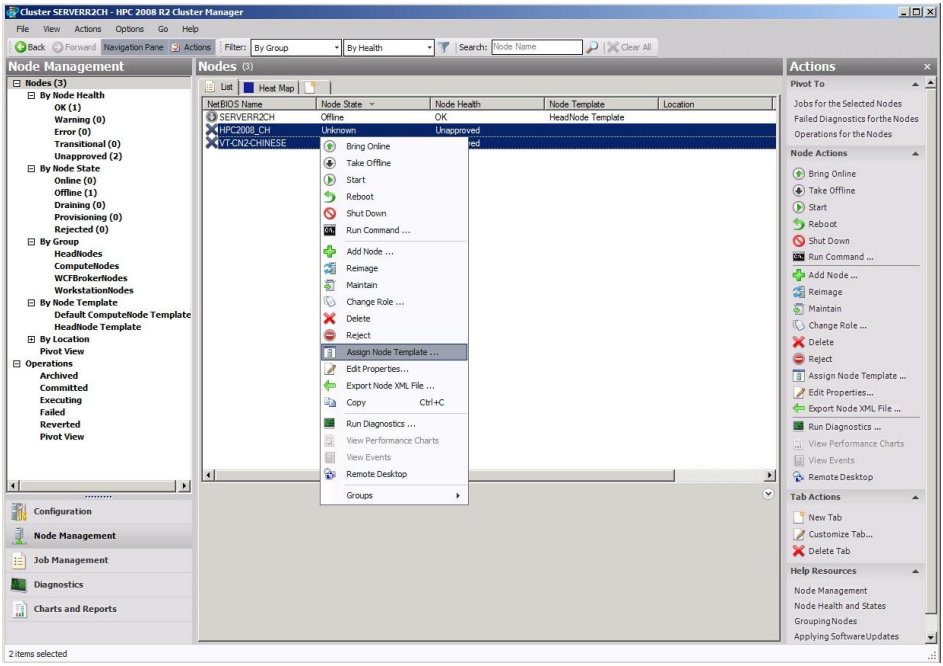



*Figure 1-37 Assign Node Template*

2. Select "Default ComputeNode Template" from the **Assign Node Template** dialog box as shown in Figure 1-38.

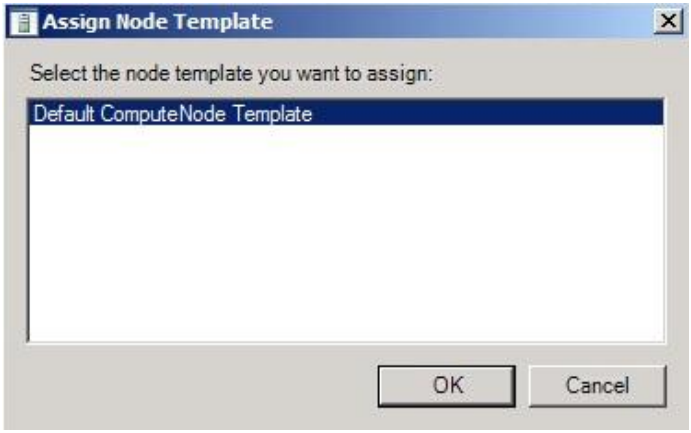



*Figure1-38 Assign Node Template*

3. Click **OK**. Compute nodes will go into "Provisioning" state as shown in Figure 1-39.

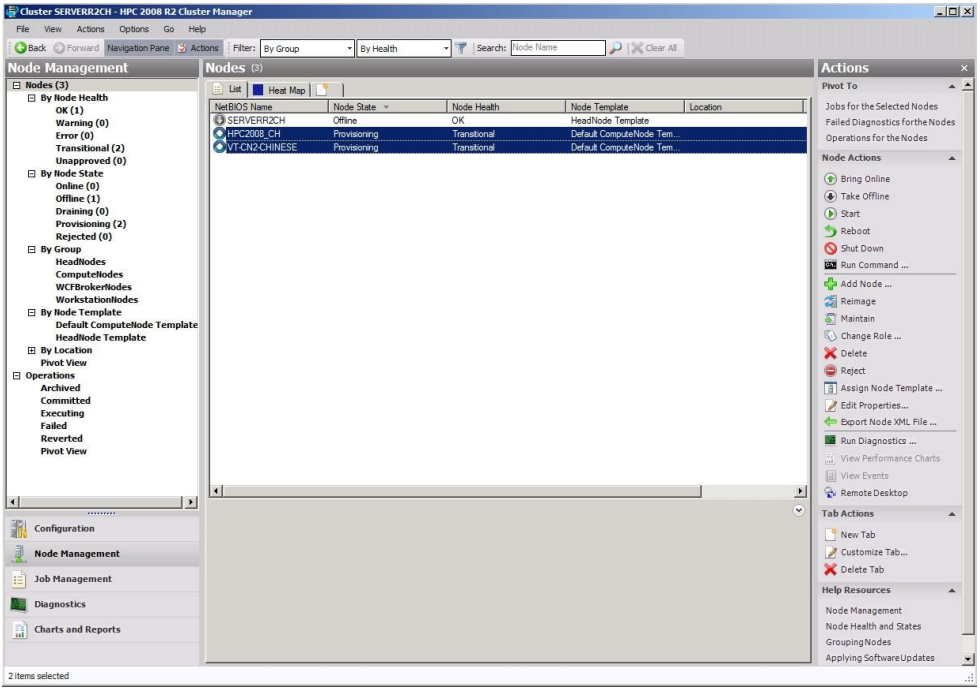



*Figure 1-39 Provisioning State*

4. A couple of seconds later, compute nodes become "Offline". Right-click and select "Bring Online" from the context menu, as shown in Figure 1-40.

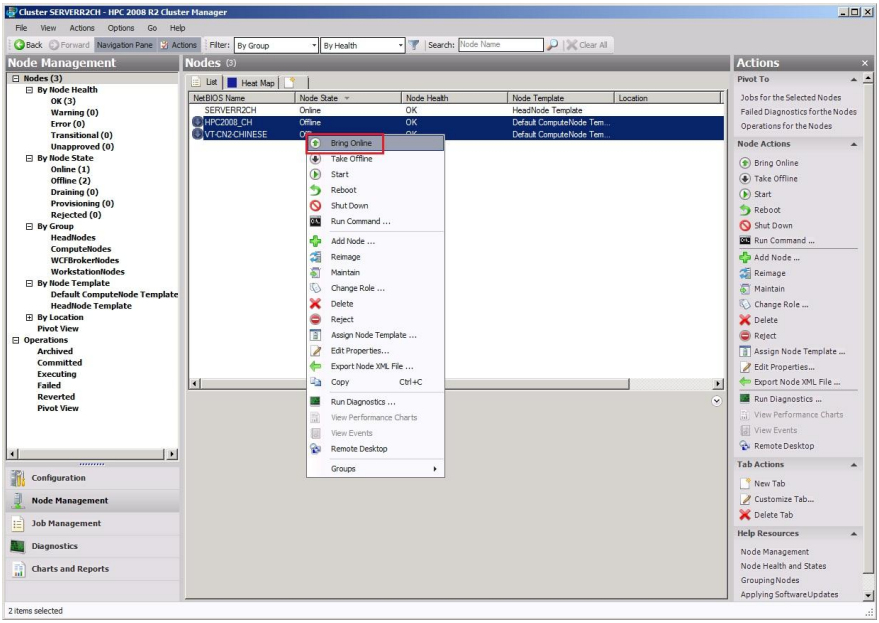



*Figure 1-40 Bring Compute Nodes Online*

5. All compute nodes become "Online".

Congratulations, you've built your Windows HPC Cluster.

## 1.4 Try the HPC Application

The first thing you want to do after setting up the cluster is to run an application to test the newly set up cluster infrastructure and demonstrate the acceleration effect of the HPC Server. There are a lot of HPC applications in the CD coming with the book; the demo here is an Asian option pricing program.

### 1.4.1 Introduction to the Asian Option Pricing program

An Asian option (or average value option) is a special type of option contract. For Asian options, the payoff is determined by the average underlying price over some pre-set period of time. This pricing application uses the derivative pricing algorithm by Monte Carlo simulation. The computation required by this method is correlated to the number of iterations. We'll run a serialized version of this Asian option pricing program, and run a cluster version to compare the speed to result.

This application requires Microsoft® Office 2007 on the desktop computer.

### 1.4.2 Run the Demo

1. Browse to the demo program of this book. Go to "Chapter 1\Service".
2. Edit the file AsianOptionsService.config, change "YOURHEADNODE" (marked bold) to the name of the cluster you just set up.

```
<?xml version="1.0" encoding="utf-8" ?>

<configuration>


 <configSections>


  <!--Register service's custom configuration sections and group-->

  <sectionGroup name="microsoft.Hpc.Session.ServiceRegistration"


type="Microsoft.Hpc.Scheduler.Session.Configuration.ServiceRegistration,
Microsoft.Hpc.Scheduler.Session, Version=2.0.0.0, Culture=neutral,
PublicKeyToken=31bf3856ad364e35">

   <section name="service"


type="Microsoft.Hpc.Scheduler.Session.Configuration.ServiceConfiguration,
Microsoft.Hpc.Scheduler.Session, Version=2.0.0.0, Culture=neutral,
PublicKeyToken=31bf3856ad364e35"

        allowDefinition="Everywhere"

        allowExeDefinition="MachineToApplication"

        />

  </sectionGroup>


 </configSections>
```

```
 <microsoft.Hpc.Session.ServiceRegistration>

  <service
assembly="\\YOURHEADNODE\HpcServiceRegistration\AsianOptionsService.dll"
/>

 </microsoft.Hpc.Session.ServiceRegistration>

 <system.serviceModel>

  <bindings>

   <netTcpBinding>

    <!--binding used by broker's front end service with security-->

    <binding name="Microsoft.Hpc.SecureNetTcpBrokerBinding"
portSharingEnabled="true" closeTimeout="0:00:05" maxConnections="1000">

     <security mode="Transport">

      <transport clientCredentialType="Windows"
protectionLevel="None"></transport>

     </security>

    </binding>

   </netTcpBinding>

  </bindings>

 </system.serviceModel>


</configuration>
```

*Code 1-1 Change head node name*

1. Copy all files under "Chapter 1\Service" to \\YOURHEADNODE\HpcServiceRegistration\
2. Go to "Chapter 1\Workbook", open AsianOptionPricingCSharpBehind.xlsx, as shown in Figure 1-41. Set "Run on Cluster" to "No", and click **Run**. The program finishes in about 46 seconds on my laptop with a 1.2GHz CPU.

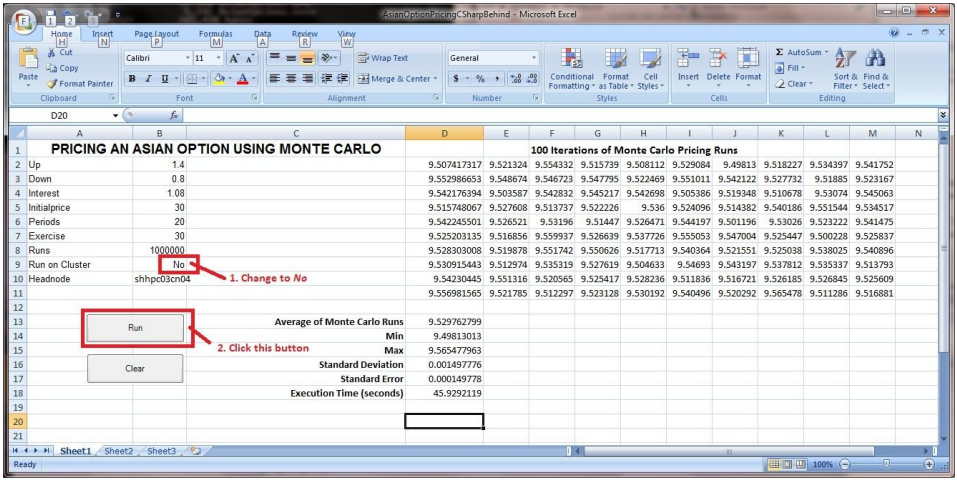



*Figure 1-41 Run the serial version of Asian Option Pricing*

3. Click **Clear** to clean the result.

4. Set "Run on Cluster" to "Yes". Setup "Headnode" to the name of the cluster that was just set up, as shown in Figure 1-42. Click **Run**. On a cluster with three nodes and 12 cores, the program finishes in about 4.5 seconds, as shown in figure 1-43. The speedup is 46/4.5 = 10.

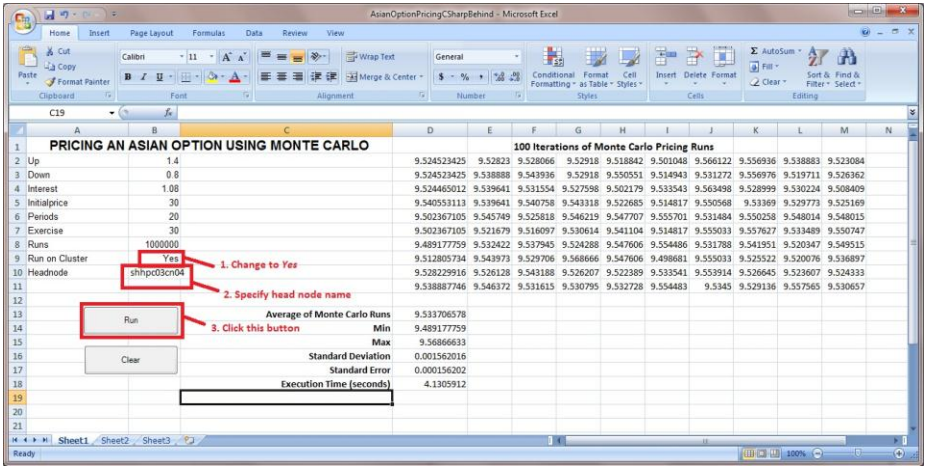



*Figure 1-42 Run Asian Option Pricing on a cluster*

# Chapter 2 Introduction of Windows HPC Server 2008 R2

To help readers understand the subsequent chapters more easily, we need to introduce the functions of HPC Server. HPC Server provides high-performance application programming interfaces, job scheduling, and management systems to ensure the balance between supply and demand of HPC resources. Programmers can efficiently develop, debug, and optimize cluster applications by various HPC development models and a friendly development environment. Administrators can effectively manage resources and jobs by using features such as automatically deployment, visual monitoring, job scheduling policy management, built-in customizable diagnosis, and reporting. User can easily submit and manage jobs by the HPC integrated applications, job submission, and management console and scripts.

Chapter Highlights:

- Overview of Windows HPC Server 2008 R2 Features
- Overview of MS MPI
- Service-oriented programming model
- Overview of job scheduling
- Management Console functional overview

## 2.1 Windows HPC Server 2008 R2

There are two DVDs in the Windows HPC Server 2008 R2 box. The first DVD contains the 64-bit versions of Windows Server 2008 R2 (HPC Edition) setup files. The second DVD contains Microsoft HPC Pack 2008 R2 and provides the API and job scheduling and resource management systems, as shown in Figure 2-1.

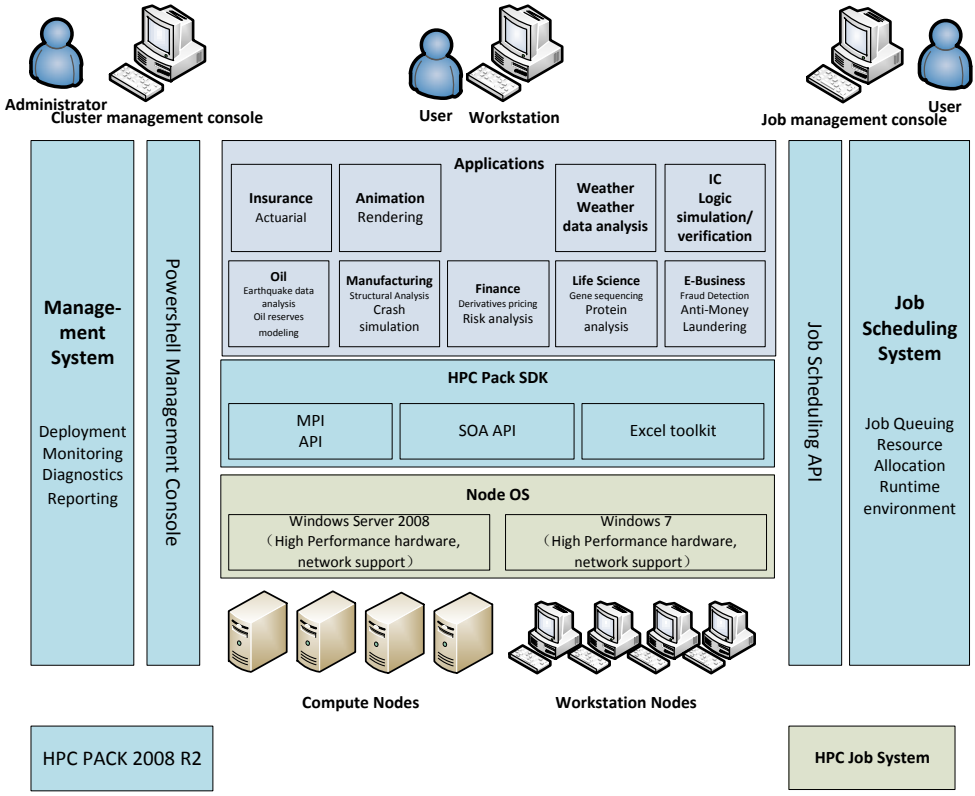



*Figure 2-1 HPC Server Stack*

HPC Server provides high-performance APIs, job scheduling, and management systems to ensure balance between supply and demand of HPC resources. On the supply side, the management system ensures that resources are deployed properly before usage, are kept healthy during usage, and are diagnosed as soon as possible when a fault occurs, and helps administrators to generate and analyze resource utilization reports regularly. On the demand side, with the help of the programming model, developers can express and encapsulate application parallel logic, decompose application domain data; applications can use multiple processing units and synchronize multiple processes efficiently. Besides the supply and demand sides, a "matchmaker" broker is also necessary for the job scheduling system to reasonably allocate resources for multiple users and projects to fulfill the production design goal of the department and the enterprise.

The section provides a general description of three features of Windows HPC Server: the HPC Pack SDK, the management system, and the jobs scheduling system.

### 2.1.1 HPC Pack SDK

HPC Server offers several HPC development methods. There are different programming models for different needs and HPC resources. As long as a suitable model for distributed development is chosen, cluster applications can be developed efficiently.

HPC Server offers three models:

- MPI model based on message passing.
- Service Oriented Architecture (SOA)–based model.
- Excel HPC Service

#### 2.1.1.1 MPI model based on message passing

In the HPC field, MPI is the industry standard for writing parallel applications with the message-passing method. The MPI runtime system starts multiple processes on multiple nodes, with each process handling different parts of the problem domain. The MPI library defines a set of functions, which applications can use to send and receive messages or do global synchronization between processes. HPC Server and MPI integrate seamlessly so that users submit MPI applications without specifying compute nodes.

With the HPC Server plug-in on Microsoft® Visual Studio® 2008, programmers can develop and debug the MPI applications running on clusters on workstations. The built-in C++ projects of Microsoft® Visual Studio® 2010 directly support MPI debugging. This enables programmers to use their familiar programming and debugging tools efficiently in the Windows development environment.

#### 2.1.1.2 SOA-based model

An SOA-based model can package the calculation logic to service and provide clearly defined interface protocols. The protocols do not need to be aware of the implementation of low-level function calls, serialization, and message passing. Therefore, service development is separated from the code of the non-business logic supporting system; programmers can concentrate on the real business problems. In the first 10 years of the 21st century, the use of SOA has been widely adopted as a best practice in software architecture.

HPC Server supports the service-oriented programming model based on Windows Communication Foundation (WCF), with the general SOA programming model based on .NET Framework. Visual Studio includes WCF project templates, so development, utilization, configuration, deployment, and debugging are greatly simplified.

HPC Server can help programmers to extend the benefits of SOA to clusters. Programmers can package business logic to a service, then deploy the service on clusters and call the services in a parallel way. These parallel calls are distributed evenly on different compute units, nodes, cores, and slots. Therefore, applications do not need to know the physical details of the cluster; a .NET Framework program can use distributed computing resources simply and flexibly.

#### 2.1.1.3 Excel HPC Service

With billions of users, Microsoft Office Excel is widely used in all kinds of work. Excel contains a wealth of statistical analysis functions, the ability to create complex analytical models and good scalability, and is the most popular tool for data analysis. With the complexity of the model growing, computing time is greatly increased; the case that complicated models analysis on vast amounts of data takes hours, days, or even weeks is common. Many organizations hope to allow the user access to critical business information as soon as possible by shortening the running time.

The Excel 2010 and HPC Server solution is to distribute Excel calculation workloads to multiple cluster nodes and run in parallel, which reduces the processing time efficiently. Windows HPC Server 2008 R2 offers three methods: run an Excel 2010 workbook on clusters; run an Excel user-defined function (UDF) on clusters; and have Excel use services on clusters as a SOA client.

You can run multiple Excel 2010 instances on a cluster in Windows HPC Server 2008 R2. Each instance runs an independent calculation cell or iteration of embedded VBA code in the same workbook. If each instance processes the proper size of workload, the method can achieve almost linear speedup, such as running a Monte Carlo simulation algorithm workbook.

Running an Excel 2010 UDF on clusters is a new feature of Excel 2010 for complex and calculation-intensive UDF. An XLL add-in is a popular way to provide an Excel function plug-in. If a workbook contains calculation-intensive UDFs, the transfer and distribution to a cluster of the UDFs will significantly improve the performance of the workbook.

With the HPC Pack SDK and a built-in Excel project in Visual Studio Tools for Office (VSTO), an Excel application can speed up by calling SOA services deployed on multiple nodes.

### 2.1.2 System Management

Cluster resources include CPU, hard drives, memory, and high-speed network. Cluster management will face great challenges, for example, deploying clusters, monitoring resources and application status, diagnostics and troubleshooting, and the cluster utilization. HPC Server 2008 R2 provides an easy and flexible management tool to allow administrators to effectively accomplish various administrative tasks. The concurrent deployment feature of Windows Development Service (WDS) helps administrators to automatically start deployment and installation of operating system and HPC Server software. The visualized heat map feature helps administrators to easily check the status of computing resources and identify a lack of node resources. The node group feature helps administrators to categorize compute nodes and operate in groups efficiently, and it also helps users to easily submit jobs without having to remember a single head node name. Built-in cluster update helps administrators to apply a unified update policy on cluster nodes just as on all other desktops and servers in an intranet. Built-in customizable diagnostics tests help administrators to detect and troubleshoot effectively. Built-in, extensible reports help administrators to get resource utilization details and to expand the resources based on information. A PowerShell cmdlet provided by HPC Server helps administrators to write powerful scripts to automated cluster control operation. System Center Operation Manager (SCOM) Management Pack helps administrators to effectively monitor the cluster through SCOM management console, identify the exceptions, and proactively prevent system failure caused by lack of resources.

### 2.1.3 Job Scheduling System

In cluster computing, applications and users are the demand side, and computing resources are the supply side. There is a many-to-many relationship between them. Without an intermediate coordinator, the use of clusters will be disorderly and inefficient. Resource allocation will be randomized and will not be able to meet the organization's business priorities, application resource requests will produce deadlock, hardware resource failure will reduce work efficiency, and resource expansion decisions will not be supported by complete and reliable utilization records.

Job Scheduler is used for queuing jobs and other related tasks by priority, so that the priorities of an organization's calculation transactions can be met. There are multiple priority levels in Job Scheduler, ordering by priority and by a first-come, first-served principle. A seize strategy can reduce the wait time of high-priority jobs; urgent tasks can get resources in time. Job Scheduler has multiple resource allocation policies that can enable administrators to achieve the organization's goal of easily and effectively allocating resources. Jobs can use the node-, slot-, and core-level granularity to guarantee the performance of different applications. The backfill policy brings better utilization of small resource fragments. The adaptive growth and shrink policy expands and reduces the number of resources on demand, greatly improving the efficiency of resource usage. Job Scheduler can start tasks on compute nodes, and monitor the status of jobs, tasks, and compute nodes; the job will be restarted on a new node if failure occurs without user operation. Job Scheduling records the resource utilization in detail, providing history records to support resource expansion.

Job Scheduler can also be extended effectively. Administrators can customize the job acceptance policy, filtering policy, and job start policy through the filter plug-in, for example, job submission filter and the job start filter.

HPC Server features are shown in Table 2-1.

*Table 2-1 HPC Server feature list*

| **Category** | **Feature** | **User benefits** |
|---|---|---|
| SDK | MPI programming model | |
| | SOA programming model | |
| | Excel HPC service | |
| Management System | Large scale bare-metal deployment based on multicast | |
| | Support deployment of non-local hard disk machines | Image-based deployment, fully support multicast. |
| High Availability | Support redundancy of head node and broker node<br>Windows failover clustering feature and SQL Server failover clustering feature | Continuously process jobs, meet critical business needs. |
| Management System | Integrated management console | Node and job management, grouping, monitoring, diagnostics, and reporting in a single application. |
| | Network configuration wizard | Network configuration wizard helps to automatically configure network topology of cluster with more intuitive interface. |
| | MS-MPI | Use NetworkDirect driver to implement the new high-speed network stack; use shared memory for multiple core processors. Highly compatible with existing MPICH2 implementation. |
| Job Scheduling System | Job scheduler console | GUI console integrated in management console or separate operation; command |

| Category | Feature | User benefits |
|---|---|---|
| | | line supports Windows PowerShell script and traditional command line script from Windows Compute Cluster Server (Version 1 of Windows HPC); greatly improve scalability and the speed of job submission and monitoring; support SOA applications. |
| Job Scheduling System | Integrated in management console | Provide standard preset reports; extensibility feature allows use of SQL Server Analysis Service to perform additional analysis; provide historical performance charts. |
| Monitoring | Integrated in management console | Provide heat map on cluster or node group; provide charts of each node; provide performance overview within cluster. |
| Diagnostics | Integrated in management console | Built-in validation test and performance test; store, filter, and view test results and history. |

## 2.2 MS-MPI

### 2.2.1 MS-MPI Introduction

MS-MPI is a portable, flexible, vendor- and platform-independent interface for passing messages on HPC nodes. MPI is the specification, and MS-MPI, MPICH2, and others are the implementations of that standard. MPI2 is an extension of the original MPI specification.

MS-MPI is based on the open-source implementation of the MPI2 specification of Argonne National Laboratory. It is compatible with the reference MPICH2 implementation and other MPI2 implementations. MPI is the application connection specification connecting the nodes in the HPC cluster. MPI provides a portable and powerful inter-process communication mechanism that simplifies the complexities of communication between hundreds or even thousands of processors working in parallel.

MS-MPI Features:

- Allocate nodes dynamically: MS-MPI is integrated with the Job Scheduler seamlessly. Users need to specify only the total count of the resources, rather than specifying the nodes explicitly.
- Collect the resource utilization data accurately in real time: When an MPI job is running, the Job Scheduler collects the resource utilization data in real time on the allocated nodes of the job, such as CPU usage under kernel mode and user mode, and memory usage (the working set size of the job).
- Clean up the process thoroughly: If the job is terminated or an exception occurs, the MPI processes on all the nodes are cleaned up without leaving the runaway processes.
- MS-MPI in Windows HPC Pack 2008 R2 leverages the brand new NetworkDirect based on RDMA for the best performance and CPU efficiency.

- Support for runtime tracing and profiling: An important new feature of MS-MPI is the integration with Event Tracing for Windows. It uses tuning to optimize performance, and provides the time-synchronized log for debugging the MPI system and the application events on the multiple nodes working in parallel.
- Visual Studio 2010 debugger add-in: MS-MPI includes support (bindings) for the C, Fortran77, and Fortran90 programming languages. Visual Studio 2008 and Visual Studio 2010 include a parallel debugger that works with MS-MPI. Developers can launch their MPI applications on multiple compute nodes from within the Visual Studio environment, and then Visual Studio will automatically connect the processes on each node, enabling the developer to individually pause and examine program variables on each node.
- MS-MPI improves MPI significantly, providing support for shared-memory communication. The improvement benefits the widely used multi-core processors in HPC.

### 2.2.2 Windows Kernel Support for MS-MPI

Figure 2-2 shows that NetworkDirect supports the connection hardware in a more direct way, and provides a fast and efficient network solution. IHV can provide a specific, high-speed MPI driver for certain hardware, so speed and latency are as important as customization.

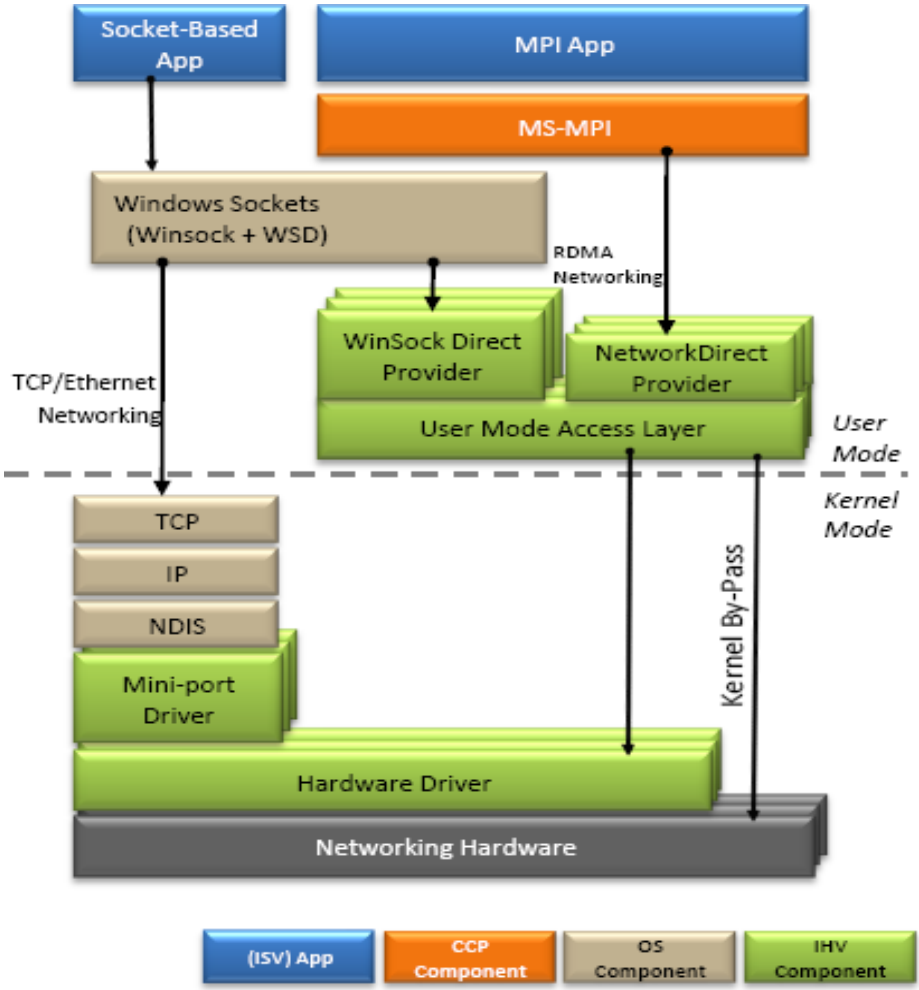



*Figure 2-2 Network architecture*

Via the NetworkDirect driver provided by IHV, MS-MPI can use any Ethernet connection that is supported by Windows Server 2008, as well as low-latency and high-bandwidth connections such as InfiniBand, Myrinet, and 10-Gigabit Ethernet. 10-Gigabit Ethernet provides a high-speed and cost-effective interconnect fabric, while InfiniBand and Myrinet are ideal for latency-sensitive and high-bandwidth applications.

For more information about MS-MPI, see Using Microsoft Message Passing Interface for the details about MS-MPI (check http://go.microsoft.com/fwlink/?LinkId=55930 on the TechNet website).

## 2.2.3 Debug, Profile, and Analyze MPI Program

Visual Studio 2010 significantly improves the user experience on parallel debugging, most features of which are provided by the Visual Studio 2008 add-in as well. The following section introduces the parallel programming, debugging, and diagnostics in Visual Studio 2008 and Visual Studio 2010.

### 2.2.3.1 Debugging

Although Visual Studio 2005 had a simple built-in debugger for MPI programs, it did not provide a full F5 experience. The new add-in for Visual Studio 2008, which is also integrated into Visual Studio 2010, allows you to select a cluster head node, specify how many cores you want, and press F5 to debug your MPI program. Figure 2-3 shows the add-in.

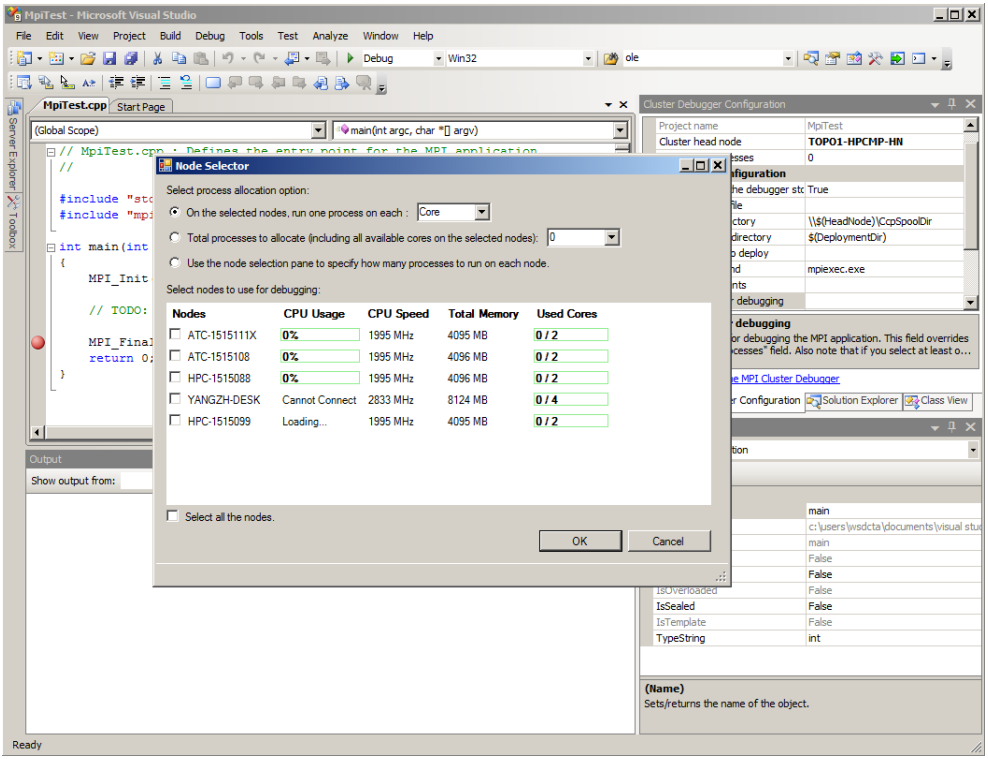



*Figure 2-3 Debug MPI Application*

In addition to the great core work that the debugger team has done, Allinea Software, a leader in parallel debugging technologies, has ported its environment to Visual Studio. Allinea's add-in enables even further streamlined MPI-specific debugging, including rank-based context switching, group-wise step, pause and run, parallel stack view, and lamination. The Allinea Software MPI debugging environment is shown in figure 2-4.

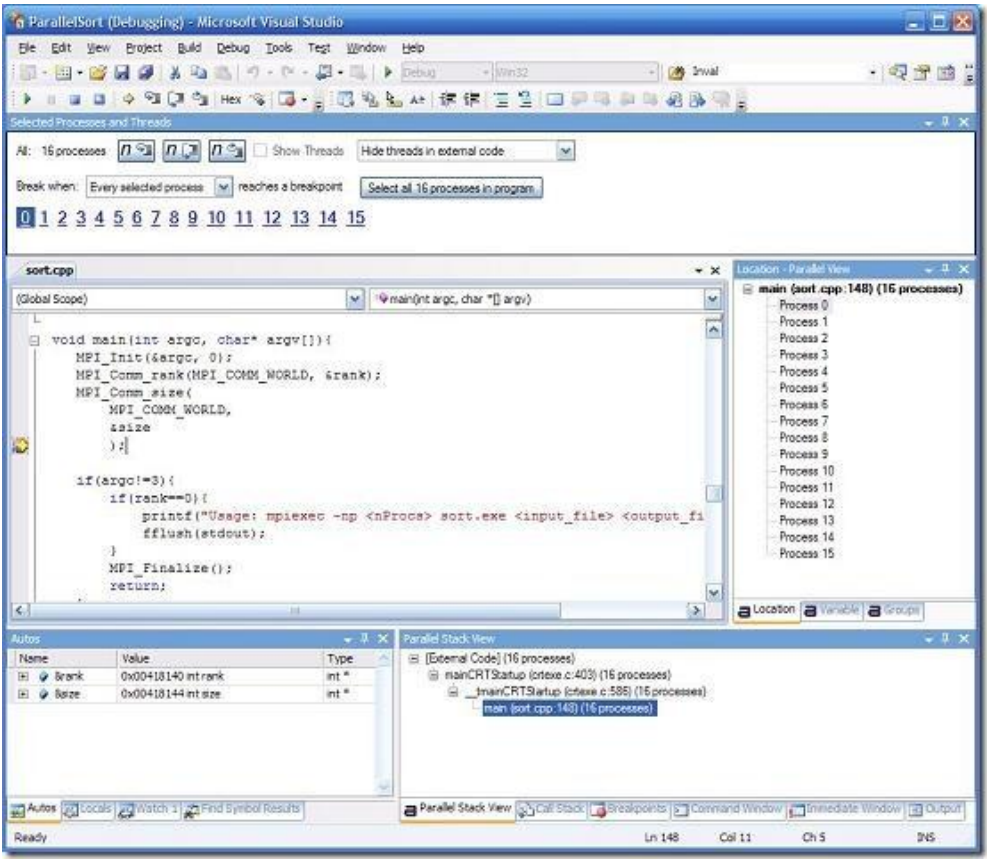



*Figure 2-4 Allinea Software debugging environment*

### 2.2.3.2 MPI Runtime Analysis

Beyond debuggers and profilers, sometimes you need specialized analysis tools to help with the complexities of large-scale parallel programs. HLRS/ZIH at Stuttgart, a leading institute in Germany, has ported Marmot, their dedicated MPI analysis tool, to Visual Studio 2008. Marmot can be used to check the validity of parameters passed to MPI calls and detect irreproducibility, deadlocks, and incorrect management of resources. Figure 2-5 shows Marmot in action.

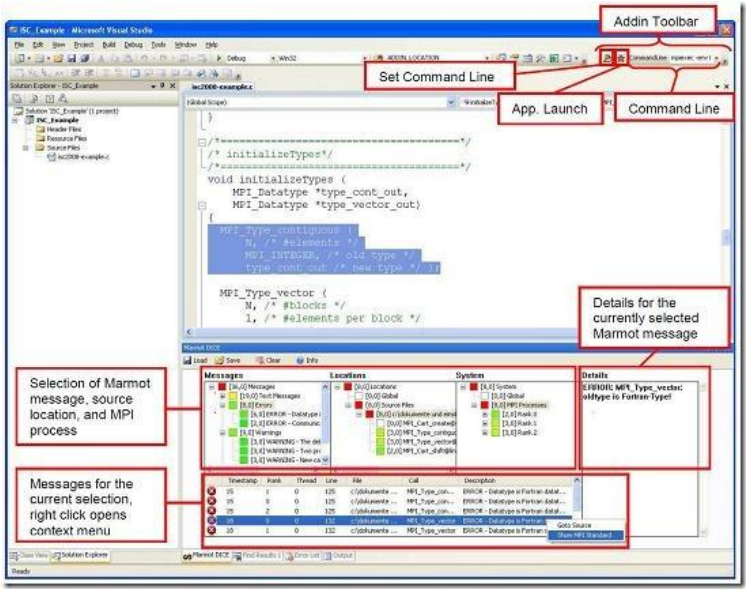



*Figure 2-5 Profile Marmot Application*

### 2.2.3.3 From Printf to Integrated Profiling and Debugging

In a world where printf-style debugging was the norm not long ago, state-of-the-art debugging and profiling tools have taken a major step forward.

From within Visual Studio, you can debug and profile native as well has high-performance MPI and Cluster SOA applications that scale from hundreds to thousands of cores. You can use XPerf and ETW to get a truly holistic view of the application in the context of the whole system. The new multi-core profiling and debugging tools that were introduced in Visual Studio 2010 can be effectively used on a cluster at the node level as well.

Visual Studio is becoming a rich and productive environment for writing parallel programs of all types. For more information about Windows HPC programming models, visit the Windows HPC Server Developer Resource Center. You can find a suite of samples that use various parallel programming models on the CodePlex Parallel Dwarfs site.

## 2.2.4 The SOA Programming Model and Runtime System

The lifecycle of an SOA application, from development to in-production, is illustrated in Figure 2-6. The developer develops server and client applications using a Visual Studio WCF project. The server administrator deploys a service onto a cluster by copying the service and registering the service. The administrator installs the client on the user's workstation. After the application is put into operation, the user runs the client to call the compute service on the cluster and to get results.

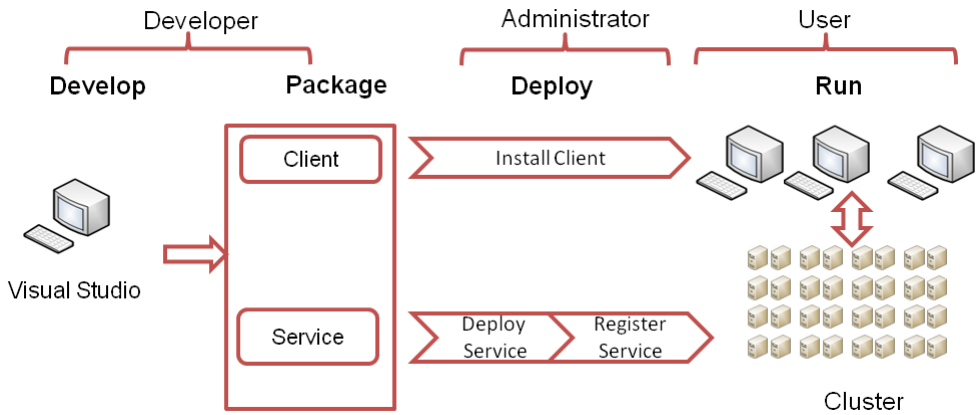



*Figure 2-6 SOA Application Lifecycle*

### 2.2.4.1 Create a Service

Code 2-1 lists a service that can be loaded by HPC Server. This service searches in a given range for numbers that are a multiple of any of the supplied divisors, and returns the sum of all these numbers.

```
[ServiceContract]
public class SumOfMultiplesService
{
    // The OperationContract attribute indicates that the method is part of
the service
    [OperationContract]
    public long SumOfMultiples(int startValue, int count, int[] divisors)
    {
        int end = startValue + count;
        long sum = 0L;

        // For every number in the indicated range
        for (int i = startValue; i < end; i++)
        {
            bool isMultiple = false;

            // Determine whether the number is a multiple of any of the
supplied divisors
            foreach (int divisor in divisors)
            {
                if (i % divisor == 0)
                {
                    isMultiple = true;
                    break;
                }
```

```
            }
            if (isMultiple)
            {
                sum += i; // Add multiples to the total sum
            }
        }
        return sum;
    }
}
```

*Code 2-1 Service*

Note that this service is a general WCF service. There is nothing that indicates that it is usable by HPC. Code 2-5 lists a client that makes a call to this service. It is used to find the sum of all multiples of 3 or 5 between 1 and 1 billion.

### 2.2.4.2 Deploy a Service

There are three strategies available for deploying a service: local deployment, central deployment, and hybrid deployment, as shown in Figure 2-7.

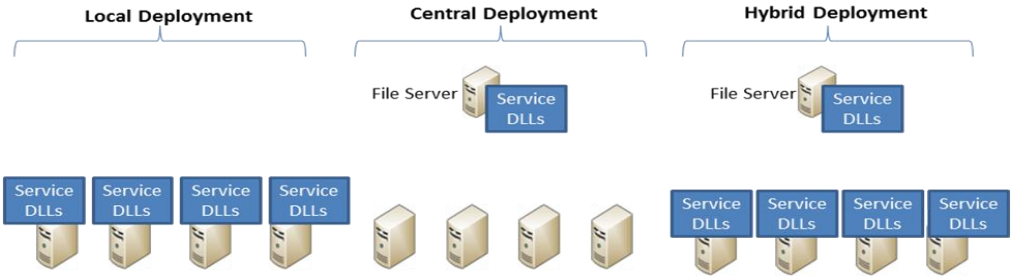



*Figure 2-7 Service Deployment Strategies*

A comparison of these three deployment strategies is presented in Table 2-2.

*Table 2-2 Comparison of Service Deployment Strategies*

| | Description | Advantage | Disadvantage | Suitable Scenario |
|---|---|---|---|---|
| Local Deploy-ment | All service DLLs are located on compute nodes. | Good service loading perfor-mance. | Difficult to update service, because it needs to update service DLLs on all nodes. | ISV-provided software package |
| Central Deploy-ment | Service DLLs are located on a file server. | Easy to update service; only need to update service DLLs on the file server. | Service perfor-mance is subject to scalability of file server | In-house developed software |
| Hybrid Deploy-ment | Frequently updated service DLLs are located on file server, and non-frequently updated service DLLs are located on compute nodes. | Good service loading performance. Easy to update service. | | In-house developed software (frequently updated) makes call to ISV-provided software package (not frequently updated) |

### 2.2.4.3 Register a service

Cluster needs to know the service deployment directory, service class, protocol, and service loading configurations. Developers should provide an XML registration file, like the one listed in Code 2-2. The administrator copies this file to the HpcServiceRegistration share folder on the head node (that is, \\HeadNodeName\\HpcServiceRegistration).

```
<?xml version="1.0" encoding="utf-8" ?>
<!— This file is located in HpcServiceRegistration share folder on the head
   node -->

<configuration>
  <microsoft.Hpc.Session.ServiceRegistration>
    <service assembly="%CCP_HOME%bin\EchoSvcLib.dll"
             contract="EchoSvcLib.IEchoSvc"
             type="EchoSvcLib.EchoSvc"
             includeExceptionDetailInFaults="true"
             maxConcurrentCalls="0"
             endpointNotFoundRetryTimeout="60000"
             maxExcludedNodesLength="3"
                 >
      <!--Below is a sample for adding environment variables to the
   service-->
      <environmentVariables>
        <add name="myname1" value="myvalue1"/>
        <add name="myname2" value="myvalue2"/>
      </environmentVariables>
    </service>
  </microsoft.Hpc.Session.ServiceRegistration>
</configuration>
```

*Code 2-2 Service XML Registration File*

#### 2.2.4.4 Create a client

The central concept of client programming model is the session, which is a handle between the application and the service. Steps to call an HPC SOA service are as following:

1. Create a session and obtain a service endpoint reference.
2. Create a service proxy using the service endpoint reference. HPC Server 2008 R2 supports two different kinds of proxies: the WCF proxy and the HPC service proxy. The WCF proxy is generated using svcutil or Visual Studio WCF tools, while the HPC service proxy is provided by HPC Pack SDK.
3. Send requests and receive responses using the service proxy.
4. A session has four optional settings, as shown in Table 2-3.

*Table 2-3 Four optional settings for session*

| Settings | Description | Advantage |
|---|---|---|
| Durable | Messages in the session are persisted onto disk. Only the HPC service proxy is supported. | • Reliable message transfer.<br>• Support WCF Broker Node fail-over. |
| Non-Durable | Messages in the session are kept in memory. Both the WCF proxy and the HPC service proxy are supported. | Low latency, quick response time. |
| Shared | Session is shared by multiple clients. | • Quick start-up.<br>• Services use the same security context.<br>• Different clients may share the same data. |
| Non-Shared | Session is exclusively owned by one client. | • Guaranteed client application resource supply and SLA.<br>• Isolated process, good security. |

The WCF proxy is used to send compute requests via asynchronous method invocation and receive processing results via callback, as shown in Code 2-3.

```
// Event used to synchronize the main and the callback threads
Static AutoResetEvent done = new AutoResetEvent(false)
// Counter that used to
Static int count = 0;

SessionStartInfo info = new SessionStartInfo("shpc-0110", "EchoService");
using (Session session = Session.CreateSession(info))
{
     NetTcpBinding binding = new NetTcpBinding(SecurityMode.Transport,
   false);
     using (Service1Client client = new Service1Client(binding,
   session.EndpointReference);
     {

         using (Service1Client client = new Service1Client(binding,
   session.EndpointReference);
         {
          for (i = 0; i < 10000; i++)
              client.BeginEcho("hello world!", callback, new MyState(client,
   i));
```

```
            }
            done.WaitOne();
        }
}

class MyState { Service1Client client; int i; }
Void callback(IAsyncResult result)
{
        lock (Monitor) {
            MyState s = result.AsyncState as MyState;
            string reply = s.client.EndEcho(result);
            Console.WriteLine("Received response for request {0}: {1}", s.i,
   reply);
            if (Interlocked.Increment(count) == 10000)
                done.Set();
        }
}
```

*Code 2-3 Client program that uses WCF Proxy*

The HPC service proxy is used to send compute requests via **SendRequest()** and receive results via **GetResponses()**, as shown in Code 2-4.

```
SessionStartInfo info = new SessionStartInfo("shpc-0110", "EchoService");


// Create a durable session

// Request and response messages in a durable session are persisted so that

// in event of failure, no requests nor responses will be lost.  Another
authorized

// client can attached to a session with the same session Id and retrieve
responses

using (DurableSession session = DurableSession.CreateSession(info))

{

    Console.WriteLine("done session id = {0}", session.Id);

    NetTcpBinding binding = new NetTcpBinding(SecurityMode.Transport);


    // Create a BrokerClient proxy

    // This proxy is able to map One-Way, Duplex message exchange patterns

    // with the Request / Reply Services.  As such, the client program can
send the

    // requests, exit and re-attach to the session to retrieve responses (see
the

    // FireNRecollect project for details

    using (BrokerClient<IService1> client = new
BrokerClient<IService1>(session, binding))

    {
```

```
        for (int i = 0; i < numRequests; i++)
        {
            // EchoRequest are created as you add Service Reference
            // EchoService to the project
            EchoRequest request = new EchoRequest("hello world!");
            client.SendRequest<EchoRequest>(request, i);
         }

         // Flush the message.  After this call, the runtime system
         // starts processing the request messages.  If this call is not
called,
         // the system will not process the requests.  The
client.GetResponses() will return
         // with an empty collection
         client.EndRequests();

         // GetResponses from the runtime system
         // EchoResponse class is created as you add Service Reference
"EchoService"
         // to the project
         foreach (var response in client.GetResponses<EchoResponse>())
         {
             try
             {
                 string reply = response.Result.EchoResult;
                 Console.WriteLine("\tReceived response for request {0}:
{1}", response.GetUserData<int>(), reply);
              }
              catch (Exception ex)
              {
                  Console.WriteLine("Error occured while processing {0}-th
request: {1}", response.GetUserData<int>(), ex.Message);
               }
            }
        }
        session.Close();
   }
}
```

*Code 2-4 Client program that uses HPC service proxy*

### 2.2.4.5 Client Program Example

Code 2-5 lists an HPC SOA client application example.

```
    class Program
    {
        static int totalSearchRange = 1000000000; // Search up to 1 billion
        static int rangePerCall = 10000000; // Search in 10 million sized
blocks
        static int[] divisors = new int[] { 3, 5 }; // For multiples of 3 or 5

        static int AsyncResultCount = 0;
        static AutoResetEvent AsyncResultsDone = new AutoResetEvent(false);

        static long total = 0L;

        static void Main(string[] args)
        {
            // Request to reserve at least 8 and up to 64 cores for this job
            SessionStartInfo startInfo = new SessionStartInfo(
                "MyHeadNodeServer", "SumOfMultiplesService");
            startInfo.ResourceUnitType = JobUnitType.Core;
            startInfo.MinimumUnits = 8;
            startInfo.MaximumUnits = 64;

            // Create a session and use the endpoint address it provides
            // to make calls to the WCF service.
            // The SumOfMultiplesServiceClient is an automatically generated
client proxy
            // created by Visual Studio when adding a reference to your WCF
service.
            using (Session session = Session.CreateSession(startInfo))
            using (SumOfMultiplesServiceClient client = new
SumOfMultiplesServiceClient(
                new NetTcpBinding(SecurityMode.Transport, false),
                session.EndpointReference))
            {
                client.InnerChannel.OperationTimeout = new TimeSpan(1, 0, 0);

                // For searching to 1 billion with a range of 10 million per
request,
                // 100 reqests will be made
                AsyncResultCount = totalSearchRange / rangePerCall;
```

```
                for (int i = 1; i < totalSearchRange; i+=rangePerCall)
                {
                    // Make all of the requests asynchronously, providing a
completion callback
                    client.BeginSumOfMultiples(
                        i,
                        rangePerCall,
                        divisors,
                        RequestCompleteCallback,
                        new RequestState(client));
                }

                // This thread will wait until all of the calls have been
processed
                // and returned by the cluster
                AsyncResultsDone.WaitOne();
            }

            Console.WriteLine("Total sum of multiples: {0}", total);
            Console.ReadLine();
        }

        // Called whenever the cluster completes one of the requests
        static void RequestCompleteCallback(IAsyncResult result)
        {
            // Add the method's result to the running total
            RequestState state = result.AsyncState as RequestState;
            Interlocked.Add(ref total, state.GetResult(result));

            if (Interlocked.Decrement(ref AsyncResultCount) <= 0)
            {
                // When all the requests have been completed,
                // indicate completion to the main thread
                AsyncResultsDone.Set();
            }
        }

        // Encapsulates information about a service request
        // to be passed to the completion callback
        class RequestState
```

```
        {
            SumOfMultiplesServiceClient client;

            public RequestState(SumOfMultiplesServiceClient client)
            {
                this.client = client;
            }

            public long GetResult(IAsyncResult result)
            {
                return client.EndSumOfMultiples(result);
            }
        }
    }
```

*Code 2-5 HPC SOA Client Application*

This client program creates a session on the cluster, obtains a service endpoint reference, and then sends 100 asynchronous requests to the service. Behind the session are multiple service instances involved in the computation. Cluster dispatches these requests to service instances of the session. On completion of each request, the callback function, which aggregates results and notifies the main thread of all requests done, gets executed at the client. Note that this WCF client program uses a standard .NET Framework asynchronous call; its only difference to a normal client is that the service endpoint reference is from an HPC session.

Leveraging the HPC SOA model, you can separate out the compute-intensive part from an application, convert it into a service, and run it on a cluster. In the preceding example, an important detail is the granularity of each request—we split the 1 billion search range into 100 10 million subranges. Compared to MPI, SOA is suitable for dispatching compute requests of medium-granularity (>50ms). The difference to a parametric-sweep job is is that the service instance doesn't exit after processing each compute request. This improves efficiency, which is especially true for service instances that require a long initialization time. The HPC SOA programming model is very convenient for building applications. This model is service oriented; thus, developers could build service programs and client programs on the widely used WCF platform. Visual Studio provides a set of easy-to-use WCF service template and service reference tools. With these tools, developers could quickly prototype their applications, debug them, and unit-test them.

#### 2.2.4.6 SOA Debugging

In Visual Studio 2010, a cluster SOA plug-in provides the SOA setting tab, in which you can select the head node, debug a node or service, and deploy a runtime library and clean it up automatically. Let's take a look at the Visual Studio 2010 SOA debugger in Figure 2-8.

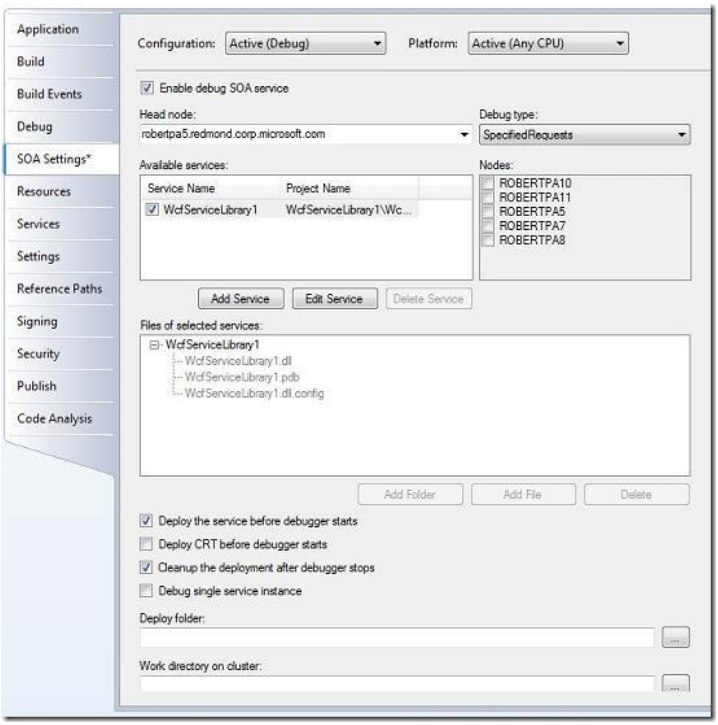



*Figure 2-8 VS2010 SOA Debugger*

## 2.3 Job Scheduler System

Job scheduler supports both command line and graphic interfaces. The GUI-based job management module is integrated into the management console. The Windows PowerShell-based command line interface provides more flexibility and makes the system easier to use. Through the Common Object Model (COM) interface, system administrators can write VBScript and other scripts to submit and monitor a job. Windows HPC Pack 2008 R2 is backward compatible with all command line scripts in Windows Compute Cluster Server, and is able to interoperate with third-party scheduler programs to achieve the highest compatibility to the existing environment. The command line interface supports many types of languages, including Perl, Fortran, C/C++, C#, and Java.

With Windows HPC Pack 2008 R2 Job Scheduler, users can schedule jobs, allocate the necessary resources to the jobs, and update properties and tasks associated with the jobs. Job Scheduler has built-in parametric-sweep support. It also provides the ability to apply filter plug-ins to submitted jobs. In addition, it supports heterogeneous scheduling and multi-core cluster.

Job can include a single task or multiple tasks. The number of processes needed by each job can be specified, together with whether the CPU is exclusively used or can be shared with other jobs or tasks.

**Product design story: Why do we have two concepts, Job and Task, in our scheduler?**

Most existing scheduler systems only contain the Job concept, in which a command line is specified. The job in Microsoft HPC Server, however, splits the traditional "job" concept into two parts. The job in Microsoft scheduler is used to define the resource requirement. It can be looked at as a package of tasks, while each task defines the command line, environment variables, I/O redirection, and working directory of the computation. By splitting in this way, we are able to represent all types of user scenarios using job uniformly. For example, sequential or multithreading applications are represented as one task per job. An MPI job contains only one task, which can include multiple processes running on different nodes. Parametric-sweep can be represented as multiple tasks within a single job. Workflow can be defined as multiple tasks, too, but with dependencies between tasks.

The traditional job concept came into existence in the 1980s. At that time, the mainstream HPC job was simply a big program that generally took a long time to complete. Therefore, one job containing one command line seemed pretty straightforward. Starting in the mid-1990s, chip manufacturing and life science companies started to increase their usage of clusters. Their jobs were mainly parametric-sweep or workflow. Using the traditional job scheduler, a lot of jobs had to be submitted for a single run. This overloaded the scheduler, because it had to allocate resource for each job. On the other hand, it made it difficult for users to monitor those jobs, because a user would have to monitor thousands of jobs at the same time. However, things are much easier with the Microsoft scheduler, which enables the user to pack those thousands of tasks into a single job, which requires only one time resource allocation. Nodes can be reused. Users can track the current progress of their tasks by monitoring a single job's status.Moreover, the detail of an individual task can be queried from its parent job. In this way, HPC Server is a significant enhancement over traditional batch system in terms of both scheduler performance and user experience.

The scheduler in HPC Server R2 contains a powerful job and task management module, which includes retrying a failed job or task, identifying unreachable nodes, and automatic deletion of the completed jobs. Every job is running under the user's security context, which means that the job and task own only the access rights granted to the user who submitted them. All functionalities in the scheduler can be invoked from the command line interface.

A job's lifetime contains three phases:

1. Job submission: The user submits the job to the system. The scheduler system decides whether or not to accept the submitted job according to the policies specified by the system administrator as well as the current cluster resource availability.
2. Job scheduling: Following the scheduling policies, which were also set by the system administrator, the scheduler sorts the jobs and allocates the necessary resources to them.
3. Job execution: As long as the minimum required resource for a job is satisfied, the job will be launched on the allocated nodes. The job's tasks will then be started on appropriate nodes according to their own individual resource requirements.

There are more granulized job states corresponding to each of the above phases. Figure 2-9 shows the state machine of a job within the scheduler.

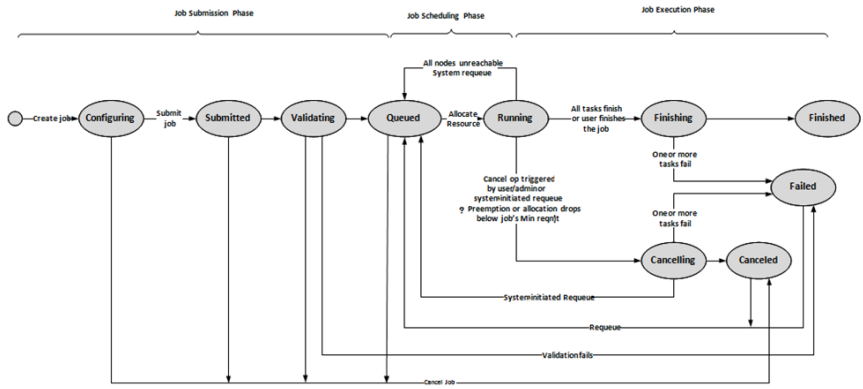



*Figure 2-9 State transitions of a job*

When a job is created, its state is Configuring. In this state, the user can add tasks to the job. After that, the user can submit the job. The job will then be switched to the Submitted state. After a job is submitted, the scheduler will validate whether or not the cluster's resource can satisfy the job's requirement, and whether or not the latter satisfies the resource allocation policies defined by the system administrator. Then the scheduler will insert the job into the queue according to its priority and submit time. At this point, the job's state will be Queued.

The scheduler system will then allocate resources to the Queued jobs. After the requirements get fulfilled, the scheduler will start the job on allocated nodes. The job state will change to Running. The tasks within a job will then be launched on those nodes following a certain order. After all tasks are completed, the job state will change to Finishing, and then finally change to the Finished state.

### 2.3.1 Job submission phase

In the job submission phase, the user can create and submit a job using various interfaces. The HPC Server R2 supports .NET Framework, WS Basic Profile, COM, SOA, command line, PowerShell, and job management GUI. There are also many third-party ISVs who have integrated their system with Microsoft's Job Scheduler; users can submit jobs through their applications as well.

**Product design story: Why there are so many interfaces?**

This is mainly determined by HPC applications. HPC applications developed in earlier times have different requirements for programming language and interface. COM interface can support traditional C/C++ applications and VBScript scripts. The command line interface supports the batch scripts written by system administrators. The .NET Framework supports the applications programmed on the .NET Framework platform. Job management GUI provides a more intuitive view for those users who are not quite familiar with the command line. WS Basic Profile supports non-Windows clients, such as Linux clients, to submit jobs to Windows HPC Server.

The user submits the job to the system. The job will then be started with the credentials provided by the user. Job priority can be specified while submitting the job. By default, all the jobs will be submitted with the priority Normal. The default priority can also be specified in the job template. If a job has to be expedited, the administrator can assign a higher priority to the job. Also, different job templates can define different job priorities. The administrator can then grant them to different users.

A job consumes resources (nodes, processor cycles, and time). These resources can be reserved for a specific job. People normally think the best way to execute their jobs is to reserve as much (or as powerful) resources as possible within the cluster. However this is not the truth. If a job is specified as needing a very powerful resource set, most likely it has to wait for a longer time for those resources to be spared. On the other hand, a job with less resource requirements or execution time specified will probably be scheduled more quickly and can finish earlier, especially when it meets the criteria of the backfill time section (the bubble in the resource schedule in which the resource stays spared).

Another factor that affects the execution time is job exclusiveness. By default, a job is not using the nodes exclusively, unless otherwise specified by the user or the job template. In this case, the user can expect the job to be scheduled more quickly, since it can share the same set of resources with other jobs. Once any part of those sharable resources is spared, the job can get scheduled.

### 2.3.2 Job scheduling phase

There is only one queue in the job scheduler. A job's position within the queue is determined by its priority and submission time. If they have the same priority and submission time, the jobs will be scheduled following the first-come, first-served (FCFS) order.

**Product design story: Why there is only one queue?**

Job scheduler software has evolved for three generations.

In the first generation of exclusively used supercomputers (the 1970s and 1980s), an institution was normally had only one Cray computer. Each computer had a queue system, such as Portable Batch System (PBS).

At the beginning of the 1990s, clusters with RISC chips, such as IBM SP, replaced the supercomputers. Job schedulers at that time were network based, with multiple queues, such as Load Sharing Facility (LSF). In most cases, the number of queues is less than the number of nodes. A multiqueue system satisfies the requirement of resources being shared by more than one user. There are different sorting algorithms for queues. The main problem in a multiqueue system is that it’s difficult for the administrator to identify which job will be executed next.

Microsoft Job Scheduler is the third generation scheduler. It has only one queue. The problem of multiple users sharing the resource is handled by multiple job templates. Job templates divide the cluster into different partitions, and grants them to different user groups with different job priorities. Through the preprocessing by job templates, the job-sorting algorithm can be simplified as just an FCFS policy. In this way, while the resource sharing requirement is satisfied, the user and administrator are also able to identify the next job quickly.

Job scheduling determines the position of a job in the queue and allocates resources to it. Both functions are controlled by the scheduling policies. HPC Server Pack R2 supports the following nine policies, with each corresponding to a specific scheduling problem.

#### 2.3.2.1 Priority-based FCFS

This scheduling policy combines the FCFS and job priority together. The jobs are positioned by their priority and submission time. Under this order, all jobs within the queue follow the FCFS principle. Therefore, all jobs with Highest priority will be scheduled earlier than any of the job with AboveNormal priority, and so on.

#### 2.3.2.2 Balanced scheduling

Under the FCFS policy, after a job is submitted, if there is no available node at the moment, the job will have to remain in the queue until the other job finishes. In this case, the bounding time of that job is affected. By using the balanced scheduling policy, the administrator can optimize the bounding time for jobs. When necessary, the balanced scheduling policy will squeeze the number of resources being used by running jobs to spare more resources and allocate them to queued jobs, so that their bounding time will be shorten. Balanced scheduling and FCFS policy are mutually exclusive. Only one of them can be enabled at any moment.

#### 2.3.2.3 Backfilling

The user specifies the necessary time and resources to execute the job. If the existing available resources cannot completely fulfill the job's requirement, the scheduler will reserve those resources for the job. During this reserved period, if the scheduler detects some of the reserved resources become available for a certain amount of time, it can then choose a job that has a shorter execution time to fill this section. This allows short jobs to jump ahead in the queue while not affecting other jobs' resource usage. In this way, the global cluster utilization will be improved.

#### 2.3.2.4 Exclusive

When a job is set to exclusive, no other job, or task from another job, can be running on the same node with it.

#### 2.3.2.5 Resource matching

Under this policy, the user has to specify the resource requirements of the application while submitting the job. The scheduler system provides several types of filters on nodes, including node group, memory size, and number of cores. The user can specify node group as the filtering criteria. If there is more than one node group specified, their intersection will be used by the scheduler. By specifying the minimal CPU core number and memory usage, the scheduler can make sure that the allocated nodes will satisfy the application's requirements. Among all the nodes that satisfy the application's requirement, the user can further specify sorting criteria, such as the node with minimal cores first or the node with minimal memory first. In this way, resources can be used efficiently and unnecessary reservations can be avoided.

#### 2.3.2.6 Job template

The system administrator can create job templates to define resource requirements for a specific application as well as to differentiate priority among multiple user groups. In the narrowest definition, a job template is the pattern for creating jobs. The job template is defined by the administrator, and is used when the user submits a job. Job templates normally set parameters according to the characteristics of the application. In this way, users are relieved from studying complex terms for job scheduling and can be more focused on their application.

In a broader definition, a job template efficiently combines the user requirements together with the resource allocation policy from their institution. Using job templates, the administrator can much more easily pursue the goal of production efficiency by simply adjusting the granularity of resource allocation among different user groups.

All job properties are optional. Once a job is submitted, the scheduler will assign the default value to the properties that are not specified. The cluster administrator can specify the default property value for all jobs using this template. For example, a site requires all jobs to share nodes, or tries to enforce the runtime or valid project name for all jobs. In the scheduler of Windows Compute Cluster Server 2003 (the first version for Windows HPC Server), the administrator has to use a submission filter to do this. That required writing a plug-in program to analyze the XML formatted job resource requirements, then updating necessary properties according to the site policy. Now, with job templates, the cluster administrator can enable the same function without writing any plug-in code under most situations.

#### 2.3.2.7 Multilevel Compute Resource Allocation (MCRA)

Using the MCRA scheduling policy, the job can specify the granularity of computational resources used in scheduling, so that the resources can be arranged more efficiently according to the application traits.

- Core-based allocation: This is suitable for applications that consume large amounts of CPU resource. More processors are provided to themto improve performance.
- Socket-based allocation: This is suitable for applications whose performance is limited by memory access. For those applications, the throughput to retrieve data from memory is the most important factor for their performance. Therefore, more tasks being executed over the same memory bus will not speed up the application, since they are competing on the same path to the system memory.
- Node-based allocation: This is most suitable for applications that rely on the node's I/O resources. Applications that rely heavily on disk or network access can be classified as this type. For those applications, more tasks running on the same node would not speed up any of them, since they are waiting for the same disk or network channel.

#### 2.3.2.8 Dynamic Grow/Shrink

Under this policy, resources allocated to a job with multiple tasks can be automatically adjusted along with its execution. When more available resources present, they will be added to the job (Grow). If the job is detected to have fewer tasks to be started comparing to the currently allocated resource, some resources will be subtracted from the job (Shrink). This improves the global efficiency of the cluster, and helps to make sure that resources are always first allocated to the job with the highest priority.

#### 2.3.2.9 Preemption

With preemption, the jobs with higher priority can preempt resources from the jobs with lower priorityto shorten the bound time for the former. HPC Server 2008 R2 job scheduler provides two types of preemption:

- Immediate preemption: It will cancel the lower-priority job to yield resources to higher-priority ones.
- Graceful preemption: Shrink the job with lower priority to get resources for higher-priority jobs. Graceful preemption will generally take longer than immediate preemption; however, it will ensure that none of the running task is abnormally terminated, so that the existing work will not be lost.

### 2.3.3 Default resource allocation

The default resource allocation is to allocate resources according to the requested number of processors, requested nodes or node groups, exclusiveness and preemptability, and so on.

The job scheduler will sort the nodes according to the memory and speed. It first sorts the nodes from the largest memory to the smallest memory; it then sorts the nodes with the same memory according to their speed, from fast to slow. This is the default handling policy for resource allocation. It can be overridden by other sorting criteria defined with the job. After the sorting, the scheduling algorithm will go through the sorted nodes and allocate CPUs, to fulfill the jobs' minimum and maximum requirement to processors. The scheduler will first try to satisfy the maximum number of CPUs requested by the job; then it will shift to the next job.

If Requested Nodes is specified in the job, instead of sorting all nodes, the scheduler will sort the nodes within the list specified in requested nodes, till the minimum or maximum processor number is reached.

To balance the requested and allocated resources for each job and maximize the cluster utilization, we need to configure the jobs properly. The cluster efficiency can be optimized by leveraging the combination of backfill windows, self-adapting allocation (grow/shrink), and default job template. To achieve the best utilization of resources, users should create, submit, and run their jobs complying with the following rules:

1. Do not use the default Infinite runtime while submitting the jobs. Instead, pick a runtime that should be the longest estimation for that job.
2. Specify the requested nodes only when the job needs special resources on those nodes.
3. While setting the number of processors for a job, put the ideal value to be the maximum, then put the least acceptable value to be the minimum
4. Reserve an appropriate number of nodes for each task. If the task needs only one node but is reserved two nodes, then the task has to wait for at least two nodes to become available to start.
5. Set a job or task to exclusive only when necessary.

Operating the jobs following the above rules will help maximize the cluster utilization.

### 2.3.4 Job execution phase

Executing a job is to start the tasks on compute nodes. Job will be running in the security context of its submitter, so that transparency and security are ensured to network resource access (such as file or database server). Jobs can also be automatically requeued when there is a shortage of computational resources (or some of them have crashed).

When the job is running, the scheduler will run its tasks one by one on the allocated nodes. It will also monitor the tasks' current status and record their resource usage. The task state transition is represented in Figure 2-10.

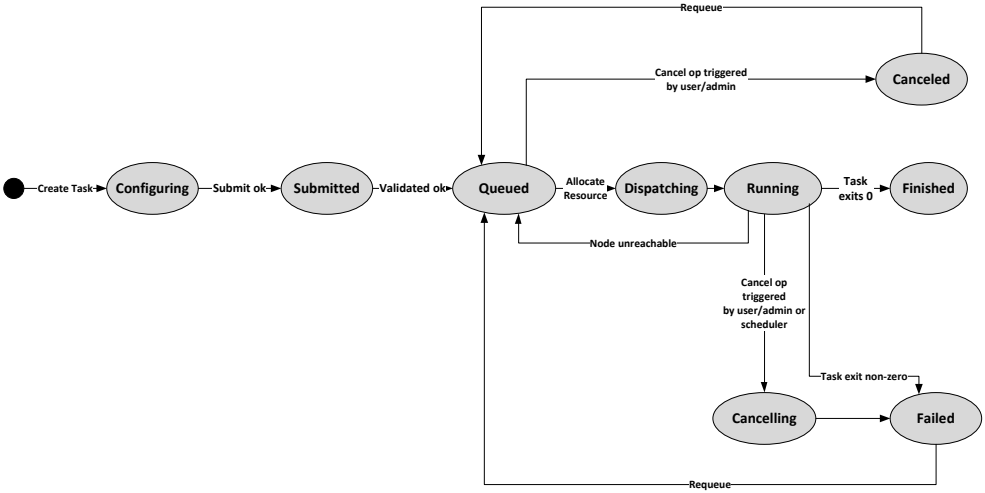



*Figure 2-10 Task state transition*

## 2.3.5 User job manager GUI

The user job manager GUI in Windows HPC Server 2008 R2 is shown as Figure 2-11. In this GUI, the user can group the jobs according to their state, and view and operate the jobs by group. The main part of job management GUI is job and task list view. Picking up a job, its tasks will be shown below the job list view. User can view the detailed properties of any task and the job's history event logs, which includes the times of submission, running, and terminating that job.

The job management GUI is a convenient operating platform. The user can create jobs of different types, including single-task job and parametric-sweep job. The user can also import a job from an XML file. By selecting one or more jobs, the user can perform different types of actions, including view, update, add task, copy, cancel, requeue, or export the job to an XML file.

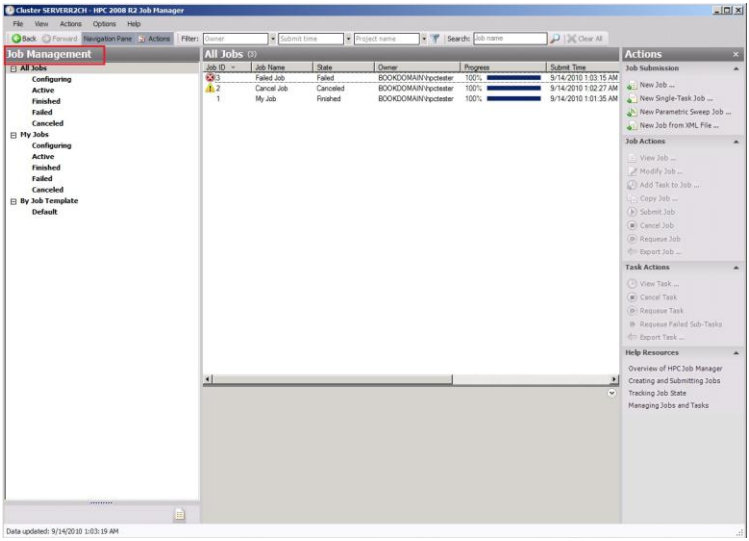



*Figure 2-11 Job management GUI*

### 2.3.5.1 Creating a job

Click **New Job...** in the menu of the job management console GUI. The **New Job** window shown in Figure 2-12 is displayed. In the first step of the wizard, fill in the job details (such as name, project, and priority), job run options (such as run time) and job resource requirements (such as minimum and maximum number of cores).

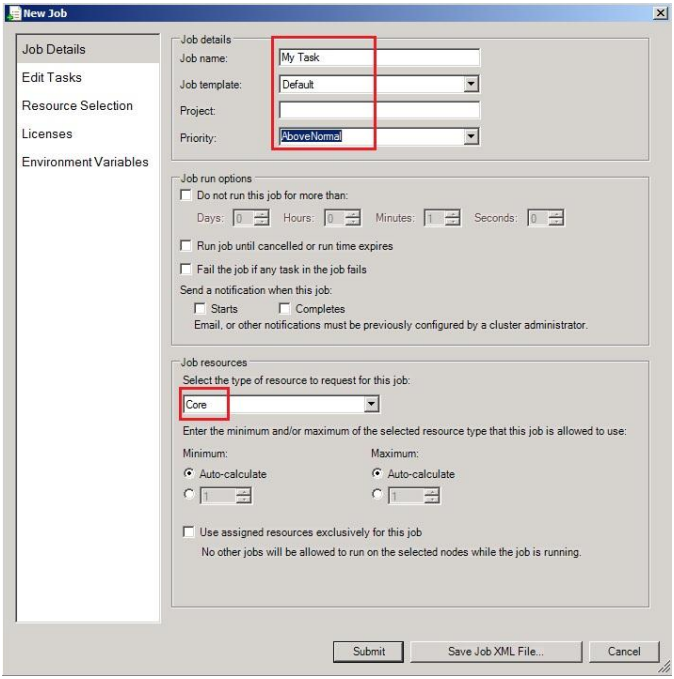



*Figure 2-12 New job wizard*

### 2.3.5.2 Adding tasks to a job

In the second step of the wizard, "Edit Tasks", click **Add**, and then fill in the command line, working directory, standard in/out/error redirection, and resource specification for that task, as shown in Figure 2-13.

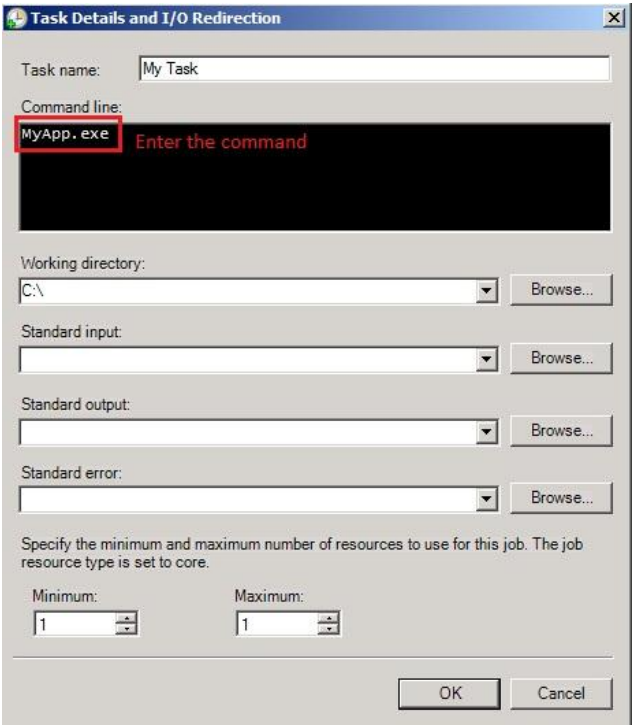



*Figure 2-13 Adding tasks to a job*

**Hint: What's the best way for tasks to access data?**

The way that tasks access data can affect their performance. The correct way to access data should be determined by the size of data files and the frequency with which you update them.

If the data is comparatively large and updated less frequently, the data should be put to the local compute nodes. If the data file is smaller, it can be put to the shared folder. If the data files are large and frequently updated, they should be uploaded to the compute nodes before the job is started.

For medium and smaller data files, put them into a shared folder and set the

working directory to that folder. As the tasks are started, applications from all compute nodes can access all the files in that folder.

#### 2.3.5.3 MPI application command line

If the task uses MPI, the input command line format would be:

```
mpiexec [mpi_options] <Myapp.exe> [arguments]
```

In this command line, Myapp.exe is the executable of the application.

### 2.3.5.4 Parametric-sweep tasks

The user can go to the **New Job** wizard, click the drop-down list of the **Add** button, and choose **Parametric Sweep Task...** (shown in Figure 2-14), and get **Parametric Sweep Task** dialog (shown in Figure 2-15).

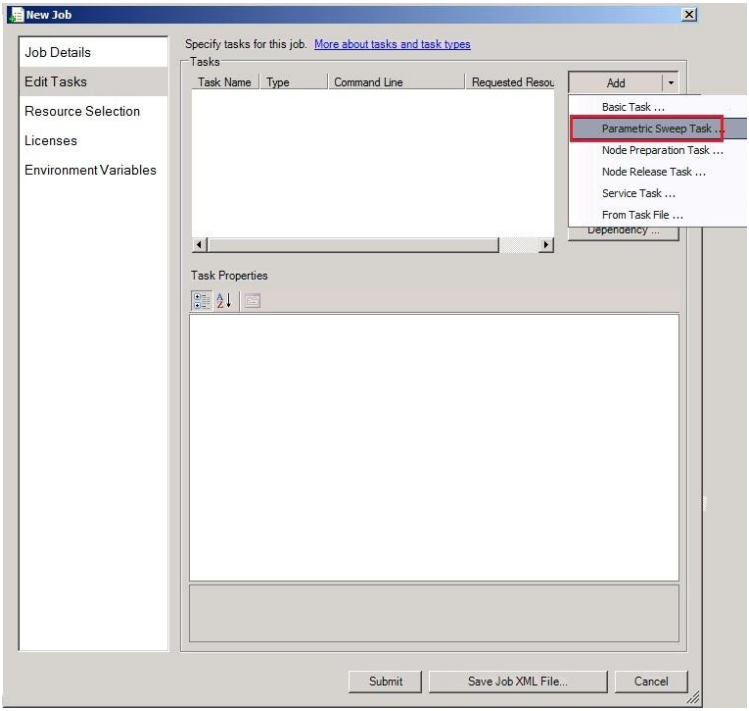



*Figure 2-14 Adding parametric-sweep task*

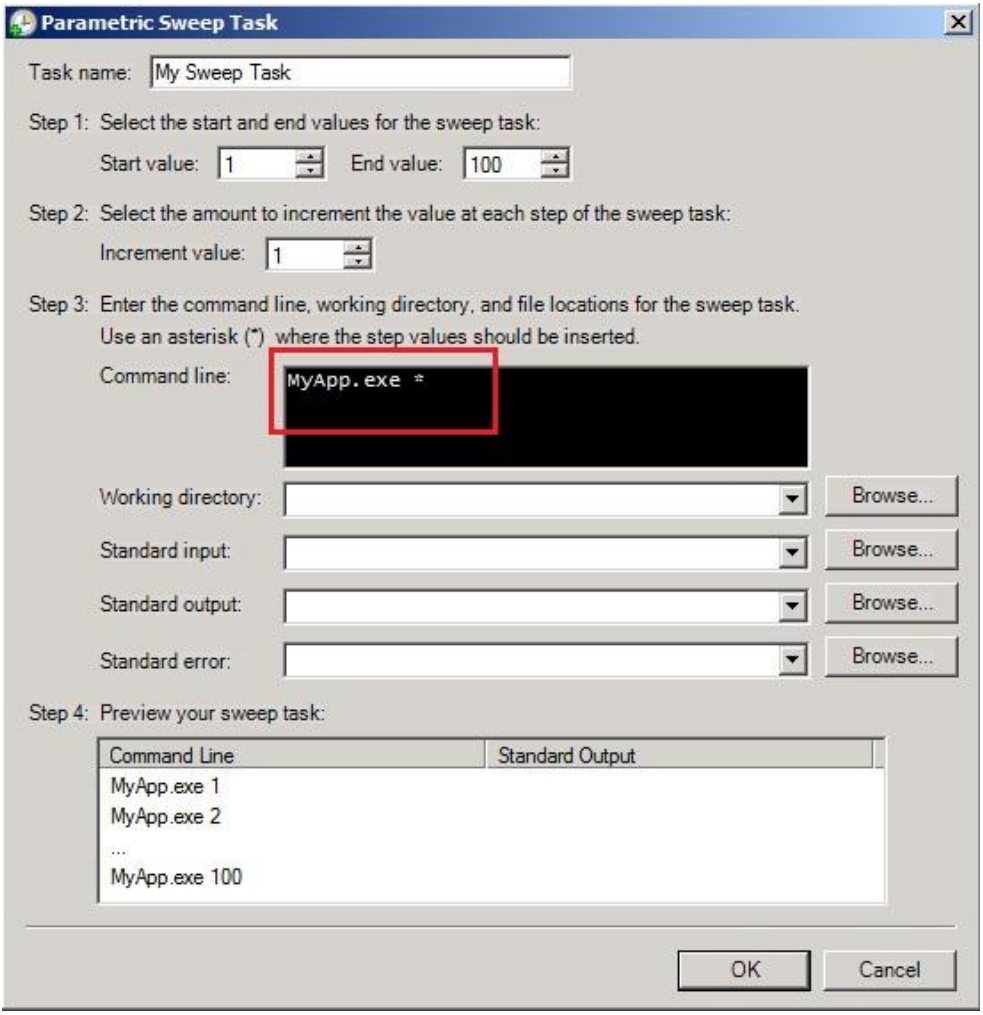



*Figure 2-15 Parametric Sweep Task dialog*

The user can specify the start and end value of the parametric sweep to easily define a task that can contain hundreds of thousands of subtasks. The user can use the wildcard (*) character in a prototype command line, such as "MyApp.exe *". It will be translated to subtasks with command lines "MyApp.exe 1", "MyApp.exe 2", and so on during the execution time. In this way, both the job submission time and the memory consumed by the scheduler can be largely saved.

## 2.4 Cluster Manager

Cluster Manager is the window for HPC, as shown in Figure 2-16. The console looks like Microsoft Outlook® at first glance. The goal of designing this console is to make it the Outlook for cluster administrators. The bottom left part of the console is the wunderbars, and the upper left part shows the categories that can be managed under each wunderbar. The list view and heat map are in the middle part of the console, where all objects in the selected categories are displayed.

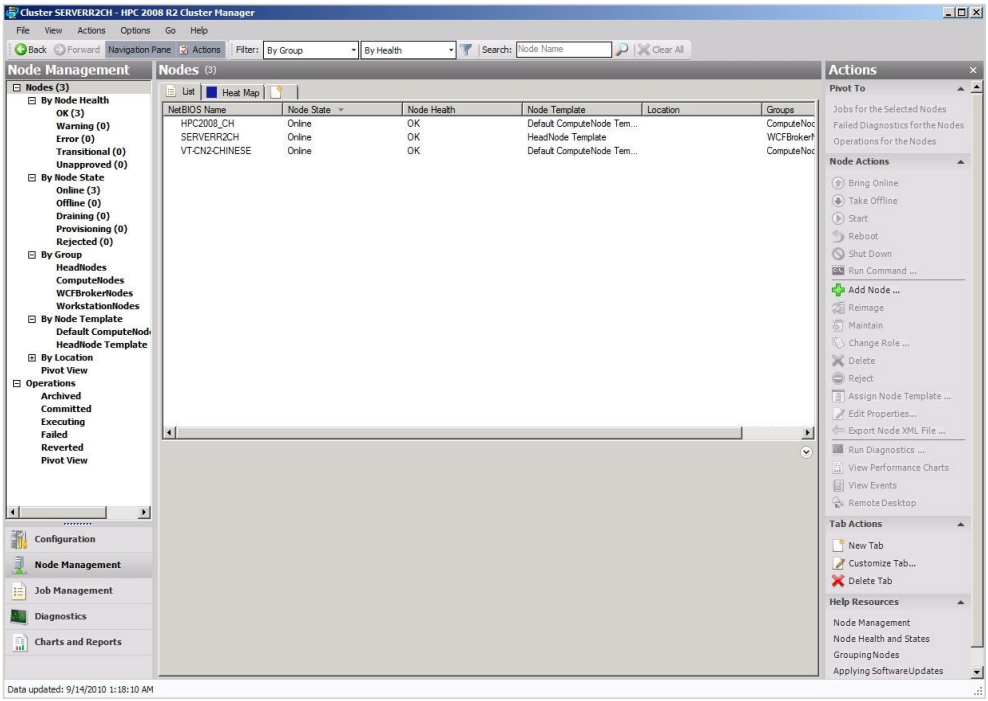



*Figure 2-16 Cluster Manager Overview*

**Product Design Highlights:** Conventional MMC Plug-in to MOM UI

In 2004, according to the common practices in the Windows Server Division, the HPC team decided to implement the management UI as an MMC plug-in. After two years of practice, we found that it has many limitations. The main problem is that MMC is designed for single computer system management, and it has only two levels of information: operations and details. However, three levels of information are needed to manage a cluster that contains multiple machines: node, operations, and details. Therefore, at the beginning of the development of Windows HPC Pack 2008 R2 in 2007, we decided to use the MOM management UI framework.

The new management console contains five major navigation panes:

- Configurations: This navigation pane contains the To-do List, which is used to set cluster configurations. It includes the network configuration wizard and the create node template wizard. The network configuration wizard simplifies the configuration of networks, and the create node template wizard guides the administrators in configuring node templates to maintain consistent compute node deployment, including updates and application deployment.
- Node Management: This navigation pane is used to start operations that apply to nodes, such as deploying nodes, quickly monitoring node states, bringing nodes online or taking nodes offline, and adding or deleting nodes.
- Job Management: This navigation pane is used to submit jobs, monitor job states, and apply operations to jobs.
- Diagnostics: This navigation pane allows administrators to select one node or a group of nodes to run diagnostic tests to verify the major cluster functionalities or configurations, such as network connections, job execution, configurations, performance, and so on. Administrators can check the test progress and historical test run results to troubleshoot general cluster issues.
- Charts and Reports: This navigation pane shows the standard reports for jobs and nodes in the cluster. It also provides job scheduler reports and on-demand reports.

### 2.4.1 Configurations

Click the "Configurations" wunderbar to get a window like Figure 2-17. The Configurations navigation pane is the user interface to configure clusters. There are required tasks and optional tasks in configurations. The items in the to-do list described in Chapter 1 are required tasks, including network configuration, providing installation credentials, configuring node naming series, and creating a node template. The to-do list also contains optional tasks, such as adding an operation system image, adding or removing users, adding nodes, adding drivers, and so on. Other configurations like node management, job management, and diagnostics configurations are optional as well.

The idea of the user interface design for configurations is trying to make the items self-explanatory so that administrators do not need to look up documents whenever they want to configure something. For administrators' convenience to look up documents, the links to the documents are provided on the UI.

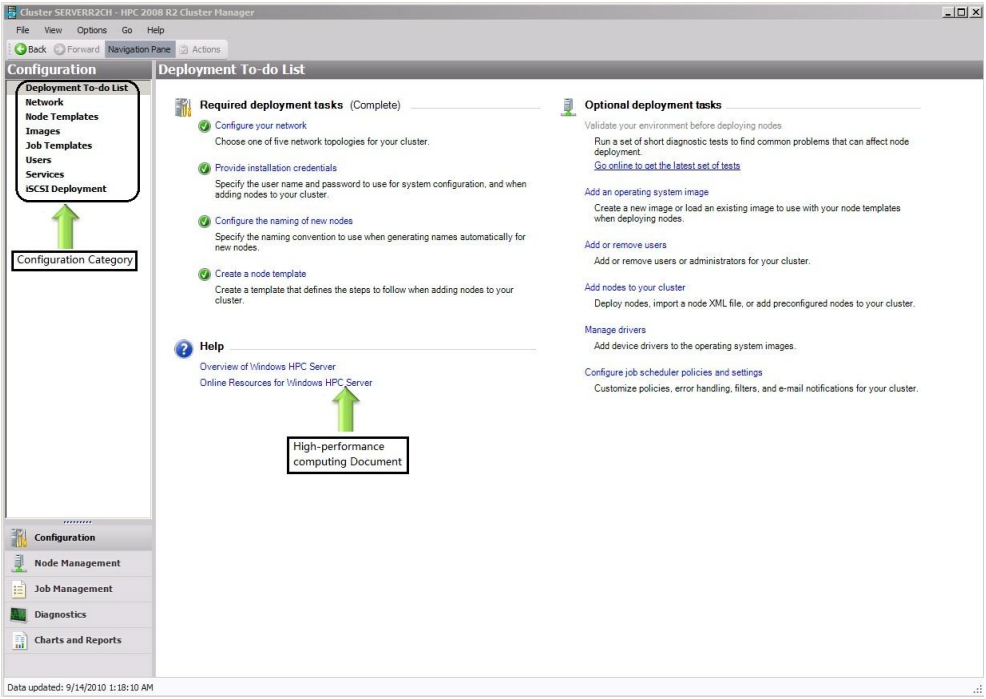



*Figure 2-17 Configurations View*

**Product Design Highlights:** How to Improve Feature Discoverability?

The cluster management and configuration user interface is very complex. As more and more functionality is added, a lot of features may be hidden if the product does not have good discoverability. Microsoft Office has a great number of features, but surveys indicate that users use only 10% of the features. Office 2007 introduced the Ribbon UI, which greatly improves feature discoverability. The first reaction of many users is, "Oh, there are so many formats and it is so easy to use!"

When designing cluster management UI, feature discoverability is also a major design metric. In the overview page of the Configurations navigation pane, the idea is to list the main configuration items along with links and concise descriptions. Thus, administrators do not have to open the configuration categories in the left pane one by one for common configuration tasks.

The items in the left part of the Configurations navigation pane are shown as Table 2-4.

*Table 2-4 Configurations Navigation Pane Items*

| **Configuration items** | **Description** |
|---|---|
| Deployment To-do List | The required steps to create a cluster, including cluster network configuration, providing installation credentials, configuring compute node naming series, and creating compute node. template. |
| Network | Cluster network configurations, including network topology, IP address allocation policy, and firewall configurations. |
| Node Templates | Node templates are used to define the required tasks to add, configure, and maintain cluster nodes. Node templates can define steps to install and configure operation system images, deploy and configure HPC Pack, install device drivers and applications, install software updates, and restore nodes back to a certain configuration. |
| Images | Windows Deployment Service uses Windows Imaging Format (WIM) files and multicast to rapidly deploy nodes in a parallel manner. Administrators can create node templates for the type of nodes to deploy by using the create node template wizard. A node template contains the base operating system image, drivers, configurations, and additional software. |
| Job Templates | Job templates define the default job resources and allocation limitations. By using job templates, administrators can effectively limit the job types in the cluster, and provide default values to assist users that are not familiar with the terminology of the HPC job scheduling system. |
| Users | User and administrator configurations. Administrators can add or remove cluster users and administrators. |
| Services | WCF service configurations in the cluster. Administrators can list the services and configure the service properties and run-time trace levels. |

## 2.4.2 Node Management

Click the "Node Management" wunderbar to get the window shown in Figure 2-18. This part of the UI categorizes nodes by their properties, and provides list view and heat map view in each category along with signs for failures and exceptions. For selected nodes, administrators can perform operations such as bring online, take offline, reboot, shutdown, run commands, and so on. Administrators can also view the health states, properties, network states, performance metrics, and operation logs for nodes. They can pivot to the jobs, diagnostic tests, and operations for the selected nodes as well. For long-running operations such as deployment, administrators can check the progress and operation logs to track the operation results.

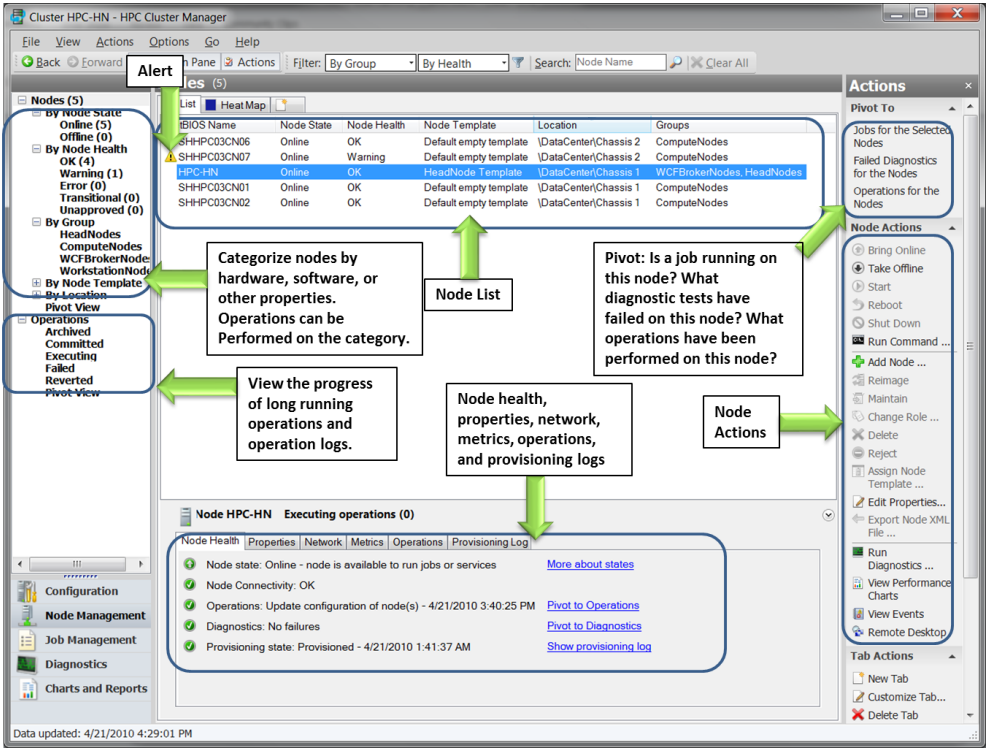



*Figure 2-18 Node Management View*

### 2.4.2.1 Node Groups

As the name indicates, a node group is a collection of one or more nodes. There are several built-in node groups like HeadNodes, ComputeNodes, and WCFBrokerNodes. Different node groups can contain the same set of nodes.

For the convenience of job scheduling, administrators can create custom node groups. Usually, users have specific requirements for applications, hardware, and network devices when submitting jobs. If some jobs need application X, at least 32 GB of memory, and the InfiniBand network, the administrator can create three node groups and name them as Application X, Large Memory (which means at least 32 GB of memory), and InfiniBand. The command line, GUI, and API for submitting R2 jobs all support filtering nodes by node groups to ensure that jobs can be run on the most suitable nodes.

### 2.4.2.2 Node Locations

Sometimes administrators need to organize nodes by their physical locations. For example, an administrator wants to check the performance counters for nodes in a certain rack to understand their work load. The management system in R2 provides the support for node locations, and administrators can define three levels of node locations. For example, the first level is rack, and the next level is the nodes in the rack, as shown in Figure 2-19.

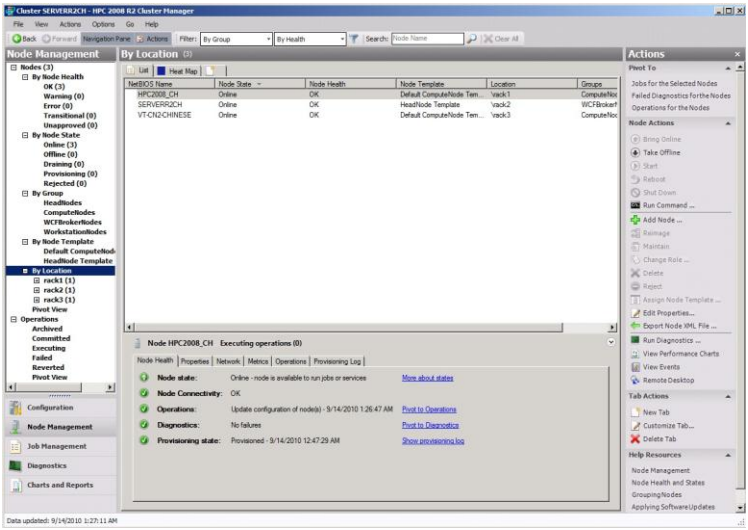



*Figure 2-19 Node Location List*

### 2.4.2.3 Node States

The node life cycle is shown in Figure 2-20. When the head node finds a node on the network, it puts the node in the Unknown state. After the administrator confirms the node and assigns a node template to it, the node changes to the Provisioning state. After the deployment succeeds, it then changes to the Offline state. By performing bringing online operation on the node, it changes to the Starting state and then becomes Online afterwards. In Offline state, no user jobs can be run on the node. Only administrators can run commands or diagnostic tests on it.

If the deployment fails, the node returns to the Unknown state. In Offline state, administrators can reassign a node template to perform maintenance operations (for more information, see 2.4.2.6 Node Operations), and the node then changes to the Provisioning state.

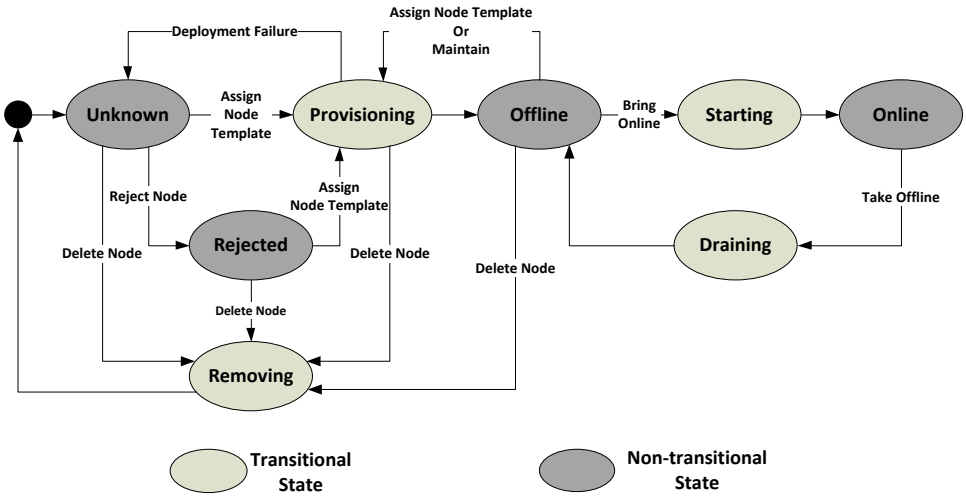



*Figure 2-20 Node State Transition*

When administrators want to troubleshoot or perform maintain operations such as installing application or software updates on the production nodes, they must first take these nodes offline. To avoid impact on the jobs running on these nodes, before the nodes change to the Offline state, the management system must ensure that all the running jobs on these nodes have finished (or the nodes are in the Draining state). Offline nodes will not accept new jobs.

Before acknowledging the identity of some Unknown nodes, which means whether the nodes should be accepted as a new member of the cluster, administrators can choose to reject them. These rejected nodes will then change to the Rejected state. After the administrators confirm that these nodes should be new members of the cluster, they can add these nodes to the cluster by assigning a node template to them.

Administrators can delete nodes in the Rejected or Offline state. These nodes will go into the Removing state and then be removed from the cluster.

#### 2.4.2.4 Node Health

Node health indicates whether there is a warning or an error on the node. The node health is associated with the icon beside the node item in the node list. When a node's health is OK, there is no icon beside the node item.

Node icons, alert types, and related health are shown as table 2-5.

*Table 2-5 Node Icons*

| **Icon** | **Alert type** | **Related health（for more information, see Table 2-6）** |
|---|---|---|
| | Error | **Unreachable**<br>**Deployment Failure** |
| | Warning | **Diagnostic Failure** |
| | Pending Operation | **On-going Operations** |

Table 2-6 lists the node health.

*Table 2-6 Node Health*

| Node Health | Description |
|---|---|
| **OK** | No errors or warnings are detected on this node. |
| **Warning** | Issues at Warning level happened on this node. |
| **Error** | Issues at Error level happened on this node. |
| **Transitional** | The node state is one of the transitional states, including Provisioning, Removing, Starting, and Draining. |
| **Unapproved** | The node has been detected by the management system, but has not been approved as a cluster member by administrators. |

When a node is selected, the node health details are displayed below the node list, as shown in Figure 2-21.

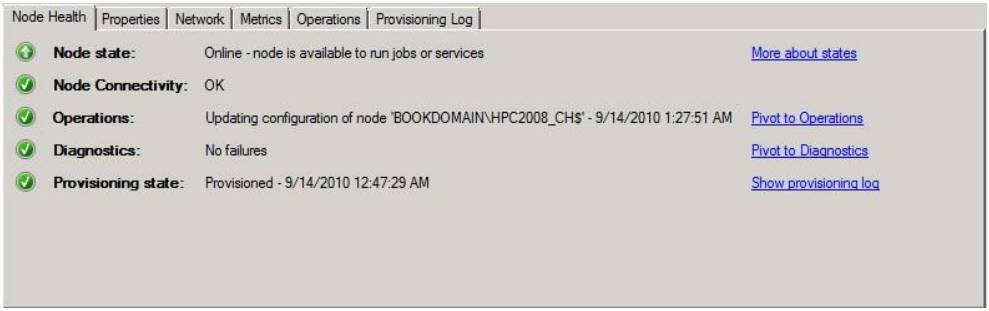



*Figure 2-21 Node Health Details*

This user interface provides information about node state, network connectivity, latest operation, diagnostic test, and the time and result of the node deployment. It also provides links for administrators to check the details.

### 2.4.2.5 Node Image Deployment

One of the biggest challenges that users will meet when adopting HPC solutions is the deployment of the cluster and nodes. It is not hard to deploy four nodes. Sometimes you just need to insert a DVD on each node, press Enter, and then add nodes into the cluster. The configuration is easy as well, and it does not matter how much time is needed. However, when deploying a cluster that contains hundreds or even thousands of nodes, it is extremely dull and error-prone.

Windows Compute Cluster Server 2003, the previous version of Windows HPC Server 2008, uses Remote Installation Services (RIS) to deploy compute nodes automatically. Compared to manually adding one node at a time, this greatly improves the efficiency. With the release of Windows Compute Cluster Server 2003 SP1, it can now leverage Windows Deployment Services. Windows Deployment Services has many advantages when deploying larger or heterogeneous clusters. In Windows HPC Pack 2008 R2, Microsoft continues to use Windows Deployment Services to deploy compute nodes. The deployment feature has been integrated into the management console, where administrators can choose the nodes to deploy and monitor the deployment progress easily.

Windows HPC Pack 2008 R2 contains a create node template wizard, which is used to provide guidance for administrators to create node templates when configuring compute nodes. The template editor also contains other advanced features, such as configuring templates for automatic application deployment. The create node template wizard in Windows HPC Pack 2008 R2 supports adding drivers into images.

Windows Deployment Services uses WIM files and multicast to quickly deploy nodes in a paralleled manner. Administrators can create node templates for the type of nodes that need to be deployed by using the create node template wizard. Node templates contain the operating system image, device drivers, configurations, and additional software.

The Windows Deployment Services running on the head node will detect the compute nodes that need to be deployed when they are powered on. Nodes can be added to the cluster by importing the node list. Administrators can also interactively select detected nodes when they are displayed in the management console. The deployment process does not involve interaction with people, during which the system applies the image, configures the nodes, and deploys applications according to the node template.

Before HPC Server 2008 R2, compute nodes had to have local hard disks to support deployment operations. HPC Server 2008 R2 supports diskless boot, which means the computer can boot from the storage array through the iSCSI support from the operating system.

Deployment is a complex operation, and many factors can affect the results of the node deployment. HPC Server 2008 R2 provides Deployment Environment Validator diagnostic tests to help the administrator verify the deployment environment, as shown in Figure 2-22.

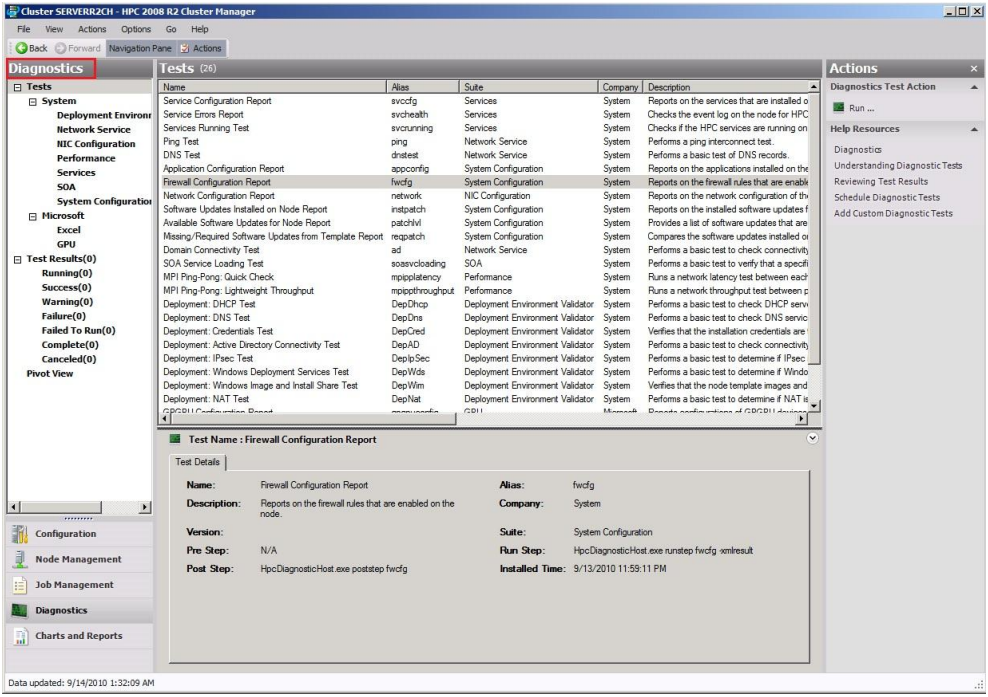



*Figure 2-22 Deployment Environment Validator Tests*

**Product Design Highlights**: Why Deployment Environment Validator Tests?

Before HPC Server 2008 R2 was released, many users encountered different deployment issues. The crux of the problem is the environment configuration of the head node. Although the product team has written detailed documentation, a lot of customers do not read them before performing operations. No design that relies on documentation to avoid operator errors from users can achieve its goals. By collecting and analyzing failure cases from customers, the product team designed eight built-in tests to verify the infrastructure that deployment depends on, including tests for DHCP, DNS, installation credentials, active directory connectivity, IPSec, TCP/IP configuration, private network, and Windows Deployment Services. Like the Deployment To-do List, the Deployment Environment Validator is another feature that is designed in HPC Pack to reduce the possibility of errors. Having simplified steps and reduced operator errors as design goals is an important reason why HPC Pack can help popularize high-performance computing.

*Figure 2-23 shows the test result for Windows Deployment Services.*

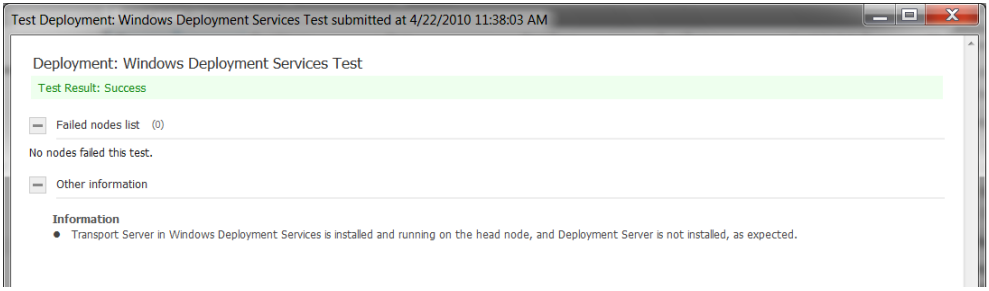



*Figure 2-23 Deployment Environment Validator/Windows Deployment Services Test Result*

The greatest benefits of using Windows Deployment Services to execute the deployment of Windows HPC Pack 2008 R2 are the following:

1. Extensibility offered by image multicast.
2. Integration in the management console directly.
3. Support for node template.
4. Driver integration (and update).
5. Support for heterogeneous compute nodes deployment.

Windows HPC Pack 2008 R2 solves and simplifies the operations to build and manage HPC clusters, and to deploy and update nodes. Providing a template for each type of deployment ensures that users can deploy thousands of nodes easily.

#### 2.4.2.6 Node Operations

When administrators want to perform operations on some nodes, they must first select these nodes and then click the operation in the right action pane or right-click the mouse to choose the desired operation. The common node operations that the management system of HPC Pack R2 supports are listed as Table 2-7.

*Table 2-7 Common Node Operations*

| Operation | Description |
|---|---|
| Deploy nodes | Administrators can deploy operation system and applications on bare-metal machines or add preconfigured nodes. (Preconfigured nodes are those that have HPC Pack 2008 installed and have been connected to the cluster according to the network topology defined by administrators.) |
| Start | Start nodes. |

| Operation | Description |
|---|---|
| Bring nodes online | |
| Take nodes offline | |
| Reboot | Reboot nodes. |
| Shutdown | Shutdown nodes. |
| Reject | Reject nodes. |
| Create node groups | |
| Edit node properties | Node properties include node descriptions, product keys, and node location. |
| Change node roles | Change the role of nodes, including head node, WCF broker node, and compute node. |
| Run commands | Administrators can run commands on the selected nodes. |
| Install applications | Administrators can use the post installation tasks in node template or use clusrun commands to install applications on the selected nodes. |
| Update nodes | Administrators can use the apply updates task in node template to update the nodes. |
| Reimage | When some errors that are hard to troubleshoot or cannot be fixed occur, administrators can reimage the nodes. |

### 2.4.2.7 Operation Logs

Operation logs are the records of the recent operations for the cluster. They can help administrators to track the cluster configuration update and monitor the progress of operations. Administrators can view the operation logs for the entire cluster or a certain set of nodes. They can also pivot to the associated diagnostic tests. The operation logs are shown in Figure 2-24.

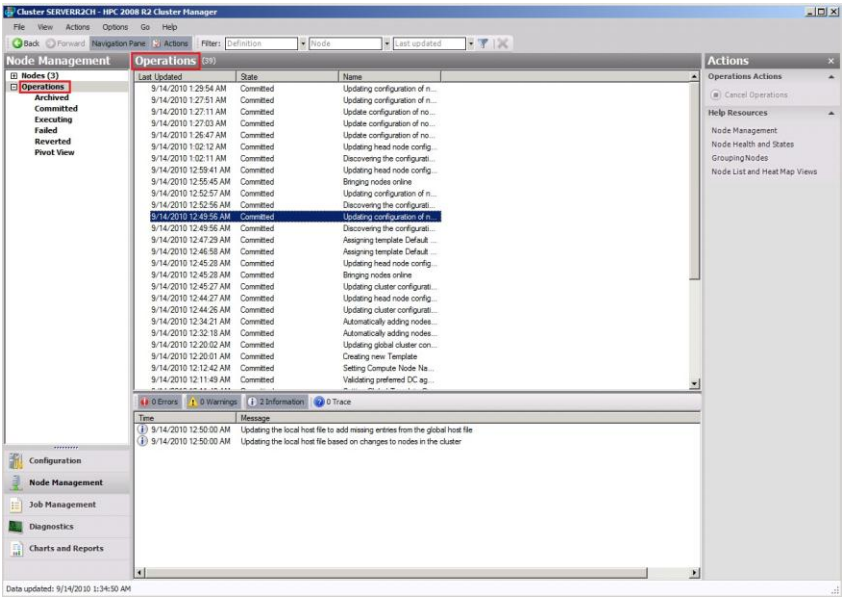



*Figure 2-24 Operation Logs*

The operation states are listed as Table 2-8.

*Table 2-8 Operation States*

| Operation state | Description |
|---|---|
| **Archived** | Operation has completed for over 24 hours, or the diagnostic test results have been cleared. When an operation is in the Archived state, it is removed from other status reports. |
| **Committed** | Operation succeeded. |
| **Executing** | Operation is executing. |
| **Failed** | Operation failed. The system has been reverted or failed to revert. |
| **Reverted** | System has been reverted to the previous state when the operation failed or was canceled. The following operations are irreversible:<br>• Assign node template<br>• Reimage<br>• Maintain<br>• Start<br>• Shutdown<br>• Reboot |

#### 2.4.2.8 Nodes Heat Map

The heat map view in R2 shown in figure 2-25 is part of the cluster management console. It shows the real-time cluster state. In a heat map, each node is represented as a square, which can simultaneously display three performance metrics configured by the administrator. Each performance metric is assigned with a color range to display different values for the counter, for example, from light red to dark red. The darker the color is, the load that the metric represents is higher. For example, the color becomes darker if there is less free memory, the CPU usage is higher, or there is more network traffic.

The R2 management UI supports two kinds of heat map: stacking heat map and overlaying heat map. A stacking heat map, as shown in Figure 2-25, can display three performance metrics at the same time. An overlaying heat map, as shown in Figure 2-26, can display only one performance metric that is related to the load and the configured priorities of performance metrics. For example, an administrator defines CPU usage as the first priority, and memory paging as the second. When a node has plenty of free memory, the overlay heat map displays the CPU usage metric. When there is little free memory and the system memory paging performance counter value is high, the node is shown as a memory paging metric in the heat map. Hence, the administrator can find the node with the extreme load among hundreds of nodes as quickly as possible, and then is able to perform deep monitoring to take early actions.

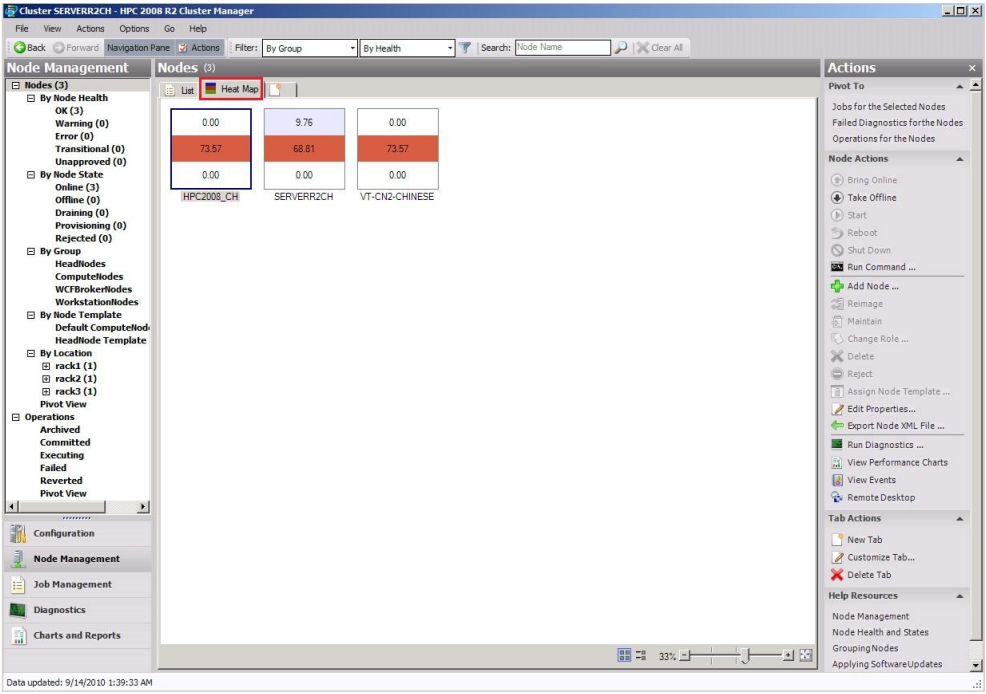



*Figure 2-25 Stacking Heat Map*

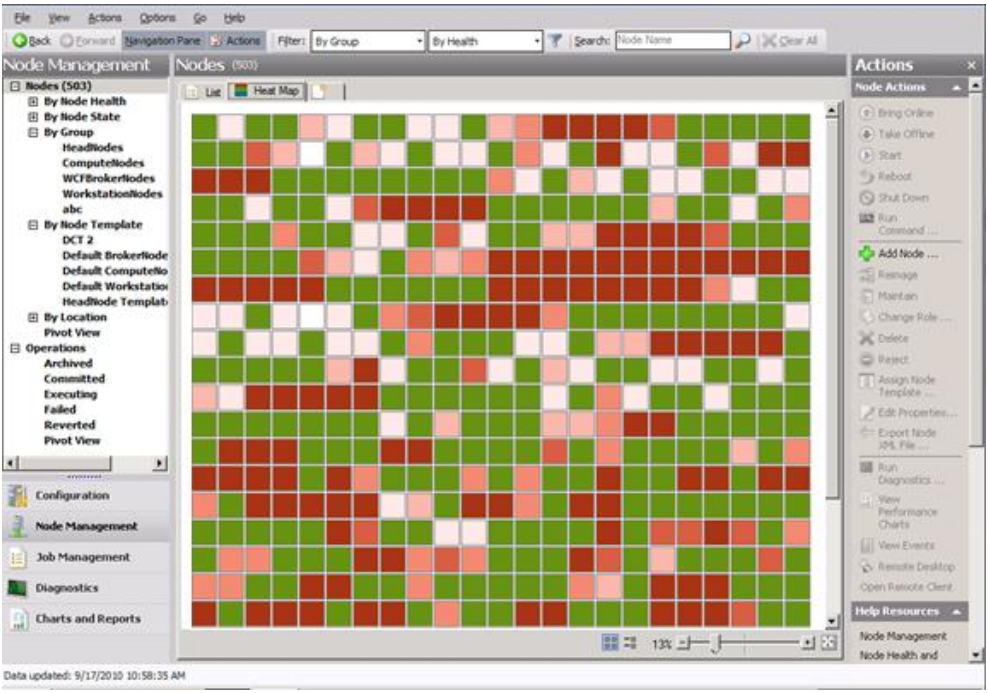



*Figure 2-26 Overlaying Heat Map: Green Nodes with Higher Memory Paging Values*

## 2.4.3 Job Management

Click the "Job Management" wunderbar to enter the job management UI. This UI is the same one as the job management console introduced in other chapters. We will not discuss it again in this section.

## 2.4.4 Diagnostic Test Tools and Framework

Administrators used to have to write and manage their own troubleshooting script files. With R2, administrators now have a series of common diagnostic tests and an extensible diagnostic framework.

### 2.4.4.1 Built-in Diagnostic Tests

System administrators spend a lot of time on diagnostics. The common methods and steps of diagnostics include verifying cluster deployment results or configuration changes, looking for causes of failures, and measuring whether there is performance degradation over time. To help administrators verify whether the cluster is working properly and to provide methods for them to run tests, store results, and view results, R2 offers built-in diagnostic test reports. This feature significantly improves the efficiency for administrators to verify deployment, troubleshoot, and determine the reason for performance degradation.

Click the Diagnostics wunderbar to enter the diagnostic test UI shown in Figure 2-27. This UI enables administrators to view the diagnostic test list, run tests, and check results.

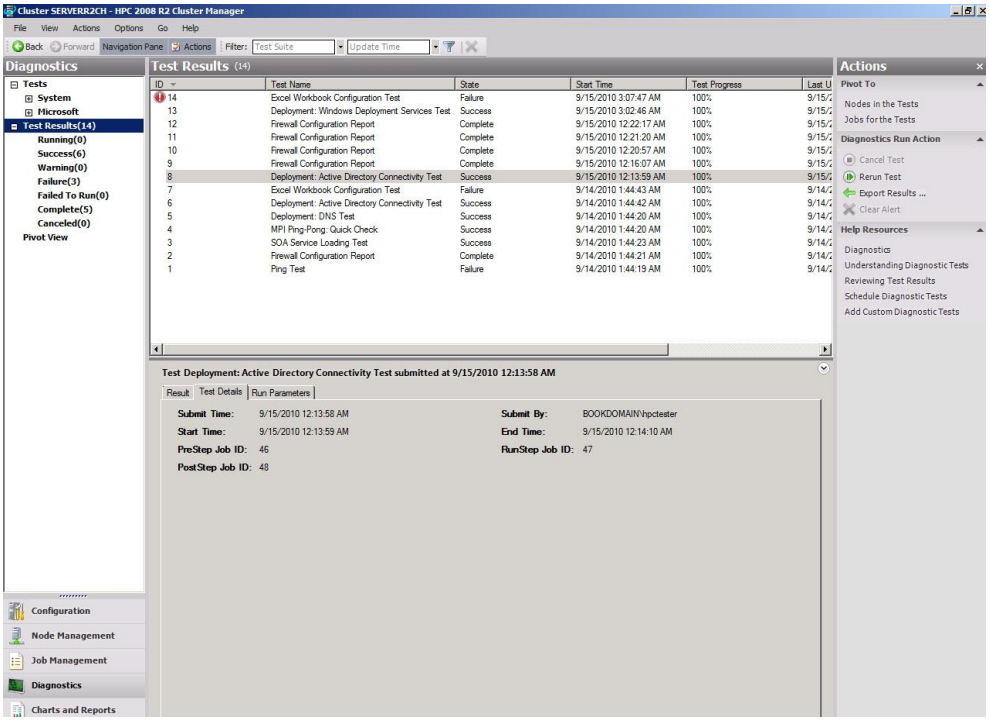



*Figure 2-27 Diagnostics View*

HPC Pack 2008 R2 provides built-in diagnostic tests to help system administrators to troubleshoot failures conveniently. System administrators can view the diagnostic test list, run tests, and check the results. These diagnostic tests include the tests for system and for Microsoft Excel Services for HPC. For example, the tests include deployment environment validator, network services, NIC configuration, performance, services, SOA, system configuration, and Excel Runner installation verification.

### 2.4.4.2 Customize Diagnostic Test Framework

The built-in tests are targeted only for HPC Server system software. While in many cases, failures are due to applications and third-party devices. HPC Server 2008 R2 offers the functionality to customize diagnostic tests. Customers, third-party independent software vendors, independent hardware vendors, original equipment manufacturers (OEMs), and system integrators can add their customized tests into the cluster so that it is simple and effective for system administrators to diagnose applications and hardware devices.

For more information about how to use diagnostic tests or create customized diagnostic tests, see Chapter 6, "Manage HPC Server Clusters".

## 2.4.5 Charts and Reports

R2 provides built-in reports and an extensible reporting framework. Administrators can use report generation tools such as SQL Server Reporting Service to author reports by the reporting framework.

### 2.4.5.1 Built-in Reports

As shown in Figure 2-28, system administrators are able to know the usage of the cluster from the built-in reports, such as resource usage like CPU or memory, job throughput, and turnaround time. System administrators can group the usage data by users, projects, or services, and then create built-in reports.

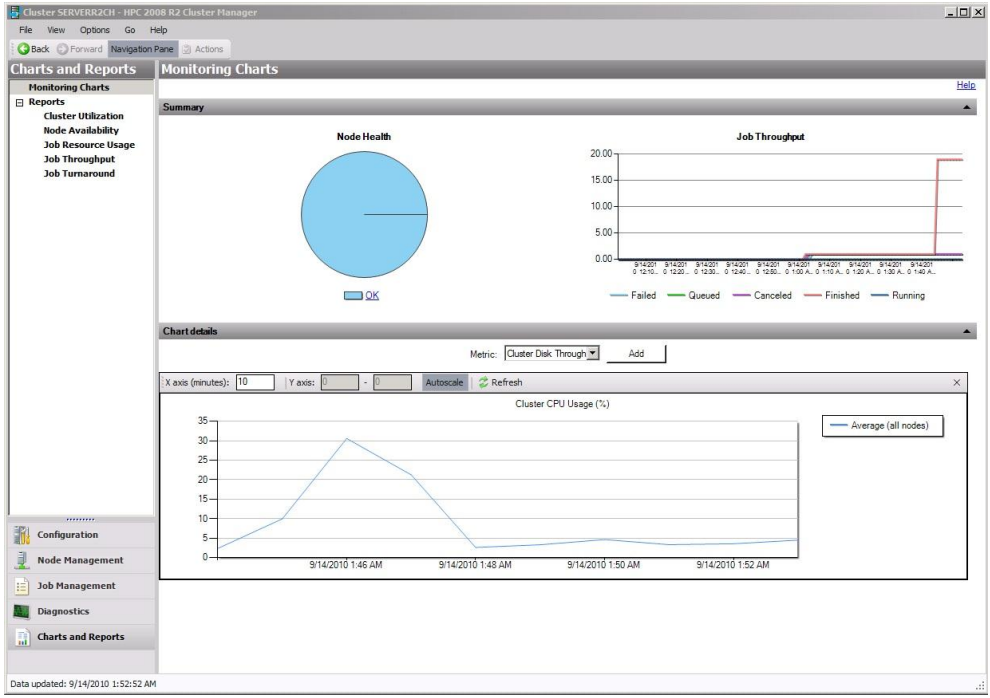



*Figure 2-28 Charts and Reports*

#### 2.4.5.2 Reporting Extensibility Framework

To fulfill the needs of customizing reports from customers, R2 collects more data than the previous version, including node properties, node historical events, job properties, task resource usage, job final states, and resource allocation history. The data is stored in the reporting database on the head node and can be exported to other database servers or data warehouses. With data retrieved through reporting database views and PowerShell cmdlets, administrators can use SQL Server Reporting Service or Excel to author reports and charts for daily operation analysis, chargeback, cluster utilization analysis, and capacity planning.

For more information about using reports, see Chapter 6, "Manage HPC Server Clusters".

### 2.4.6 Windows PowerShell Support

Windows HPC Pack 2008 R2 is not only compatible with current Windows Compute Cluster Server 2003 command line scripts, but also adds support for Windows PowerShell. Windows PowerShell provides over 130 standard command line tools. It is a brand new scripting language that focuses on management and has consistent grammar and practical tools. It can accelerate the automation of system management tasks and help companies to improve their abilities in solving unique system administration problems in some environments. Windows PowerShell is used together with Microsoft .NET Framework objects. Users and administrators can use object-oriented language through the simple and easy interactive shell.

For those users that are not familiar with Windows PowerShell, see the following two simple script examples. They briefly describe the functionality and convenience of this language. In the first script, the administrator wants to add the nodes with at least eight processors into a node group called FastNodes. Then a job is submitted to this node group to test the group.

```
Get-HpcNode | where {$_.NumProcessors -gt 8} | Add-HpcGroup "FastNodes"
New-HpcJob –Name "TestJob" –NodeGroup "FastNodes" | Submit-HpcJob
```

This simple script demonstrates the power of the language. Cmdlet **Get-HpcNode** returns nodes as objects, and the objects are passed into *where* (the alias for *Where-Object*), which checks the properties of the node object and determines the processor counts that the node contains. Then, nodes with eight or more processors are passed into the **Add-HpcGroup** cmdlet. This cmdlet creates a new node group called FastNodes. A new job is created on the FastNodes group and submitted by the **Submit-HpcJob** cmdlet.

Another example shows how Windows PowerShell and Windows Management Instrumentation (WMI) cooperate to accomplish more complex tasks. In this example, the administrator wants to get a list of all the nodes that contain more than 1 GB of free disk space.

```
$nodes=Get-HpcNode


foreach ($item in $nodes) {
$disks=get-wmiobject -computername $item.instancename -class win32 logicaldisk
   |\    where {$_.freespace/$gb -gt 1gb}


if ($disks.count -gt 0) {
   write-host "name:" $item.instancename
  }
}
```

### 2.4.7 Head Node Failover Cluster

One of the important features in HPC Pack 2008 R2 is to support failover cluster in Windows Server 2008. Windows HPC Pack 2008 R2 integrates the failover cluster in Windows Server 2008 Enterprise, and ensures the easy configuration of the head node and the failover node by using SQL Server failover cluster. When failure occurs on the head node, the failover cluster will automatically (or manually when needed) perform failover for the job scheduler. The job scheduler client will not see the change of head node during failover (or back), which ensures uninterrupted cluster operations.

## 2.5 Security

A majority of users are running critical tasks on HPC clusters, so security should be guaranteed in the current infrastructure. Windows HPC Pack 2008 R2 user Active Directory to ensure role-based security on user jobs and cluster management. Job Scheduler runs jobs with the credentials and context from user; during the entire lifecycle of jobs, the credentials will be encrypted and stored together with the related jobs. This feature enables jobs to access the resource in the user context (for example, files or database), as well as enabling the system administrators to use Active Directory to apply the security policy.

The management operation of jobs are executed in the encrypted and authenticated channel. Users' credentials are valid only to the scheduler and computer management services. This design makes credentials separated and protected. This additional security feature makes the MS-MPI implementation much stronger, though it does not implement MPICH2. HPC inherits all-new security features from Windows Server 2008—network protection, role management, network strategy management, and bidirection firewall in the Enterprise network.

Another important feature is the patching wizard, which enables cluster administrators to schedule the updating of nodes, keeping the running jobs unaffected.

## 2.6 Summary

For the users who are looking for an effective solution for high-performance computing, Windows HPC Server 2008 R2 offers a complete platform based on Windows Server 2008 R2, and simplifies the deployment, management, and integration with the current infrastructure, which greatly improves the productivity of the system administrator, the application developer, and the end user. Windows HPC Pack 2008 R2 integrates the x64 computer features, security features from Active Directory and Windows Server 2008 R2 operating system, offering an economical, easy to use, high performance, and extensible HPC solution. Windows HPC Pack 2008 R2 uses node templates to simplify and speed the computer node deployment. As long as the computer gets connected to the network, the head node can add it to the cluster. The Microsoft MPI implementation uses a high-speed network direct driver integrated with Active Directory to improve the management and security with role-based strategy, and a centralized user interface model to give the user an easy interface for managing cluster and scheduling jobs. Windows HPC Server 2008 R2 job scheduler supports heterogeneous cluster and SOA.

# Chapter 3 Job Scheduler

Job Scheduler provides many ways to submit and manage jobs. Users can submit jobs by using the UI. By implementing the interfaces, third-party vendors can seamlessly integrate applications with Job Scheduler. Using either the script or PowerShell command line, administrators can write scripts to satisfy different user needs.

Chapter Highlights:

- Understand the concept and rules of Job Scheduler
- Use the user interface, the C# application interface, and the command line and powerful scripts
- Use the user interface to monitor applications

## 3.1 Concepts

A cluster contains the following elements.

- Nodes: A single physical or logical computer with one or more processors. A node can be a head node, a compute node, or a WCF Broker node.
- Queue: An entity that provides the functions of queuing and scheduling. Each Windows HPC Server 2008 R2 cluster contains only one queue. The pending job is in the queue, too. The completed job is periodically removed from the queue.
- Job: A set of user tasks. A job holds resource for the tasks.
- Task: A given program running on the compute nodes. It can be a serial program (single process) or a parallel program (using multithreaded, message passing interface (MPI) or OpenMP).

Job Scheduler queues the jobs and tasks. It allocates resources for jobs, starts the task on compute nodes, and monitors jobs, tasks, and the state of compute nodes.

## 3.2 Application install

1. Open the attached application examples and go to Chapter 3.
2. Create a working directory. The default value of the working directory is the user's home directory (%userprofile%, which usually points to C:\User\user_name\Document). Share this directory as Applications on the head node.
3. Create the output directory. (Create a new folder as the output directory and share this folder as Output on the head node.)
4. Copy AsianOptions.exe and Aggregate.exe to \\<headnode>\Applications.

   **Note:** There are two ways to deploy the applications: local installation or central installation. Local installation enables the quick startup of applications. Central installation leads to delays in startup due to the file transfer through the network.

5. Apply the caspol policy: AsianOptions.exe and Aggregate.exe are managed applications. To access a managed application from the network, the caspol policy is required. In the head node, run the following command (replacing <headnode> with your head node name):

   ```
   clusrun c:\windows\microsoft.net\Framework\v2.0.50727\caspol -q -m -ag 1.2 -url
   file:/ /<headnode>/Applications/* FullTrust
   ```

In the author's four nodes cluster, the results are shown in Figure 3-1.

```
Administrator: Command Prompt
Microsoft Windows [Version 6.1.7061]
Copyright (c) 2009 Microsoft Corporation.  All rights reserved.

C:\Windows\system32>clusrun c:\windows\microsoft.net\Framework\v2.0.50727\caspol
 -q -m -ag 1.2 -url file://shpc-0110/Applications/* FullTrust
Enter the password for 'REDMOND\mingqxu' to connect to 'SHPC-0110':
Remember this password? (Y/N)y
--------------------------- SHPC-0115 returns 0 ---------------------------
Microsoft (R) .NET Framework CasPol 2.0.50727.4927
Copyright (c) Microsoft Corporation.  All rights reserved.

Added union code group with "-url" membership condition to the Machine level.
Success
--------------------------- SHPC-0111 returns 0 ---------------------------
Microsoft (R) .NET Framework CasPol 2.0.50727.3053
Copyright (c) Microsoft Corporation.  All rights reserved.

Added union code group with "-url" membership condition to the Machine level.
Success
--------------------------- SHPC-0112 returns 0 ---------------------------
Microsoft (R) .NET Framework CasPol 2.0.50727.3053
Copyright (c) Microsoft Corporation.  All rights reserved.

Added union code group with "-url" membership condition to the Machine level.
Success
--------------------------- SHPC-0110 returns 0 ---------------------------
Microsoft (R) .NET Framework CasPol 2.0.50727.4908
Copyright (c) Microsoft Corporation.  All rights reserved.

Added union code group with "-url" membership condition to the Machine level.
Success
```

*Figure 3-1 Run Caspol to set the permission to the shared folder*

> Frequently Asked Questions Help: Cannot run clusrun
>
> If the message "Only administrators can run clusrun" is displayed, it means that the command prompt was opened without administrator privilege. Exit the current command prompt program. Click **Start**, click **All Programs**, click **Accessories**, right-click **Command Prompt**, and then click **Run as administrator**. A command prompt window with the title **Administrator: Command Prompt** is displayed.

## 3.3 Submit Job

Chapter 2 mentioned that there are multiple ways to submit jobs. This section describes three ways in detail: using the job console, using the command line or PowerShell, and using a C# application.

### 3.3.1 Using Job Manager to Submit a Job

1. Click Start, click All Programs, click Microsoft HPC Server 2008 R2, and then click HPC Job Manager.
2. In **Actions**, click **New Parametric Sweep Job...**. The dialog box shown in Figure 3-2 is displayed.

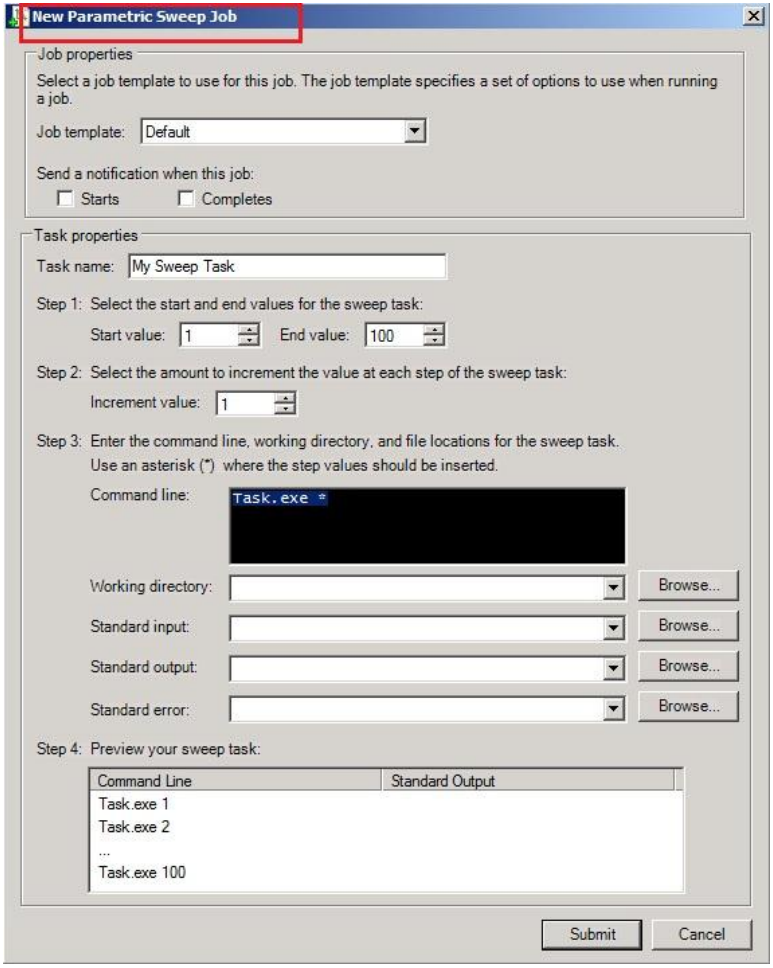



*Figure 3-2 New Parametric Sweep Job*

3. Type the name of the job "AsianOptions Valuation by Interest Rate" in the **Task name** text box.

   In **Step 1**, type 1 for **Start value** and 9 for **End value**.
   In **Step 3**, in **Command line**, type "AsianOptions.exe 1.0*".
   In **Working directory**, type "\\<headnode>\Applications". (Change <headnode> into the node name of your head node.)

4. Preview your sweep task. If the preview does not look like Figure 3-3, please modify the settings and preview again.

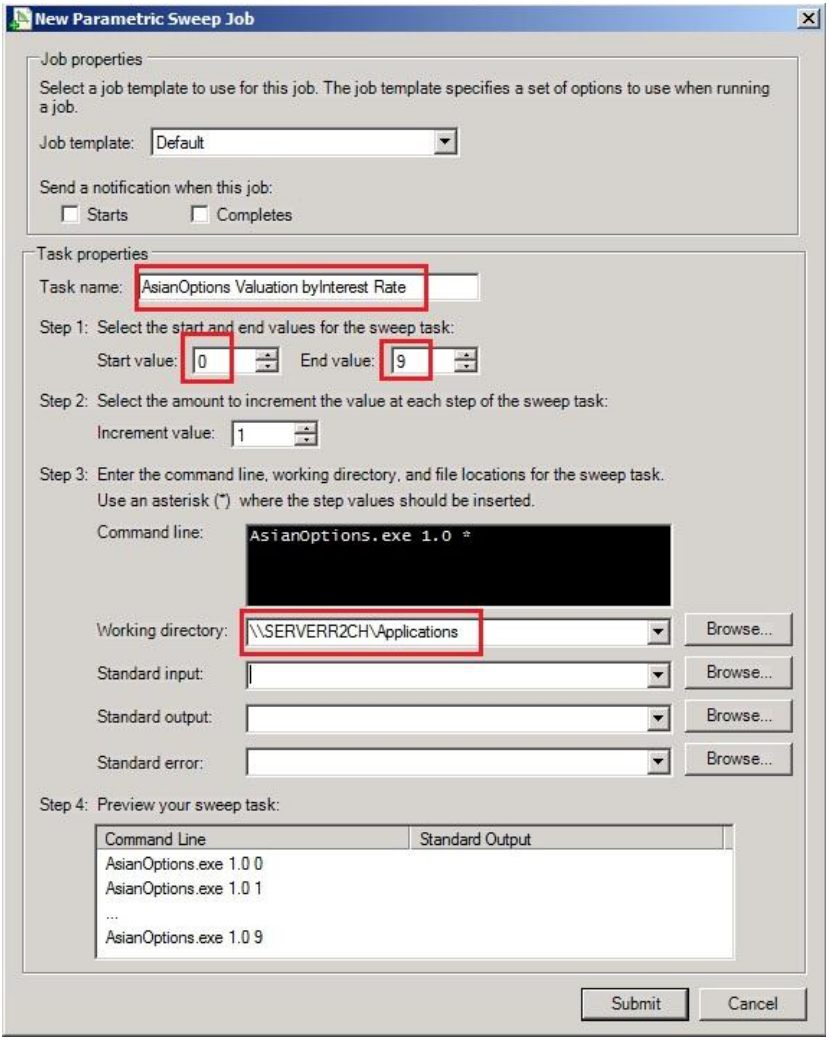



*Figure 3-3 Setting command line and working directory*

Notice:

Configuring a task's data access method is a critical factor in achieving optimal performance. The method should be different according to the data set size or reliability. If the data set is not intended to be modified very often and is relatively large, it should be configured as local data. If the data set is relatively small, it can be accessed by file share. Moving the data set to the compute node might be necessary if the data set is large and intended to be modified often. Windows HPC Server 2008 R2 supports parallel and high-performance file system to improve performance when accessing a very large data set. By specifying the working directory, users who use small and medium data sets will get the best performance. When the task starts, the compute node will be visible to all files in the working directory and will process the task correctly.

5. Click **Submit** to close the dialog. In **All Jobs** list view, you will see the job is running. The job will be finished in a few seconds, as shown in Figure 3-4.

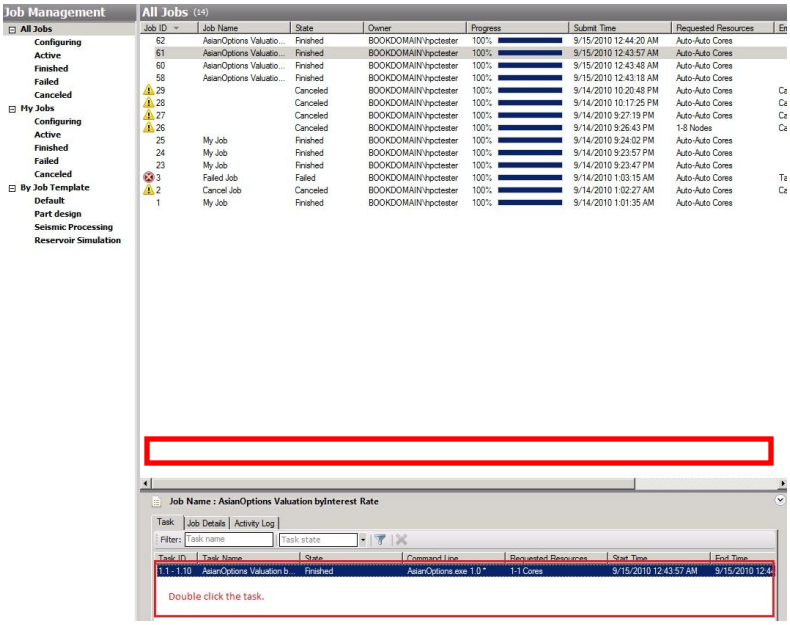



*Figure 3-4 Monitor Parametric Sweep Job*

6. In the **Task** tab, double-click Task ID **1.1-1.10** as shown in Figure 3-4. In the **View Job** dialog that opens, as shown in Figure 3-5, there's a list of all 10 tasks. When you select one task, you can see the calculation result in the **Output** text box. Take task 1.4 as an example. If you select that task, you will see the following output in the text box: "Option Price / interest rate (1.03) = 9.15572854540851". (**Note:** The result might be slightly different because we are using the Monte Carlo algorithm to calculate the result.) See Figure 3-5.

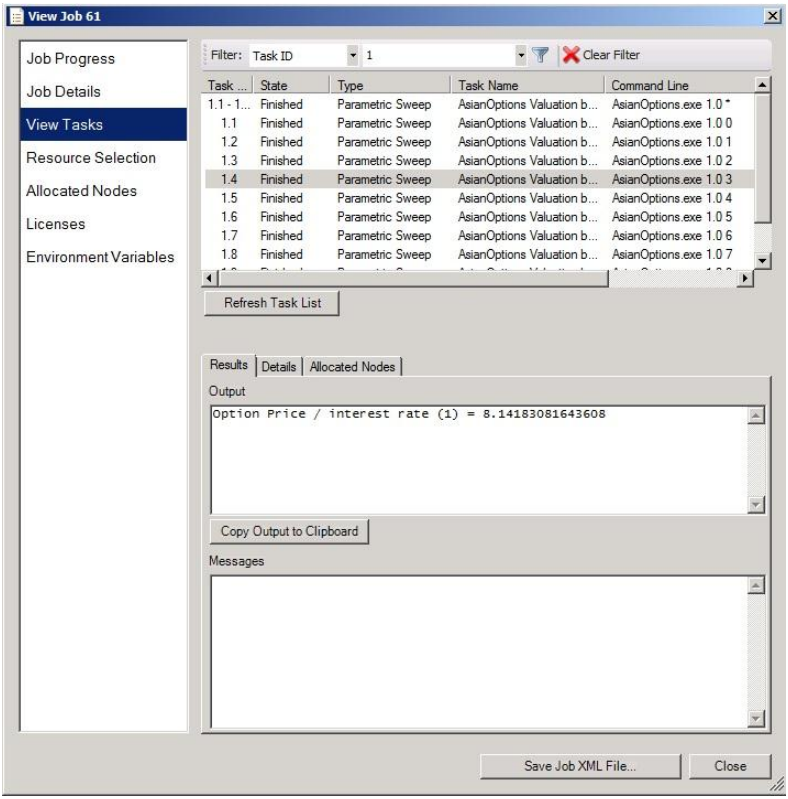



*Figure 3-5 View the detail of a parametric sweep task*

## 3.3.2 Using the Job Scheduler Command Line

1. Click Start, click All Programs, click Accessories, right-click Command Prompt, and then click Run as administrator.
2. In the command prompt window, type "job submit /parametric:0-9 /workdir:\\shpc-0110\Applications AsianOptions.exe 1.0*". This command will output the job's ID, as shown in Figure 3-6.

```
Administrator: Command Prompt

C:\Windows\system32>job submit /parametric:0-9 /workdir:\\shpc-0110\Applications
 AsianOptions.exe 1.0*
Job has been submitted. ID: 33.

C:\Windows\system32>_
```

*Figure 3-6 Use the command line to submit a parametric sweep job*

3. Monitor job state: Type "job view 33" in the command prompt window. The result is shown in Figure 3-7.

```
Administrator: Command Prompt

C:\Windows\system32>job view 33
Id                              : 33
State                           : Finished
Name                            :
Project Name                    :
Owner                           : REDMOND\mingqxu
Template                        : Default
Priority                        : Normal
Resource Request                : *-* cores
Type                            : Batch
Node Groups                     :
Requested Nodes                 :
Allocated Nodes                 :
Current Allocation              : 0 cores
Submit Time                     : 2009/9/27 15:43:05
Start Time                      : 2009/9/27 15:43:06
End Time                        : 2009/9/27 15:43:07
Elapsed Time                    : 00:00:00:01
Wait Time                       : 00:00:00:00
Run As                          : REDMOND\mingqxu
Pending Reason                  :
Error Message                   :
Progress                        : 100%
Progress Message                :
Task Count                      : 10
    Configuring tasks           : 0
    Queued tasks                : 0
    Running tasks               : 0
    Finished tasks              : 10
    Failed tasks                : 0
    Canceled tasks              : 0

C:\Windows\system32>_
```

*Figure 3-7 Use the command line to monitor a job*

4. Display the output of a task: Type "task view 33.1.<parametric sweep task ID>" (where <parametric sweep task ID> is the task number) in the command prompt window. For example, to display the output of the fourth task, type "task view 33.1.4" and press Enter. The result is shown in Figure 3-8.

```
Administrator: Command Prompt

C:\Windows\system32>task view 33.1.4
Task Id                          : 33.1.4
State                            : Finished
Task Name                        :
Command Line                     : AsianOptions.exe 1.03
Resource Request                 : 1-1 cores
Task Type                        : Parametric Sweep Task (0 sub-tasks)
Submit Time                      :
Start Time                       : 2009/9/27 15:43:06
End Time                         : 2009/9/27 15:43:07
Elapsed Time                     : 00:00:00:01
Total Kernel Time                : 124
Total User Time                  : 468
Working Set                      : 21016 KB
Processes                        :
Required Nodes                   :
Allocated Nodes                  : SHPC-0110
Pending Reason                   :
Exit Code                        : 0
Error Message                    :
Output                           :
Option Price / interest rate(1.03) = 9.16474611286181

C:\Windows\system32>
```

*Figure 3-8 Use the command line to display the output*

## 3.3.3 Using PowerShell

1. To run HPC PowerShell, click Start, click All Programs, click Microsoft HPC Pack 2008 R2, and right-click HPC PowerShell, and then click Run as administrator.
2. At the HPC PowerShell prompt, type "New-HpcJob" and press Enter. The result is shown in Figure 3-9.

```
Administrator: HPC PowerShell
PS C:\Windows\system32> New-HpcJob

Id      Name      State          Owner                Priority      Number
                                                                     OfTask
                                                                     s
--      ----      -----          -----                --------      ------
35                Configuring    REDMOND\mingqxu                    0

PS C:\Windows\system32>
```

*Figure 3-9 Use PowerShell to create a job*

3. Type "Add-HpcTask -JobId 35 -Parametric -Start 0 -End 9 -WorkDir \\shpc-0110\Applications -CommandLine "AsianOptions.exe 1.0*" and press Enter. The result is shown in Figure 3-10.

```
Administrator: HPC PowerShell
PS C:\Windows\system32> New-HpcJob

Id        Name        State            Owner                 Priority        Number
                                                                             OfTask
                                                                             s
--        ----        -----            -----                 --------        ------
35                    Configuring      REDMOND\mingqxu                       0


PS C:\Windows\system32> Add-HpcTask -JobId 35 -Parametric -Start 0 -End 9 -WorkD
ir \\shpc-0110\Applications -CommandLine "AsianOptions.exe 1.0*"

Id        Name        State            Owner                 Priority        Number
                                                                             OfTask
                                                                             s
--        ----        -----            -----                 --------        ------
35                    Configuring      REDMOND\mingqxu                       10


PS C:\Windows\system32> _
```

*Figure 3-10 Use PowerShell to add tasks*

4. Type "Submit-HpcJob -Id 35" and press Enter. The result is shown in Figure 3-11.

```
Administrator: HPC PowerShell
                                                                             s
--        ----        -----            -----                 --------        ------
35                    Configuring      REDMOND\mingqxu                       0


PS C:\Windows\system32> Add-HpcTask -JobId 35 -Parametric -Start 0 -End 9 -WorkD
ir \\shpc-0110\Applications -CommandLine "AsianOptions.exe 1.0*"

Id        Name        State            Owner                 Priority        Number
                                                                             OfTask
                                                                             s
--        ----        -----            -----                 --------        ------
35                    Configuring      REDMOND\mingqxu                       10


PS C:\Windows\system32> Submit-HpcJob -Id 35

Id        Name        State            Owner                 Priority        Number
                                                                             OfTask
                                                                             s
--        ----        -----            -----                 --------        ------
35                    Running          REDMOND\mingqxu       Normal          10


PS C:\Windows\system32> _
```

*Figure 3-11 Use PowerShell to submit a job*

5. Monitor job state: Type "Get-HpcJob –Id 35" and press Enter. The result is shown in Figure 3-12.

```
Administrator: HPC PowerShell
                                                                    OfTask
                                                                    s
--      ----      -----        -----                --------        ------
35                Configuring  REDMOND\mingqxu                      10

PS C:\Windows\system32> Submit-HpcJob -Id 35

Id      Name      State        Owner                Priority        Number
                                                                    OfTask
                                                                    s
--      ----      -----        -----                --------        ------
35                Running      REDMOND\mingqxu      Normal          10

PS C:\Windows\system32> Get-HpcJob -Id 35

Id      Name      State        Owner                Priority        Number
                                                                    OfTask
                                                                    s
--      ----      -----        -----                --------        ------
35                Finished     REDMOND\mingqxu      Normal          10

PS C:\Windows\system32>
```

*Figure 3-12 Use PowerShell to monitor a job*

6. Display task output: To display the output of the fourth task, type "Get-HpcTask -JobId 35 -TaskId 1 -SubTaskId 4 |fl" and press Enter. The result is shown in Figure 3-13.

```
Administrator: HPC PowerShell
PS C:\Windows\system32> Get-HpcTask -JobId 35 -TaskId 1 -SubTaskId 4 |fl


Id                        : 1
SubTaskId                 : 4
State                     : Finished
Name                      :
CommandLine               : AsianOptions.exe 1.03
RequiredNodes             :
AllocatedNodes            : SHPC-0110
CurrentAllocation         : 0
ExitCode                  : 0
ErrorMessage              :
Depend                    :
Output                    : Option Price / interest rate(1.03) = 9.1631250336674
                            5

Rerunnable                : True
Exclusive                 : False
PendingReason             :
Env                       :
Stderr                    :
Stdin                     :
Stdout                    :
WorkDirectory             : \\shpc-0110\Applications
CreateTime                : 2009/9/27 16:21:50
SubmitTime                :
StartTime                 : 2009/9/27 16:24:12
EndTime                   : 2009/9/27 16:24:13
RunTime                   :
ElapsedTime               : 00:00:01.7500000
UnitType                  : Core
MaxNodes                  : 1
MinNodes                  : 1
MaxSockets                : 1
MinSockets                : 1
MaxCores                  : 1
MinCores                  : 1
Type                      : ParametricSweep
TotalSubTaskCount         : 0
InstantiatedSubTaskCount  : 0
Parametric                : True
SweepStartValue           : 0
SweepEndValue             : 9
SweepIncrementValue       : 1
```

*Figure 3-13 Use PowerShell to display the result of a task*

Table 3-1 presents a summary of using the command line and PowerShell.

*Table 3-1 Summaries of using Job Scheduler command line and PowerShell*

| **Operations** | **Command line** | **PowerShell** |
|---|---|---|
| Create and submit job | **Job submit** | **New-HpcJob, Add-HpcTask, Submit-HpcJob** |
| Display a list of active (queued and running) jobs | **Job list** | **Get-HpcJob** |
| Display a list of all jobs | **Job list /all** | **Get-HpcJob –Owner * -State All** |
| Display the details of a job | **Job view** *<JobID>* | **Get-HpcJob –Id** *<JobID>* |
| Display all tasks in a job | **Job listtasks** *<JobID>* | **Get-HpcTask –JobId** *<JobID>* |
| Display the details of a parametric sweep task | **Task view** *<JobID>***.***<TaskID>*.*<SubTaskId>* | **Get-HpcTask –JobId** *<JobID>* **-TaskId** *<TaskID>* -SubTaskId *<SubTaskId>* \| fl<br>Cls |

## 3.3.4 Using C# Applications

HPC Pack 2008 API provided access to the Job Scheduler mechanism. By using these interfaces to write applications or scripts, users can connect to the cluster and manage the jobs, resources, tasks, nodes, environment variables, extended job cycle, and more.

In the simplest case, six steps are required to use HPC Pack 2008 R2 API:

1. Install the SDK.
2. Connect to the cluster.
3. Create the job.
4. Create the task.
5. Add the task into the job.
6. Submit the job/task.

Use the **IScheduler.Connect** method to connect to the cluster, the **IScheduler.CreateJob** method to create a job, the **ISchedulerJob.CreateTask** method to create a task, and the **ISchedulerJob.AddTask** method to add the task into the job. Use the **ISchedulerJob.SubmitJob** method to submit the job. To receive the job state change event, use the delegate **job_OnJobState**.

### 3.3.4.1 Create the Console Application

1. Start Visual Studio 2008. Click **File**, click **New**, and then click **Project**.

   In the **Project types** list view of the **New Project** diagram, select Visual C#.
   In Templates list view, click Console Application.

   Type "SubmitAndMonitorJob" in the **Name** text box, as shown in Figure 3-14.

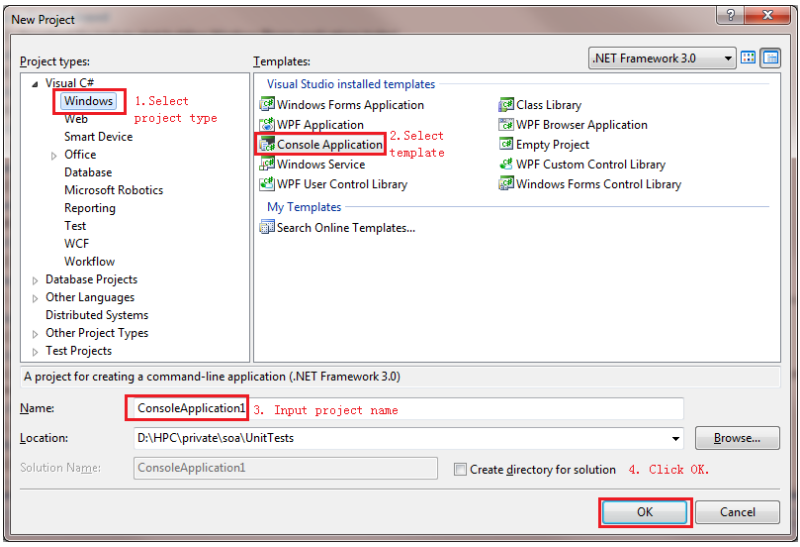



*Figure 3-14 Create the C# Project*

Click **OK** to go back to the main window of Visual Studio. You will see a Program.cs source file that contains the initial code.

2. In this project, we will add the required references using the GUI. The steps are listed below.
    1) In the **Solution Explorer** window, right-click **References**, and then click **Add Reference...**.
    2) Click the **Browse** tab in the **Add Reference** diagram. Browse to the folder where you installed HPC Pack 2008 R2 SDK (for example, C:\Program Files\Microsoft HPC Pack 2008 R2 SDK\Bin\).
    3) Choose Microsoft.Hpc.Scheduler.dll and Microsoft.Hpc.Scheduler.Properties.dll, as shown in Figure 3-15.

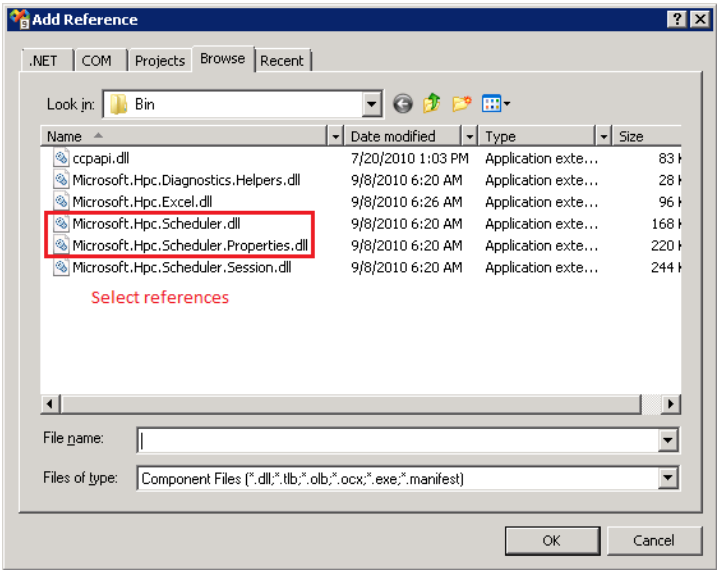



*Figure 3-15 Add the HPC SDK Reference*

Click **OK** and you will see Microsoft.Hpc.Scheduler.dll and Microsoft.Hpc.Scheduler.Properties.dll listed in the references, as shown in Figure 3-16.

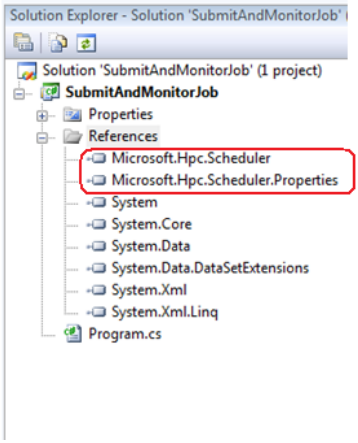



*Figure 3-16 Reference tree view of the HPC SDK included*

Now we can write the code to submit and monitor jobs.

### 3.3.4.2 Write Code to Submit the Job

1. In the **Solution Explorer** window, select Program.cs. The code shown in Code 3-1 is displayed.

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;
namespace SubmitAndMonitorJob
{
    class Program
    {
        static void Main(string[] args)
        {
        }
    }
}
```

*Code 3-1 Template of the console application source file*

2. Add the "using Microsoft.Hpc.Scheduler;" and "using Microsoft.Hpc.Scheduler.Properties;" namespaces.

3. Add code to connect to Job Scheduler: Add the code in method Main(), as shown in bold in Code 3-2.

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;
using Microsoft.Hpc.Scheduler;
using Microsoft.Hpc.Scheduler.Properties;

namespace SubmitAndMonitorJob
{
    class Program
    {
        static void Main(string[] args)
        {
            Scheduler scheduler = new Scheduler();
            scheduler.Connect("headnode");

        }
    }
}
```

*Code 3-2 Connect to the head node of the cluster*

You need to replace "headnode" with the head node name of the cluster.

4. Add code to create and submit jobs.

First we need to create a job, as shown in Code 3-3.

```
            // Create job
            ISchedulerJob job = scheduler.CreateJob();
```

*Code 3-3 Create a job*

There are many properties to configure for a job. Because we are going to add a parametric sweep task, we are not going to specify the number of cores so that Job Scheduler could dynamically allocate compute nodes according to the runtime resource allocation status. This is called the grow/shrink policy of Job Scheduler. In Code 3-4, we add a parametric sweep task.

```
            ISchedulerTask task = job.CreateTask();


            task.Type = TaskType.ParametricSweep;
            task.StartValue = 0;
            task.EndValue = 9;
            task.WorkDirectory = @"\\headnode\Applications";
            task.CommandLine = "AsianOptions.exe 1.0*";
```

*Code 3-4 Create a parametric sweep task*

Then, we add this task into the job as shown in Code 3-5.

```
            job.AddTask(task);
```

*Code 3-5 Add the task into the job*

As a final step, we submit the job and output the job's ID as shown in Code 3-6.

```
            scheduler.SubmitJob(job, null, null);
            Console.WriteLine("Job {0} has been submitted", job.Id);
```

*Code 3-6 Submit the job*

The **SubmitJob()** method takes three parameters: the job itself, username, and password. If username remains null, the logon user will be used. If the password is null, the user will be prompted to input the password. By default, the user will be prompted in the console.

> Hint: How to use the pop-up dialog box prompts?
>
> You will need to use the instance method of the scheduler class:
>
> In the Main() method, add the following line:
>
> `scheduler.SetInterfaceMode(false, (System.IntPtr)null);`
>
> The first parameter of the **SetInterfaceMode()** method indicates whether to use console mode. The second parameter indicates the handle of the father window.

Now we can compile the code by clicking Build, and then clicking Build Solution. If there's no error, click Debug, and then click Start Without Debugging to run the application, as shown in Figure 3-17.

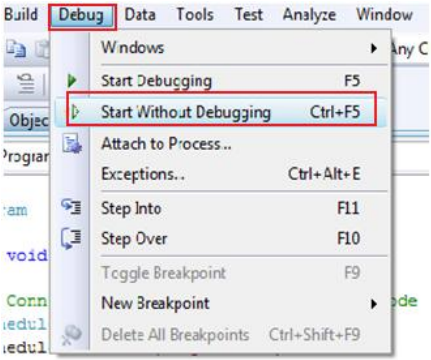



*Figure 3-17 Run the application*

If you did not store your password in the cache, you will see the following password prompt. Input the password and select **Remember my password**, as shown in Figure 3-18. Then click **OK**.

*Figure 3-18 Password prompt dialog*

The text output from the console shown in Figure 3-19 is displayed.

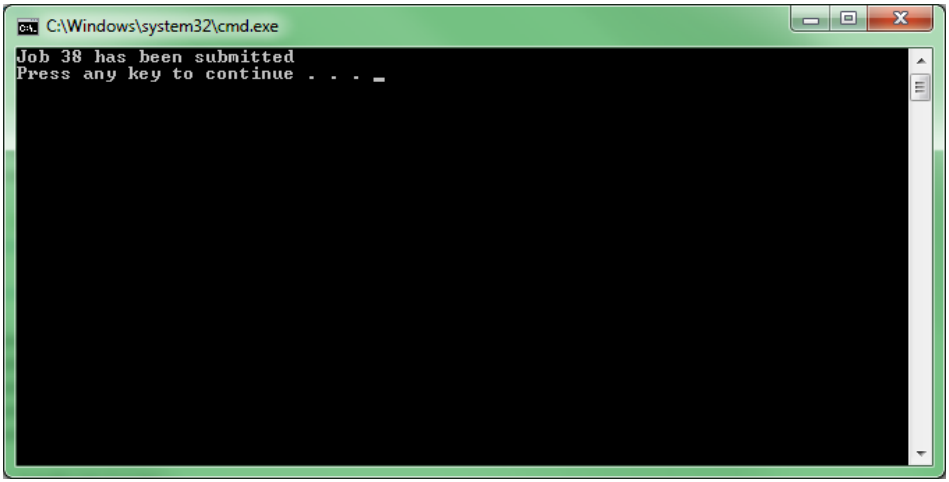



*Figure 3-19 Standard output of the console application*

"Job 38 has been submitted" is printed by the application. Now we can open the job manager to monitor the job state.

### 3.3.4.3 Use a C# application to monitor the job state

1. Before submitting the job, add an event handler to a job event, as shown in Code 3-7.

```
job.OnJobState += new EventHandler<JobStateEventArg>(job_OnJobState);
```

*Code 3-7 Add an event handler to a job event*

2. Add the static method **job_OnJobState** to display the job state change event, as shown in Code 3-8.

```
static void job_OnJobState(object sender, JobStateEventArg e)
{
    Console.WriteLine("Job <{0}> has changed from state <{1}> to <{2}>",
e.JobId, e.PreviousState.ToString(), e.NewState.ToString());
}
```

*Code 3-8 Static method to display the job state change event*

The user's application should be as shown in Code 3-9.

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;
using Microsoft.Hpc.Scheduler;
using Microsoft.Hpc.Scheduler.Properties;

namespace SubmitAndMonitorJob
{
    class Program
    {
        static void Main(string[] args)
        {
            // Connect to the scheduler on the headnode
            Scheduler scheduler = new Scheduler();
            scheduler.Connect("shpc-0110");
            scheduler.SetInterfaceMode(false, (System.IntPtr)null);

            // Create job
            ISchedulerJob job = scheduler.CreateJob();

            ISchedulerTask task = job.CreateTask();

            task.Type = TaskType.ParametricSweep;
            task.StartValue = 0;
            task.EndValue = 9;
            task.WorkDirectory = @"\\headnode\Applications";
            task.CommandLine = "AsianOptions.exe 1.0*";

            job.AddTask(task);
```

```
            job.OnJobState += new
EventHandler<JobStateEventArg>(job_OnJobState);


            // Submit the job
            scheduler.SubmitJob(job, null, null);
            Console.WriteLine("Job {0} has been submitted", job.Id);
        }


        static void job_OnJobState(object sender, JobStateEventArg e)
        {
            Console.WriteLine("Job <{0}> has changed from state <{1}> to
<{2}>",
                              e.JobId, e.PreviousState.ToString(),
e.NewState.ToString());
        }
    }
}
```

*Code 3-9 The entire program*

Compile the application and run it. The results are shown in Figure 3-20.

*Figure 3-20 Output of the application*

We can see that the job passes the Configuring, Submitted, Validating, and Queued states. Then the application ends. Why didn't we see the job go into the Finished state?

> Hint: Synchronization between the event processing thread and main thread
>
> Among the above outputs, "Job <39> has changed from state <…> to <…>" was printed by the event processing thread. But "Job <39> has been submitted" was printed by the main thread. After the main thread printed this, the main thread is finished and thus the entire application ended. So we didn't see the job finished state. We need to force the main thread not to finish before the job is finished by adding thread synchronization code.

3. Thread synchronization: Find the sentence as shown in Code 3-10.

```
        static void Main(string[] args)
```

*Code 3-10 Main method*

Then add a declaration of an instance of the **ManualResetEvent** class as shown in bold in Code 3-11.

```
    class Program
    {
        private static ManualResetEvent jobFinishEvt = new
ManualResetEvent(false);
        static void Main(string[] args)
```

*Code 3-11 Introduce the thread synchronization variable*

*jobFinishEvt* is a variable to synchronize threads. When it is initialized as **ManualResetEvent(false)**, the thread that calls **jobFinishEvt.WaitOne()** will be blocked.

So add the call to **jobFinishEvt.WaitOne()** after submitting the job in the main thread, as shown in bold in Code 3-12.

```
            scheduler.SubmitJob(job, null, null);
            Console.WriteLine("Job {0} has been submitted", job.Id);
            jobFinishEvt.WaitOne();
```

*Code 3-12 Main thread waits for the job to finish*

Then, in the event handler, we need to check whether the job is finished and call the **jobFinishEvt.Set()** method if it is, as shown in bold in Code 3-13.

```
        static void job_OnJobState(object sender, JobStateEventArg e)
        {
            Console.WriteLine("Job <{0}> has changed from state <{1}> to
<{2}>",
                              e.JobId, e.PreviousState.ToString(),
e.NewState.ToString());
            if (e.NewState.Equals(JobState.Finished))
            {
                Console.WriteLine("Job <{0}> has finished", e.JobId);
                jobFinishEvt.Set();
            }
        }
```

*Code 3-13 The event processing thread uses jobFinishEvent to synchronize with the main thread*

**JobState.Finished** is declared in assembly **Microsoft.Hpc.Scheduler.Properties**. So we need to add "using Microsoft.Hpc.Scheduler.Properties;".

After **jobFinishEvt.Set()** has been called, the blocked main thread will continue.

The complete program is shown in Code 3-14.

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;
using Microsoft.Hpc.Scheduler;
using System.Threading;
using Microsoft.Hpc.Scheduler.Properties;

namespace SubmitAndMonitorJob
{
    class Program
    {
        private static ManualResetEvent jobFinishEvt = new
ManualResetEvent(false);
        static void Main(string[] args)
        {
            // Connect to the scheduler on the headnode
            Scheduler scheduler = new Scheduler();
            scheduler.Connect("headnode");
            scheduler.SetInterfaceMode(false, (System.IntPtr)null);

            // Create job
            ISchedulerJob job = scheduler.CreateJob();

            ISchedulerTask task = job.CreateTask();

            task.Type = TaskType.ParametricSweep;
            task.StartValue = 0;
            task.EndValue = 9;
            task.WorkDirectory = @"\\headnode\Applications";
            task.CommandLine = "AsianOptions.exe 1.0*";

            job.AddTask(task);

            job.OnJobState += new
EventHandler<JobStateEventArg>(job_OnJobState);
```

```
            // Submit the job
            scheduler.SubmitJob(job, null, null);
            Console.WriteLine("Job {0} has been submitted", job.Id);
            jobFinishEvt.WaitOne();
        }

        static void job_OnJobState(object sender, JobStateEventArg e)
        {
            Console.WriteLine("Job <{0}> has changed from state <{1}> to
<{2}>",
                              e.JobId, e.PreviousState.ToString(),
e.NewState.ToString());
            if (e.NewState.Equals(JobState.Finished))
            {
                Console.WriteLine("Job <{0}> has finished", e.JobId);
                jobFinishEvt.Set();
            }
        }
    }
}
```

*Code 3-14 The entire program*

Compile and run the application. Now you can see that the job goes through the Queued, Running, and, finally, the Finished state. The application ends after the job is finished, as shown in Figure 3-21.

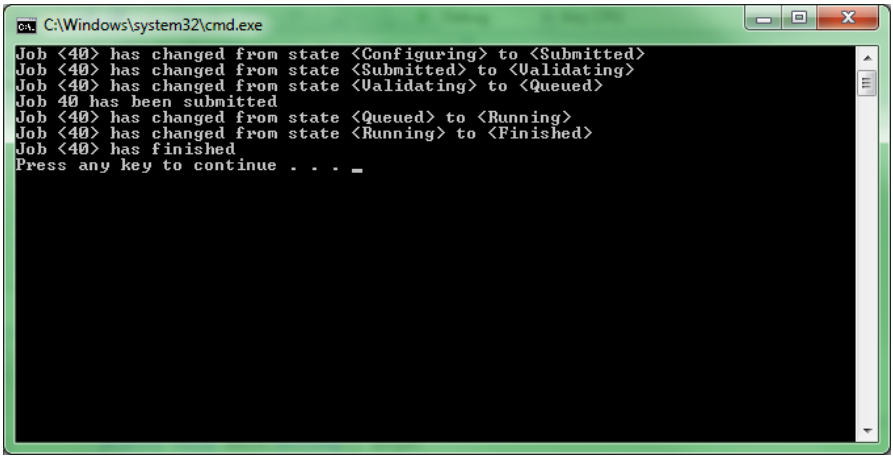



*Figure 3-21 Standard output of the application*

## 3.4 Tune Resource Allocation

The following are some real user scenarios that introduce the job scheduling system.

### 3.4.1 Submit a big model of MPI applications

Assume that you need to run an eight-way MPI application and hope each node has at least 16 GB of memory. If your application at run time requires a lot of memory, you can use the following methods to submit the job:

1. Start HPC job manager. In the **Actions/Job Submission** pane, click **New job ...**. The **New job** dialog is displayed.
2. In the **New Job** dialog, type "A Big Model Job" in the **Job name** text box, type "Engine Design" in the **Project** text box, and select "Socket" from the **Select the type of resource to request for this job** drop-down option in the **Job resources** group box, as shown in Figure 3-22.

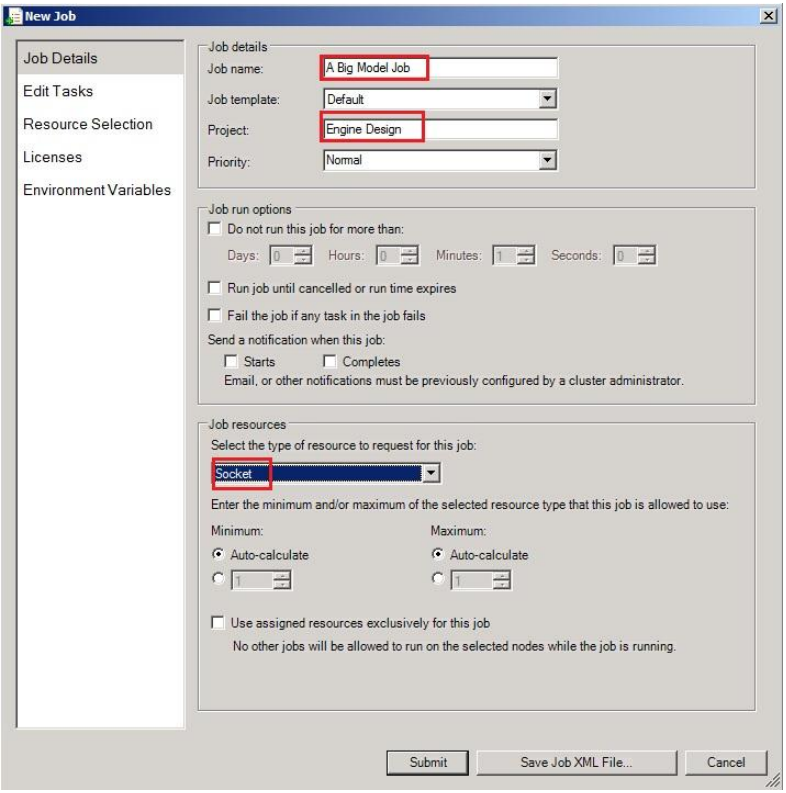



*Figure 3-22 Using the Socket resource type*

Note: Why use Socket?

To allocate resources more efficiently according to job usage pattern, job scheduler supports three job resource allocation types:

- *Core*: The core allocation type usually applies to the applications that could consume a large amount of CPU resources, such as parametric sweep job.
- *Socket*: The socket allocation type usually applies to the applications that are sensitive to memory access. The speed of the job usually depends on the amount of data operations read from memory. Therefore, running more jobs in the same memory bus will not improve performance since all of these jobs are in contention for the same memory path.
- *Node*: The node allocation type usually applies to the applications that are designed to run on node, such as I/O-intensive applications. Running multiple jobs within the same node will not improve performance since all of these jobs are waiting for access to the same disk or network.

In the **Job resources** group box, find the **Minimum** and **Maximum** labels. The default value is **Auto-calculate**. Click **Minimum** and enter 8, as shown in Figure 3-23.

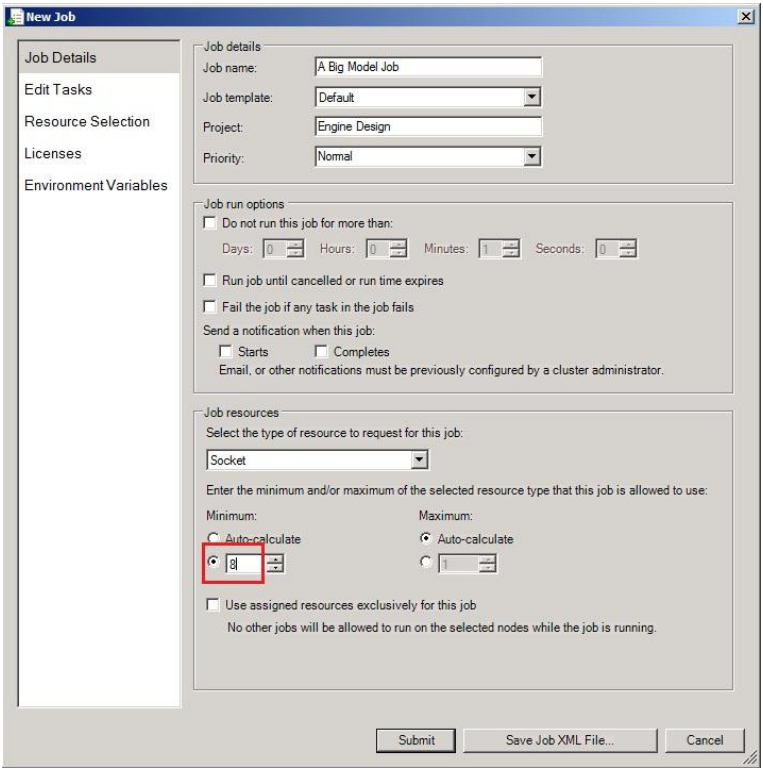



*Figure 3-23 Set Minimum and Maximum resource unit*

3. In the left pane, click Resource Selection. Here you can select a different node group, or specify the nodes to use. You can also specify the node's minimum memory or cores requirements. In the Hardware preferences group box, select Minimum memory (MB), and then enter 16000 (16 GB), as shown in Figure 3-24.

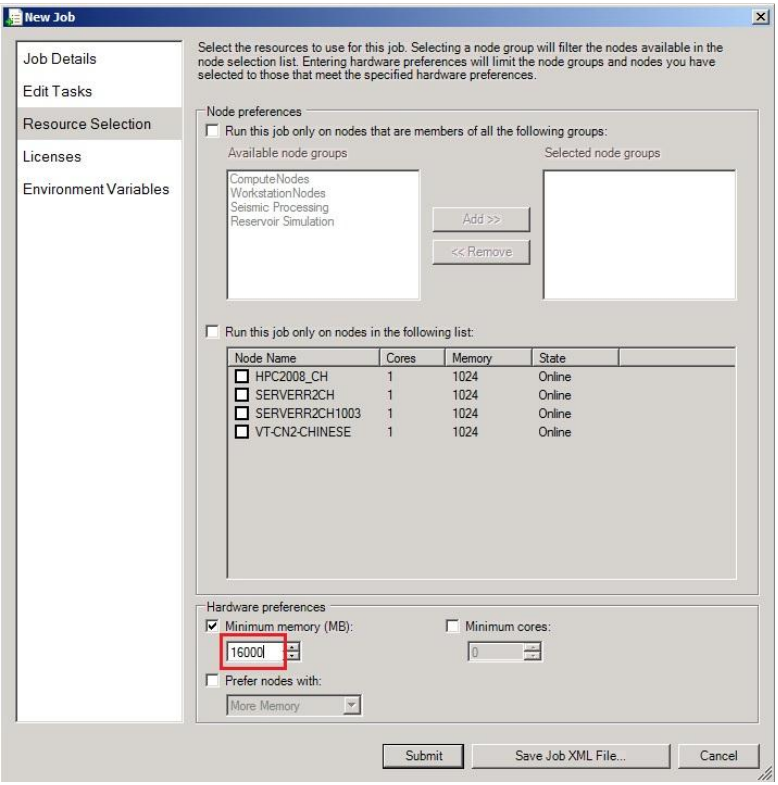



*Figure 3-2 Set minimum memory requirements*

4. After completing the resource request, input the command line to do the job.

   Click **Edit Tasks** in the left pane, click **Add** in the right pane, and, in the drop-down list, click **Basic Task ...**. The **Task Details and I/O Redirection** window is displayed.

   In the **Task name** text box, enter "MPI Task".

   In the **Command line** text box, enter the command to run, such as "mpiexec.exe myapp.exe".

   In the **Working directory** text box, enter the directory where Myapp.exe is located, such as "\\ <headnode>\Applications" (replace <headnode> with your head node name).

   Set both minimum and maximum resource units of the job to 8, as shown in Figure 3-25.

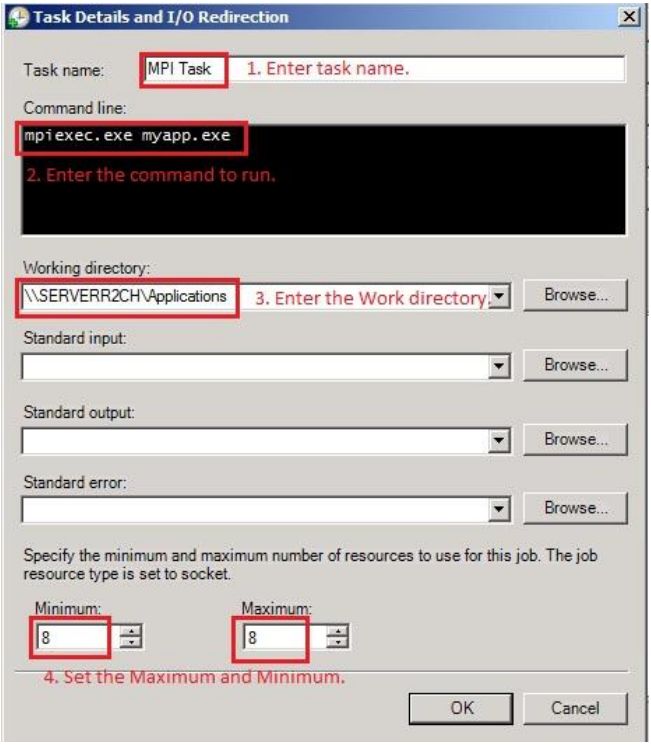



*Figure 3-25 Set the job command line, working directory, and minimum and maximum resource units values*

**Note**

You must use the Mpiexec command if the parallel application requires MS-MPI. The command used for a parallel task must comply with the following format:

mpiexec [mpi_options] <Myapp.exe> [arguments]

In the command, Myapp.exe is the name of the application. Users can use Job Manager or the command line to submit MS-MPI jobs.

**FAQ** : Why use the Mpiexec.exe command line without any resources parameters?

You may have become accustomed to using the Mpiexec.exe – n N or Mpiexec.exe –hosts parameter to start the MPI applications. So, without specifying it, how could Mpiexec know how many process to start and on which node?

MS-MPI and Job Scheduler have seamless integration. The resources that Mpiexec needs are provided by Job Scheduler. Therefore, users don't have to specify nodes; all they need to do is to set the resources requirement at the high level. Job Scheduler is responsible for allocation and scheduling.

Click **OK** to return to the **New Job** dialog box. You can see that a task is added to the task list. Click the task, and you will see the properties of the task, as shown in Figure 3-26.

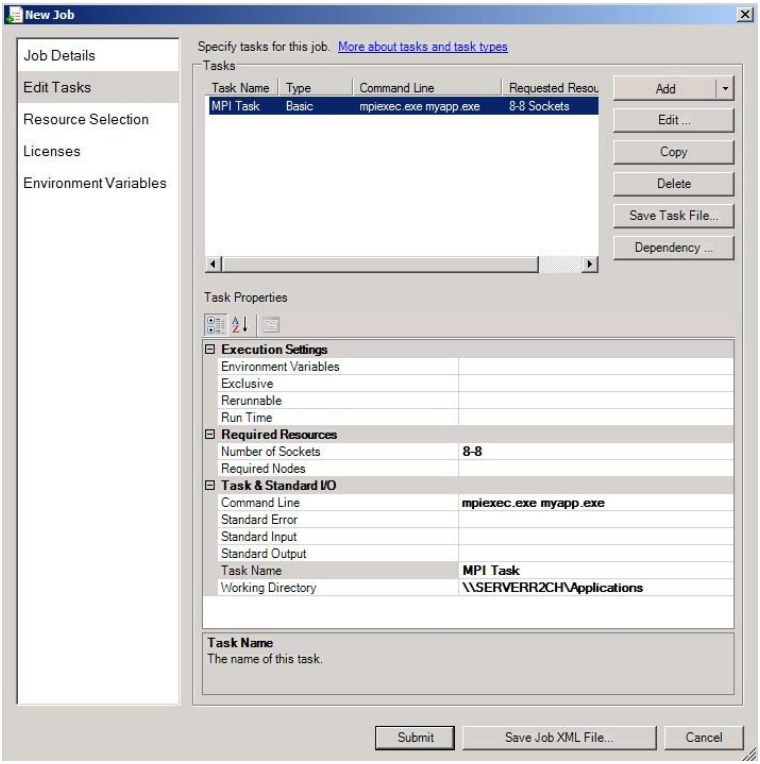



*Figure 3-26 Task property table*

5. You need to set many properties while submitting a task, and you often need to repeat this type of work. You can save jobs for future reuse. Click **Save Job XML File...**, and, in the **File Save** dialog box, type a file name, and then click **Save** to save the job XML file.
6. Click **Submit** to submit the job.

Command line and PowerShell script are shown in Table 3-2.

*Table 3-2*

| | |
|---|---|
| Command line | job submit /memorypernode:16000 /numsockets:8 /workdir:\\<headnode>\Applications mpiexec.exe myapp.exe (replace <headnode> with your head node name) |
| PowerShell | New-HpcJob –NumSockets 8 –MinMemoryPerNode 16000<br><br>Add-HpcTask –JobId <ID> –WorkDir \\<headnode>\Applications -CommandLine mpiexec.exe myapp.exe (Replace <headnode> with your head node name)<br><br>Submit-HpcJob –Id <ID> |

## 3.4.2 Submit a task flow

Now let's come back to the AsianOptions parametric sweep job. Recall that we have a total of 10 tasks, to calculate an Asian option price in the interest of 1.00 1.01 ... 1.09, respectively. When the tasks are completed, we hope to be able to put the data together, so that user does not need to start the Job Manager to look at these 10 tasks individually; instead, he can find them in a file. To achieve this, an application called Aggregate.exe is written. This code can go through all the files in a directory and summarize the results.

### 3.4.2.1 Using Job Manager

1. Start Job Manager.
2. In the navigation bar, click **All Jobs**, and then click **Finished**. In the job list, right-click the AsianOptions job and, in the pop-up menu, click **Copy Job...** to open the **Copy Job** dialog.
3. Click **Edit Tasks**.
4. Double-click the AsianOptions task.

   In the **Standard output** text box, type "\\ <headnode>\Output\ASout *" (replace <headnode> with your head node name), as shown in Figure 3-27.

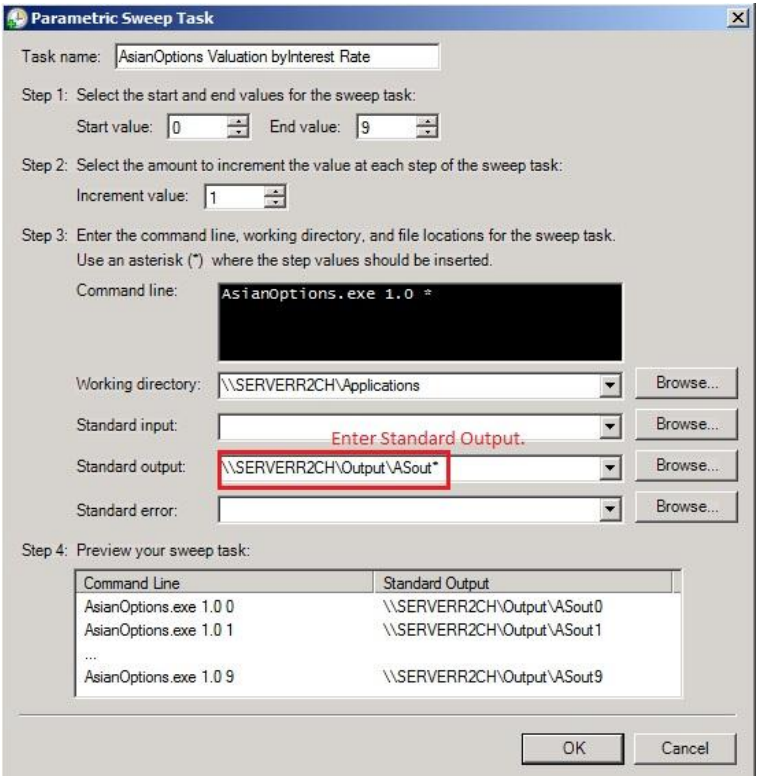



*Figure 3-27 Edit the parametric sweep job's task*

5. Click **Add**, select **Basic Task ...**, type a task name "Aggregate", type the command line "Aggregate.exe \\ <headnode>\Output" (replace <headnode> with your head node name), as shown in Figure 3-28.

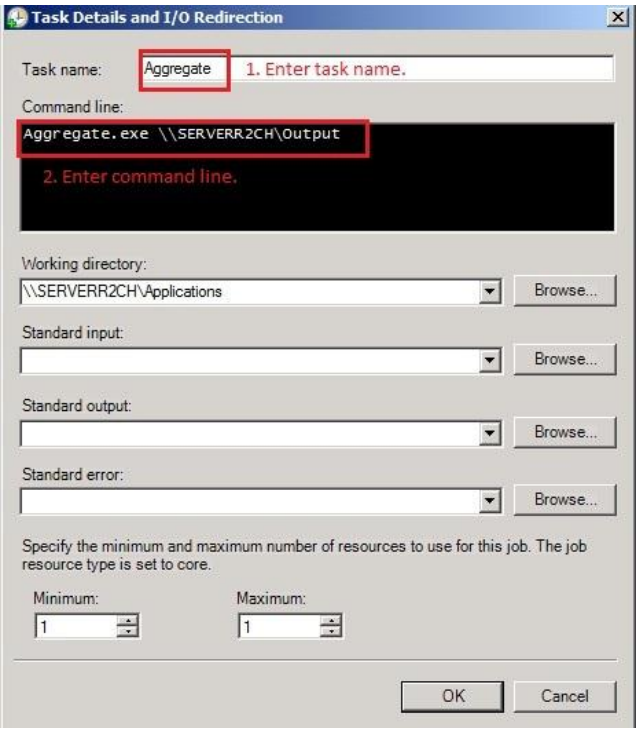



*Figure 3-28 Add a new task*

Click **OK**.

6. Click **Dependency...**, and the **Task Dependency** dialog box is displayed. In the **Task Dependency** dialog box, you can organize the tasks into several groups. A group of tasks would all be completed before the next group of tasks starts. Initially, all the tasks are in Group 1, as shown in Figure 3-29.

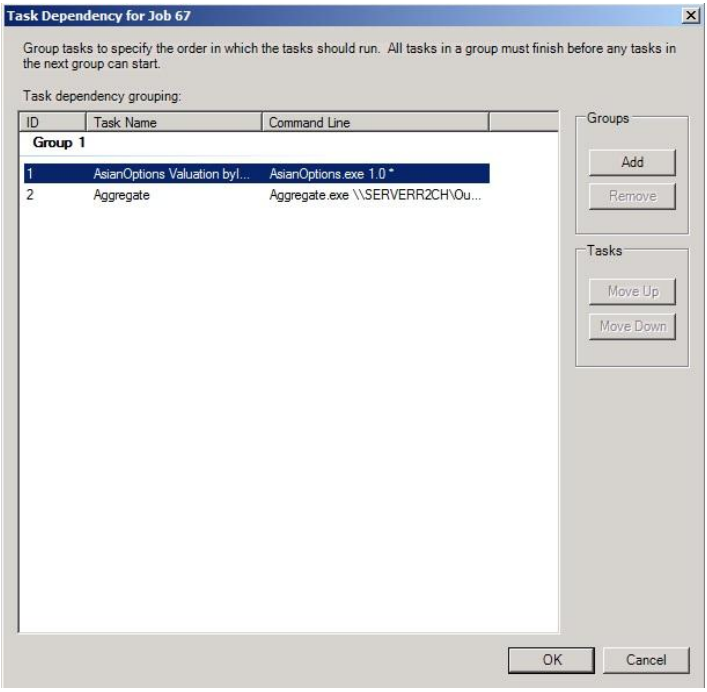



*Figure 3-29 Set up dependencies between tasks*

Click **Add**. Group 2 will be added to the **Task dependency grouping** pane, as shown in Figure 3-30.

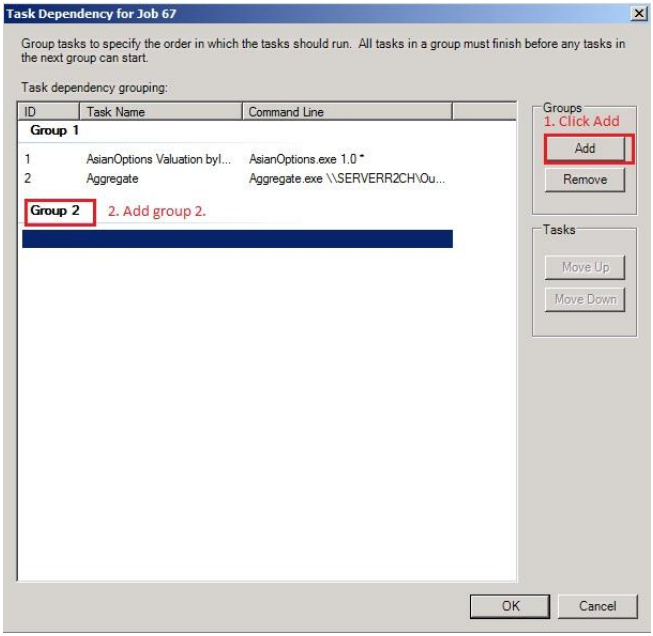



*Figure 3-30 Add a new task group*

Select the Aggregate task, click **Move Down** in the **Tasks** group box, and the Aggregate task would be moved to the second group (Group 2). In this way, the first group of tasks must end before the Aggregate task can begin. In other words, the Aggregate task has a dependency on the AsianOptions task, as shown in Figure 3-31.

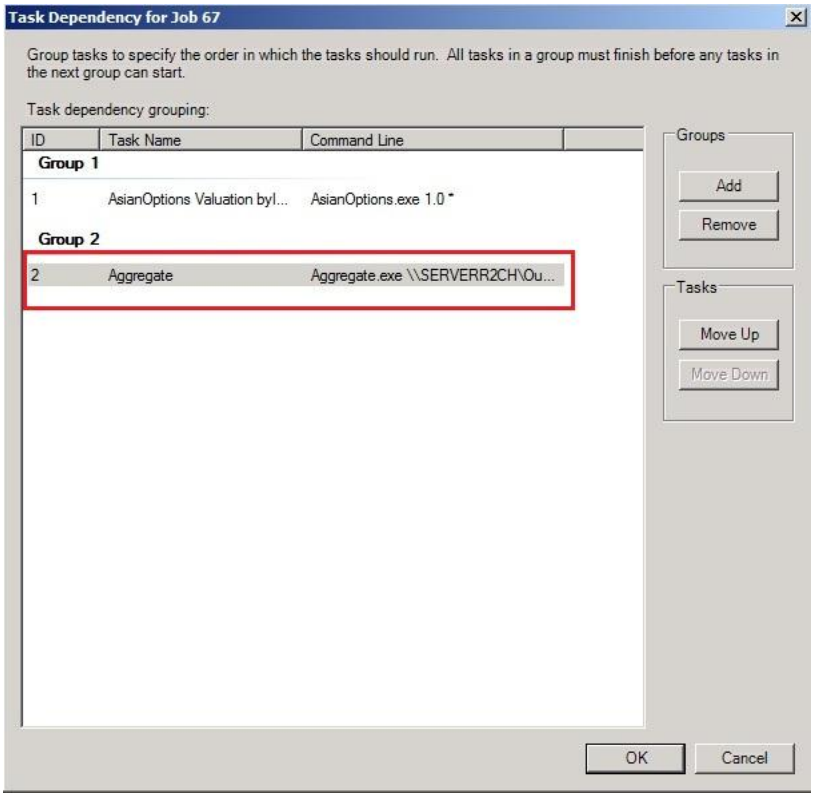



*Figure 3-31 Set task dependency between Aggregate and AsianOptions*

7. Click **OK** to return to the **Copy Job** window, as shown in Figure 3-32.

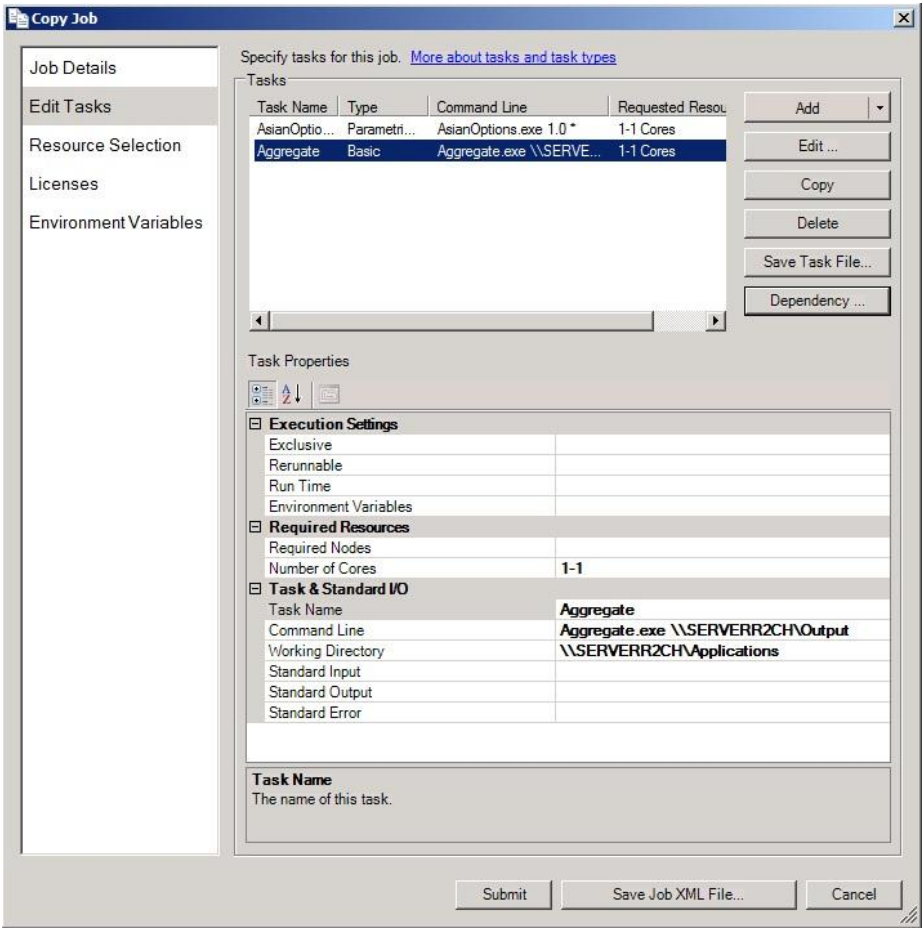



*Figure 3-32 Copy Job window*

8. Click **Submit**. After the job runs, at the "Active" jobs in the "Job Management" in the navigation bar, select the job. It would show the two tasks in the task list below. The "AsianOption" task is running, and the "Aggregate" task is in the queued state, as shown in Figure 3-33.

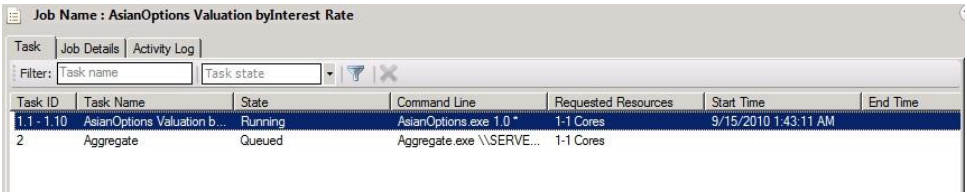



*Figure 3-33 Task flow – the "AsianOptions" task is runing while the "Aggregate" task is queued*

9. After the AsianOptions task is finished, the Aggregate task starts to run. When the job completes, the two tasks are in the Finished state, as shown in Figure 3-34.

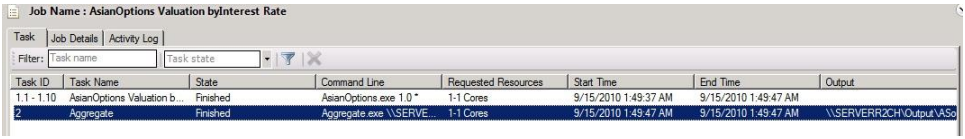



*Figure 3-34 Task flow is complete*

10. In the task list view, double-click the "Aggregate" task. The View Job window is displayed. We can see that the "Aggregate" task output contains all the calculation results by AsianOptions by scrolling down the text in the output, as shown in Figure 3-35.

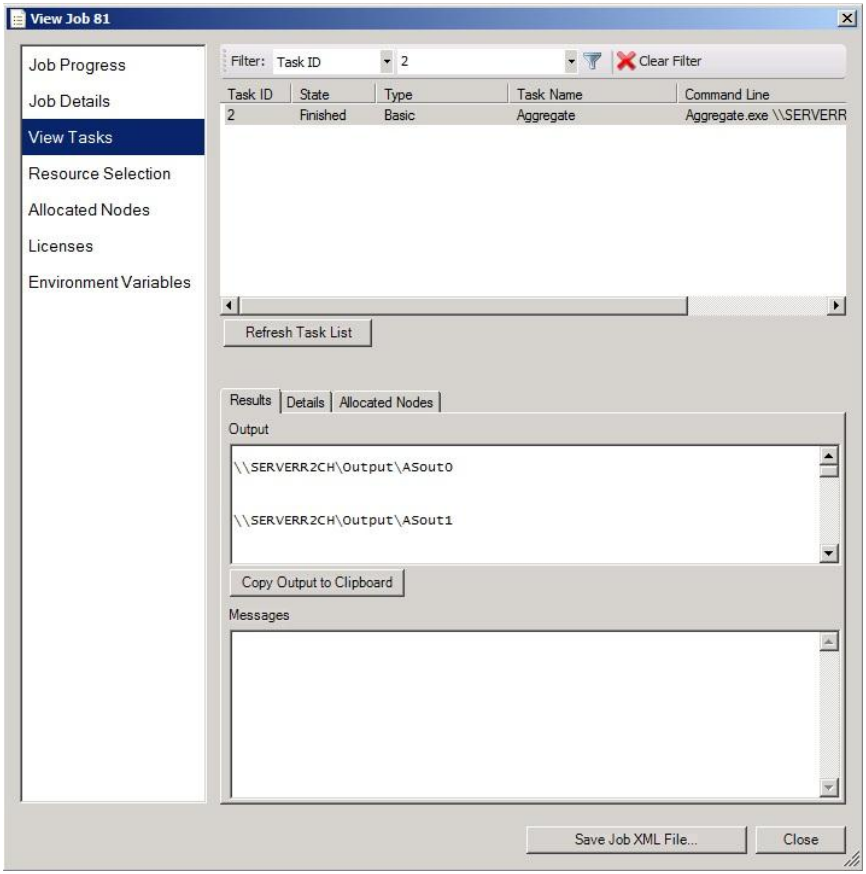



*Figure 3-35 Task output of the Aggregate task*

#### 3.4.2.2 Using the command line and PowerShell

*Table 3-3*

| | |
|---|---|
| Command line | job new<br><br>job add <ID> /name:compute /workdir:\\<headnode>\Applications /stdout:\\<headnode>\Output\ASout.txt /parametric:0-9 AsianOptions.exe 1.0*<br><br>job add <ID> /name:aggregate /depend:compute aggregate.exe \\<headnode>\Output (Replace <headnode> with your head node name) |
| PowerShell | New-HpcJob<br><br>Add-HpcTask –JobId <ID> -Name Compute –WorkDir \\<headnode>\Applications -Stdout \\<headnode>\Output\ASout.txt -CommandLine "AsianOptions.exe 1.0*"<br><br>Add-HpcTask –JobId <ID> -Name Aggregate –Depend Compute –WorkDir \\<headnode>\Applications -CommandLine aggregate.exe -Stdout \\<headnode>\Output<br><br>Submit-HpcJob –Id <ID> (Replace <headnode> with your head node name) |

## 3.5 Appendix

### 3.5.1 Job Properties

Job Properties are listed in Table 3-4.

*Table 3-4 Job properties*

| **Property** | **Description** | **Command line option** | **PowerShell option** |
|---|---|---|---|
| Job Name | The name of this job. | `/jobname` | `-JobName` |
| Project | The name of the project of which this job is a part. | `/project` | `-Project` |
| Template | The name of the Job Template against which this job is being submitted. | `/jobtemplate` | `-TemplateName` |
| Priority | The priority of the job; it can be one of five values: Lowest, BelowNormal, Normal, AboveNormal, or Highest. | `/priority` | `-Priority` |

| Property | Description | Command line option | PowerShell option |
|---|---|---|---|
| Run Time | The hard limit (dd:hh:mm) on the amount of time this job will be allowed to run. If the task is still running after the specified run time has been reached, it will be automatically cancelled by the scheduler. | `/runtime` | `-Runtime` |
| Number of Processors | The number of processors required by this job; the job will run on at least the Minimum and no more than the Maximum. If set to *, the scheduler will automatically calculate these values based on the job's tasks. | `/numcores` | `-NumCores` |
| Number of Sockets | The number of sockets required by this job; the job will run on at least the Minimum and no more than the Maximum. If set to *, the scheduler will automatically calculate these values based on the job's tasks. | `/numsockets` | `/NumSockets` |
| Number of Nodes | The number of nodes required by this job; the job will run on at least the Minimum and no more than the Maximum. If set to *, the scheduler will automatically calculate these values based on the job's tasks. | `/numnodes` | `-NumNodes` |
| Exclusive | If set to True, no other jobs can be run on a compute node at the same time as this job. | `/exclusive` | `-Exclusive` |

| Property | Description | Command line option | PowerShell option |
|---|---|---|---|
| Requested Nodes | A list of nodes; the job can be run only on nodes that are in this list. | `/requestednodes` | `-RequestedNodes` |
| Node Groups | A list of node groups; the job can be run only on nodes that are in all of these groups. | `/nodegroup` | `-NodeGroup` |
| Minimum Memory | The Minimum amount of memory (in MB) that must be present on any nodes on which this job is run. | `/memorypernode` | `-MinMemoryPerNode` |
| Minimum Processor Count | The Minimum number of processors that must be present on any nodes on which this job is run. | `/corespernode` | `-MinCoresPerNode` |
| Node Ordering | The ordering to use when selecting nodes for the job. For example, ordering by "More Memory" will allocate the available nodes with the largest amount of memory. | `/orderby` | `-OrderBy` |
| Fail On Task Failure | If set to True, failure of any task in the job will cause the entire job to fail immediately. | `/failontaskfailure` | `-FailOnTaskFailure` |
| Run Until Cancelled | If set to True, this job will run until it is cancelled or its run time is expired. It will not stop when there are no tasks left within it. | `/rununtilcanceled` | `-RunUntilCanceled` |
| Licenses | A list of licenses that are required for the job. This list can be interpreted by a Job Activation Filter, but it is not used by the Windows HPC Server 2008 Job Scheduler. | `/license` | `-License` |

## 3.5.2 Task Properties

Task Properties are listed in Table 3-5.

*Table 3-5 Task properties*

| Property | Description | Command line option | PowerShell option |
|---|---|---|---|
| Task Name | The name of this task. | `/name` | `-Name` |
| Command Line | The command line that will be executed by this task. | N/A | `-CommandLine` |
| Working Directory | The working directory to be used during execution of this task. | `/WorkDir` | `-WorkDir` |
| Standard Input | The path (relative to the working directory) to the file from which the `stdin` of this task should be read. | `/stdin` | `-StdIn` |
| Standard Output | The path (relative to the working directory) to the file to which the `stdout` of this task should be written. | `/stdout` | `-Stdout` |
| Standard Error | The path (relative to the working directory) to the file to which the `stderr` of this task should be written. | `/stderr` | `-Stderr` |
| Number of Processors | The number of processors required by this task; the task will run on at least the Minimum and no more than the Maximum. | `/numcores` | `-NumCores` |
| Number of Sockets | The number of sockets required by this task; the task will run on at least the Minimum and no more than the Maximum. | `/numsockets` | `-NumSockets` |
| Number of Nodes | The number of nodes required by this task; the task will run on at least the Minimum and no more than the Maximum. | `/numnodes` | `-NumNodes` |

| Property | Description | Command line option | PowerShell option |
|---|---|---|---|
| Required Nodes | Lists the nodes that must be assigned to this task and its job in order to run. | `/requirednodes` | `-RequiredNodes` |
| Run Time | The hard limit (dd:hh:mm) on the amount of time this task will be allowed to run. If the task is still running after the specified run time has been reached, it will be automatically cancelled by the scheduler. | `/runtime` | `-Runtime` |
| Exclusive | If set to True, no other tasks can be run on a compute node at the same time as this task. | `/exclusive` | `-Exclusive` |
| Rerunnable | If the task runs and fails, setting Rerunnable to True means the scheduler will attempt to rerun the task. If Rerunnable is set to False, the task will fail after the first run attempt fails. | `/rerunnable` | `-Rerunnable` |
| Environment Variables | Specifies the environment variables to set in the task's run-time environment. Environment variables must be separated by commas in the format *name1=value1.* | `/env` | `-Env` |
| Sweep Start Index | The starting index for the sweep. | `/parametric:<start>` | `-Start` |
| Sweep End Index | The ending index for the sweep. | `/parametric:<start>-<end>` | `-End` |
| Sweep Increment | The amount to increment the sweep index at each step of the sweep. | `/parametric:<start>-<end>:step` | `-Step` |

## 3.6 Working with Job Files and Submitting Jobs

Jobs can be saved as Job Descriptor files by clicking **Save Job As** in the **Create New Job Wizard**. Job Description files can be exchanged in an XML format that allows them to be reused, edited, and generated by other applications. It's a good idea to save any recurring job or task as a Job Descriptor file. Not only do the templates facilitate job or task re-creation, they also support job and task submission using the **Command Prompt** window.

*Job submission* means placing items into the job queue. Jobs can be created and submitted interactively or using a job template. To create and submit the job interactively, fill in the pages in the **Create New Job Wizard**, and then click **Submit** when the job is fully described. To submit jobs from Job Descriptor files, click **Create New Job from Description File**. From there, select the appropriate Job Descriptor file, modify its properties (if necessary), and then click **Submit**.

## 3.7 Controlling Jobs Using Filters

Windows HPC Server 2008 includes several features for job submission and execution. Jobs can consist of simple tasks or can include comprehensive feature sets. Administrators can control jobs using execution filters that impose a set of conditions before the job begins. Two types of filters are available:

- Job submission filters
- Job execution filters

Administrators can use these filter sets together to verify that jobs meet specific requirements before they pass through the system. Examples of the conditions for these filters include:

- **Project validation.** This condition verifies that the project name is that of a valid project and that the user is a member of the project.
- **Mandatory policy.** This condition ensures that run times are not set to **infinite**, which would clog the cluster, and that the job does not exceed the user's resource allocation limit.
- **Usage time.** Usage time helps ensure that the user's time allocations are not exceeded. Unlike the mandatory policy, this filter limits jobs to the overall time allocations that users have for all possible jobs.

In addition, job execution filters can help to ensure that jobs meet licensing conditions before they run. Filters are a powerful job control feature that should be part of every Windows HPC Server 2008 implementation. Job Scheduler invokes the filters by parsing the job file, which contains all the job properties. The exit code of the filter program tells Job Scheduler what to do. Three types of values are possible:

- 0: It is okay to submit the job without changing its terms.
- 1: It is okay to submit the job with changed terms.
- If any other value appears, the job is rejected.

## 3.8 Security Considerations for Jobs and Tasks

Windows HPC Server 2008 uses Windows-based security mechanisms within the cluster context. Jobs are executed with the submitting user's credentials. These credentials are stored in encrypted format on the local computer by Job Scheduler, and only the head node has access to the decryption key.

The first time a user submits a job, the system prompts for credentials in the form of a user name and password. At this point, the credentials can be stored in the credential cache of the submission computer. When in transit, credentials are protected using a secure Microsoft .NET remoting channel, then secured using the Windows Data Protection API and stored in the job database. When a job runs, the head node decrypts the credentials and uses another secure .NET remoting channel to pass them along to compute nodes, where they are used to create a token and then erased. All tasks are performed using this token, which does not include the explicit credentials. When jobs are complete, the head node deletes the credentials from the job database.

When the same user submits the same job for execution again, no credentials are requested because they are already cached on the local computer running Job Scheduler. This feature simplifies resubmission and provides integrated security for credentials throughout the job life cycle, as shown in Figure 3-36.

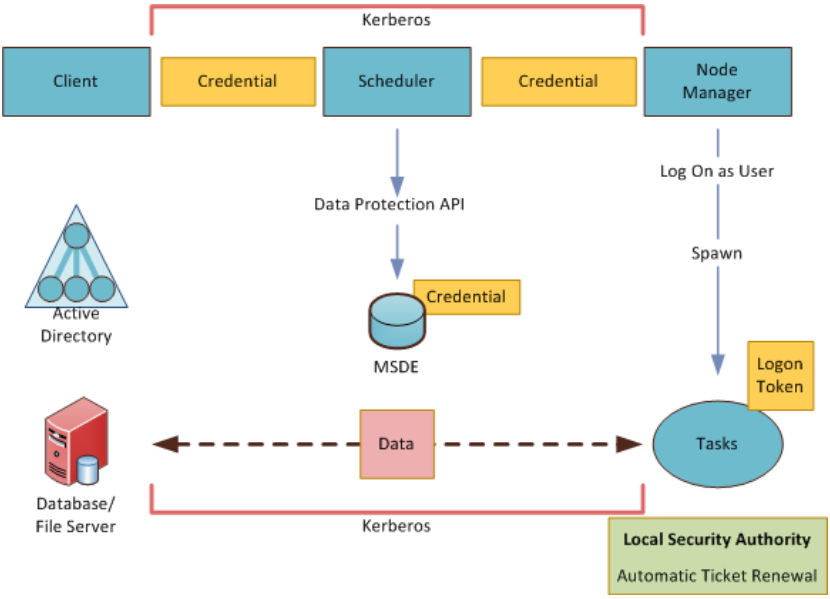



*Figure 3-36 The end-to-end Windows HPC Server security model*

# Chapter 4 Developing HPC SOA Applications

In high-performance computing, there is a well-known type of problem called the "embarrassing parallel" problem. The problem involves work that requires no synchronization among the individual tasks, nor does it require static decomposition of the problem domain. Developers can easily subdivide the work into independent sub-problems and distribute the processing of these problems onto multiple nodes, thereby meeting users' acceleration needs.

The HPC SOA programming model is well-suited to handle the "embarrassing parallel" problem. With the HPC SOA framework, developers can easily encapsulate the calculation logic into a service. Using the SOA Session API, developers can easily code an application that runs multiple service instances on the cluster, concurrently invokes these services remotely, and aggregates the calculation results.

In this chapter, we look at an "embarrassing parallel" problem—using the Monte Carlo algorithm for pricing an Asian Option. In the last chapter, we used the Asian Option pricing application as a black-box. In this chapter, we are going to describe, in-depth, how to use the SOA framework and Session API to parallelize the Asian Option Pricing application. In the last chapter, the pricing application is delivered in the form of a console application. In this chapter, we are going to deliver the client application using one of the most common user interfaces, Microsoft Excel. Through this exercise, the reader can learn how to use the Visual Studio Tools for Office (VSTO) to invoke the HPC Services.

Chapter Highlights:

- Overview of Monte Carlo algorithms used for Asian Option pricing
- How to create HPC service and applications that can run on an HPC Server R2 cluster
- How to create a VSTO-Excel client application that uses the services on an HPC Server R2 cluster

## 4.1 Sequential and Parallel Monte Carlo Simulation for Asian Option Pricing

### 4.1.1 Asian Option pricing simulation algorithm

An Asian option (or average value option) is a special type of option contract. For Asian options, the payoff is determined by the average price of the underlying asset over some preset period of time. We are going to use the Monte Carlo simulation method, a technique commonly used in determining the price of a derivative product.

To compute the price of the option, Monte Carlo simulation uses the risk neutral theory. In a risk neutral world, the algorithm first generates the asset's price paths, uses those paths to compute the expected payoff, and then discounts the expected value of the asset payoff using the risk neutral interest rate.

Consider a security product, say a stock, and its payoff within **n** periods after purchase. Assume that the interest rate is a constant **R**, the initial price of the stock is **initial**. Within the valid period of the option, the stock price may go up from $S_0$ to $S_0 * u$, or go down from $S_0$ to $S0 * d$, where $u > 1$, $d < 1$. When the price goes up, its ratio of increase is $u-1$; when the price goes down, the ratio of decrease is $1-d$. The risk neutral up and down probability is $P_{up}=(R-d)/(u-d)$, $P_{down}=(u-R)/(u-d)$. We can use the following steps to price an Asian Option:

1. Generate sample paths for a stock in the risk-neutral world using a pseudo-random number generator, as shown in Figure 4-1. If the random number Rand > $P_{down}$, the price of the stock goes up; if Rand <= $P_{down}$, the price goes down
2. Calculate the payoff: if Rand > $P_{down}$, then payoff += $S_0 \times u$; otherwise, payoff += $S_0 \times d$.
3. Repeat steps (1) and (2) multiple times, calculate the average pay-off.
4. Use a risk-neutral rate to discount the Asian Option pay-off: the approximate price of the Asian Option is price = payoff / $R^n$.

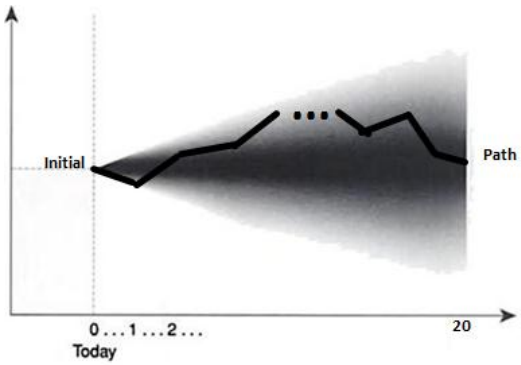



*Figure 4-1 Risk-neutral path space*

### 4.1.2 Traditional sequential simulation code

The Monte Carlo program in C# is shown in Code 4-1, where:

- Initial: initial price of the stock
- Exercise: exercise or strike price
- Up: risk-neutral up growth
- Down: risk-neutral down growth
- Interest: interest rate
- Periods: time to maturity
- Runs: Monte Carlo simulation iterations.

```
double PriceAsianOptions(double initial, double exercise, double up,
                                 double down, double interest, int
periods, int runs)
        {
            double[] pricePath = new double[periods + 1];

            // Risk-neutral probabilities
            double piup = (interest - down) / (up - down);
            double pidown = 1 - piup;

            double temp = 0.0;

            Random rand = new Random();
            double priceAverage = 0.0;
            double callPayOff = 0.0;

            for (int index = 0; index < runs; index++)
            {
                // Generate Path
                double sumPricePath = initial;
```

```
            for (int i = 1; i <= periods; i++)
            {
                pricePath[0] = initial;
                double rn = rand.NextDouble();

                if (rn > pidown)
                {
                    pricePath[i] = pricePath[i - 1] * up;
                }
                else
                {
                    pricePath[i] = pricePath[i - 1] * down;
                }
                sumPricePath += pricePath[i];
            }

            priceAverage = sumPricePath / (periods + 1);
            callPayOff = Math.Max(priceAverage - exercise, 0);

            temp += callPayOff;
        }
        // calculate average payoff
        temp /= runs;
        // return the price of the asian option
        // price = discounted expected value of the asset payoff
        return temp / Math.Pow(interest, periods);
    }
}
```

*Code 4-1 Asian Option pricing function*

To ensure accuracy, we are making 100 invocations of the pricing function, as shown in Code 4-2.

```
int count = 0;

for (int i = 0; i < 100; i++)

{

    double price = PriceAsianOptions(initial, exercise, up, down, interest,
periods, runs);




    min = Math.Min(min, price);

    max = Math.Max(max, price);


    sumPrice += price;

    sumSquarePrice += price * price;

    count++;


    stdDev = Math.Sqrt(sumSquarePrice - sumPrice * sumPrice / count) / ((count
== 1) ? 1 : count - 1);

   stdErr = stdDev / Math.Sqrt(count);

}
```

*Code 4-2 Invoking the Asian Option pricing function*

### 4.1.3 Parallelizing the Asian Option pricing algorithm

The loop in the last section is inter-iteration dependent, meaning that the values of the variables min and max at iteration i depend on their value in iteration i – 1. As such, the loop would need to be executed sequentially to ensure correctness. However, this dependency can be easily removed through a simple code transformation. To do so, we put the return value of **PriceAsianOptions()** into an array price[], as shown in Code 4-3.

```
double[] price = new double[100];


for (int i = 0; i < 100; i++)

{

    price[i] = PriceAsianOptions(initial, exercise, up, down, interest,
periods, runs);

}


int count = 0;


for (int i = 0; i < 100; i++)

{

    min = Math.Min(min, price[i]);
```

```
    max = Math.Max(max, price[i]);


    sumPrice += price[i];

    sumSquarePrice += price[i] * price[i];

    count++;


    stdDev = Math.Sqrt(sumSquarePrice - sumPrice * sumPrice / count) / ((count
== 1) ? 1 : count - 1);

   stdErr = stdDev / Math.Sqrt(count);

}
```

*Code 4-3 Loop transformation – making the original loop inter-iteration independent*

The marked loop is inter-iteration independent and therefore can be parallelized. To realize the maximum scalability, we deploy the Asian Option calculation logic to compute nodes and then concurrently distribute the 100 invocations onto multiple nodes, leading to a reduction in processing time.

## 4.1.4 Steps of parallelization using the HPC SOA model

We are going to use the HPC SOA model to implement the parallel simulation for Asian Option pricing. The steps are:

1. Create the Asian Option pricing service.
2. Create the client application program.

### 4.1.4.1 System Requirements

You must have the following items to complete this lab:

- Microsoft HPC Pack 2008 R2
- Microsoft HPC Pack 2008 R2 SDK
- Visual Studio 2008 SP1 and Microsoft Office 2007 SP1
- Microsoft Visual Studio Tools for Office (VSTO) 3.0

**Watch Out**

You also need to be granted permissions to access compute and head nodes to perform a successful deployment on the HPC Cluster.

### 4.1.4.2 Example code

The example code can be found in Chapter 4 of the Sample code.

## 4.2 Create the HPC SOA Service

### 4.2.1 Create a service

In this task, you will create a simple WCF Web service to simulate the calculation of an Asian stock option, which will then be distributed on the compute nodes of the HPC 2008 R2 Cluster.

1. Click **Start**, click **All Programs**, click **Microsoft Visual Studio 2008**, and then click **Microsoft Visual Studio 2008**.
2. On the **File** menu, point to **New**, and then click **Project**.
3. In the **New Project** dialog box, make sure that .NET Framework 3.5 is selected. Set the project type by clicking **Visual C#**, and set the template by selecting **WCF Service Application**. You can set the location to **Chapter 4\Labs\IntroToParallelComputing\Ex01-CreatingParallelComputingApplication\begin**, which is the provided folder, as shown in Figure 4-2.
4. Change the **Name** to **AsianOptionsService** and the **Solution Name** to **Begin**, and then click **OK**.

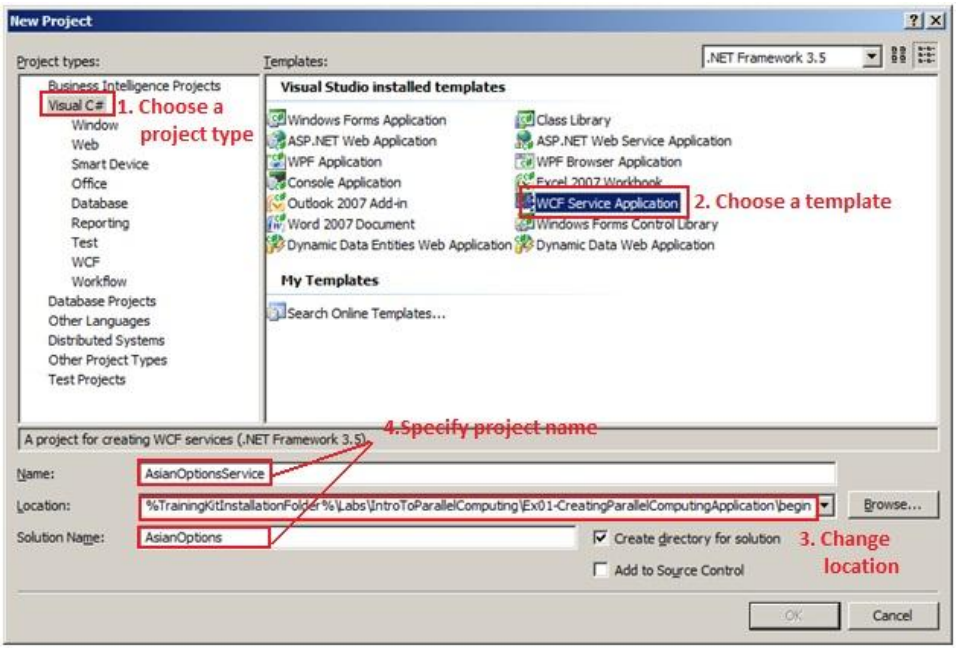



*Figure 4-2 Add the AsianOptionsService project*

5. Remove the IService1.cs and Service1.svc files from the project. To do this, right-click each file on the Solution Explorer, click **Delete**, and confirm deletion on the confirmation dialog.
6. Add a new service to the project. To do this, right-click the **AsianOptionsService** project on Solution Explorer, click **Add**, and then click **New Item**. On the **Add New Item** dialog, select **WCF Service**, change the name to **AsianOptions.svc**, and then click **Add**.

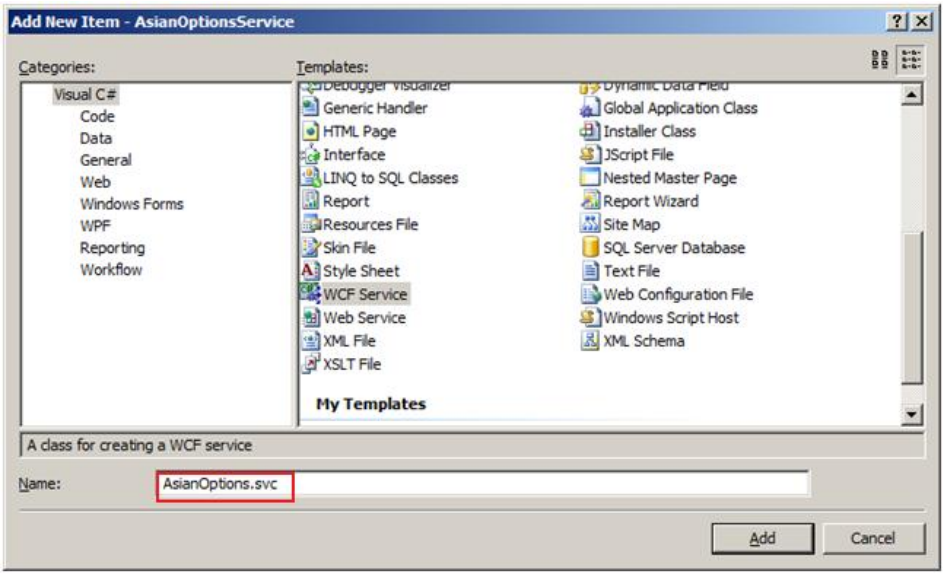



*Figure 4-3 Add the AsianOptions Service*

7. Replace the code of the **IAsianOptions** interface with the following code, as shown in Code 4-4.

   (Code Snippet – *IntroToParallelComputing - IAsianOptions interface*)

C#

```
namespace AsianOptionsService
{
    using System;
    using System.Collections.Generic;
    using System.Linq;
    using System.Runtime.Serialization;
    using System.ServiceModel;
    using System.Text;

    [ServiceContract]
    public interface IAsianOptions
    {
        [OperationContract]
        double PriceAsianOptions(double initial, double exercise,
double up, double down, double interest, int periods, int runs);
    }
}
```

*Code 4-4 AsianOptionsService service interface*

8. Implement the interface on the **AsianOptions** service. To do this, right-click **AsianOptions.svc** on Solution Explorer and select **View Code**. Next, replace the existing code with the code shown in Code 4-5.

(Code Snippet – *IntroToParallelComputing – AsianOptions service implementation*)

C#

```
namespace AsianOptionsService
{
    using System;
    using System.Collections.Generic;
    using System.Linq;
    using System.Runtime.Serialization;
    using System.ServiceModel;
    using System.Text;

    [ServiceBehavior(ConcurrencyMode = ConcurrencyMode.Multiple)]
    public class AsianOptions : IAsianOptions
    {
        public double PriceAsianOptions(double initial, double
exercise, double up, double down, double interest, int periods, int
runs)
        {
            double[] pricePath = new double[periods + 1];

            double piup = (interest - down) / (up - down);
            double pidown = 1 - piup;

            double temp = 0.0;

            Random rand = new Random();
            double priceAverage = 0.0;
            double callPayOff = 0.0;

            for (int index = 0; index < runs; index++)
            {
                double sumPricePath = initial;

                for (int i = 1; i <= periods; i++)
                {
                    pricePath[0] = initial;
                    double rn = rand.NextDouble();

                    if (rn > pidown)
```

```
                    {
                        pricePath[i] = pricePath[i - 1] * up;
                    }
                    else
                    {
                        pricePath[i] = pricePath[i - 1] * down;
                    }

                    sumPricePath += pricePath[i];
                }

                priceAverage = sumPricePath / (periods + 1);
                callPayOff = Math.Max(priceAverage - exercise, 0);

                temp += callPayOff;
            }

            return (temp / Math.Pow(interest, periods)) / runs;
        }
    }
}
```

*Code 4-5 Service Implementation*

9. Finally, you need to add a configuration file for the service. To do this, right-click **AsianOptionsService** on Solution Explorer and click **Add**, and then click **New Item**. On the **Add New Item** dialog, select **Web Configuration File** and change the name to **AsianOptionsService.config**, as shown in Figure 4-4.

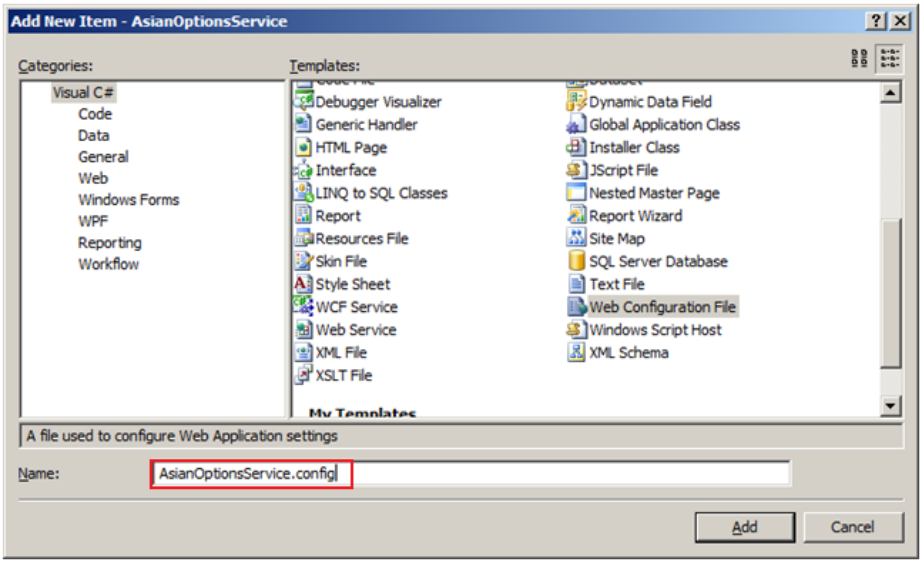



*Figure 4-4 Add the AsianOptionsService Configuration File*

10. Replace the content of the AsianOptionsService.config file with the code shown in Code 4-6.

XML

```
<?xml version="1.0" encoding="utf-8" ?>
<configuration>

  <configSections>

    <!--Register service's custom configruation sections and group-->
    <sectionGroup name="microsoft.Hpc.Session.ServiceRegistration"

type="Microsoft.Hpc.Scheduler.Session.Configuration.ServiceRegistration
, Microsoft.Hpc.Scheduler.Session, Version=2.0.0.0, Culture=neutral,
PublicKeyToken=31bf3856ad364e35">
      <section name="service"

type="Microsoft.Hpc.Scheduler.Session.Configuration.ServiceConfiguratio
n, Microsoft.Hpc.Scheduler.Session, Version=2.0.0.0, Culture=neutral,
PublicKeyToken=31bf3856ad364e35"
               allowDefinition="Everywhere"
               allowExeDefinition="MachineToApplication"
               />
    </sectionGroup>

  </configSections>


  <microsoft.Hpc.Session.ServiceRegistration>
    <service assembly="c:\AsianOptionsService\AsianOptionsService.dll"
/>
  </microsoft.Hpc.Session.ServiceRegistration>
```

```
  <system.serviceModel>
    <bindings>
      <netTcpBinding>
        <!--binding used by broker's front end service with
security-->
        <binding name="Microsoft.Hpc.SecureNetTcpBrokerBinding"
portSharingEnabled="true" closeTimeout="0:00:05"
maxConnections="1000">
          <security mode="Transport">
```

```
            <transport clientCredentialType="Windows"
protectionLevel="None"></transport>
          </security>
        </binding>
      </netTcpBinding>
    </bindings>
  </system.serviceModel>


</configuration>
```

*Code 4-6 Service Configuration File*

> **Note:** The **<service>** element in the **microsoft.Hpc.Session.ServiceRegistration** section specifies the path where the service assembly will be deployed on the compute nodes. We will return to this on the Exercise Verification.

11. Press CTRL+SHIFT+B to build the solution.

## 4.2.2 Create the Excel client application

There are many ways to create the client application. Users can consider using a command line interface or a graphical user interface. Here we will use the Excel interface, because it's the killer application in the Finance industry. Almost all traders are familiar with Excel. We're creating an Excel application to use AsianOptionsService with the following steps:

1. Add a new Excel Workbook application to the solution. To do this, right-click **Begin** solution on Solution Explorer and click **Add**, and then click **New Project**. On the **Add New Project** dialog, shown in Figure 4-5, select the **Office** project type on the left pane and **Excel 2007 Workbook** project template on the right pane. Change the Name to **AsianOptions** and the Location to **%TrainingKitInstallationFolder%\Labs\IntroToParallelComputing\Ex01-CreatingParallelComputingApplication\begin**, and then click **OK**.

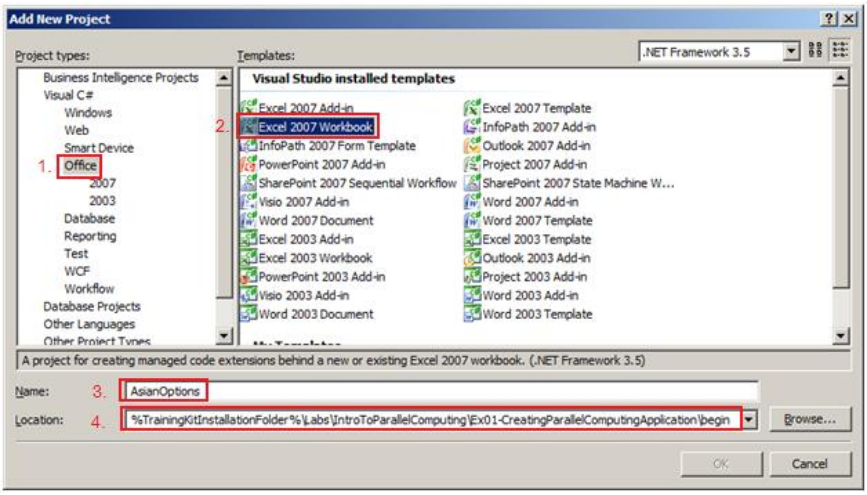



*Figure 4-5 Add the AsianOptions client application project*

On the Visual Studio Tools for Office **Project Wizard** dialog, select the **Copy an existing document** option and browse for the **Chapter 4\Labs\IntroToParallelComputing\Assets\AsianOptionPricingCSharpBehind.xlsx** file, as shown in Figure 4-6, and then click **OK**.

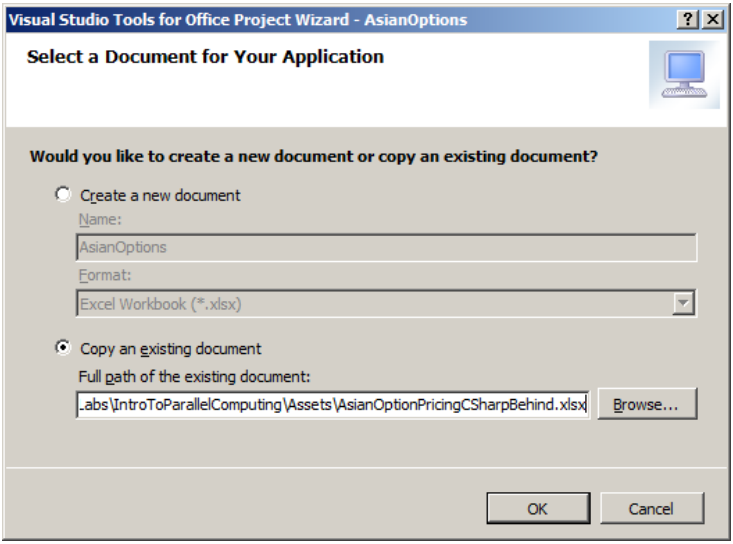



*Figure 4-6 Add the Excel Workbook*

2. Add a service reference to the AsianOptions service. To do this, on Solution Explorer, right-click the **AsianOptions** project and select **Add Service Reference...** On the **Add Service Reference** dialog, shown in Figure 4-7, change the namespace to **AsianOptionsService**, click **Discover**, and select the AsianOptions.svc service.

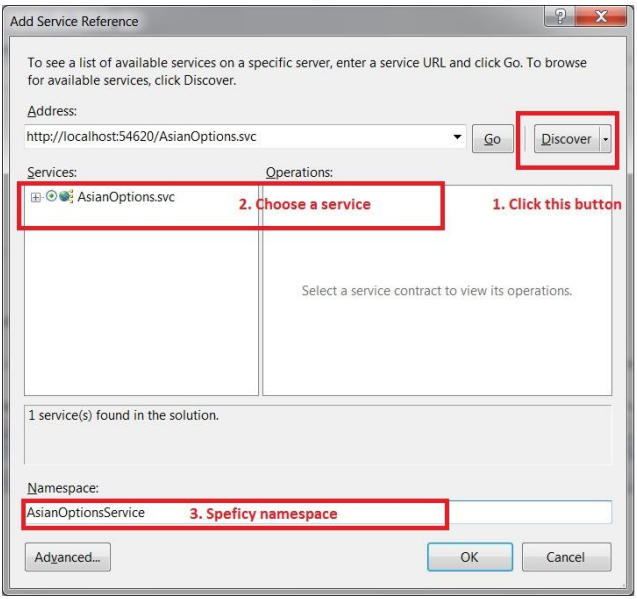



*Figure 4-7 Add a service reference*

3. Click **Advanced**, and on the **Service Reference Settings** dialog, shown in Figure 4-8, select **Generate asynchronous operations** and click **OK**.

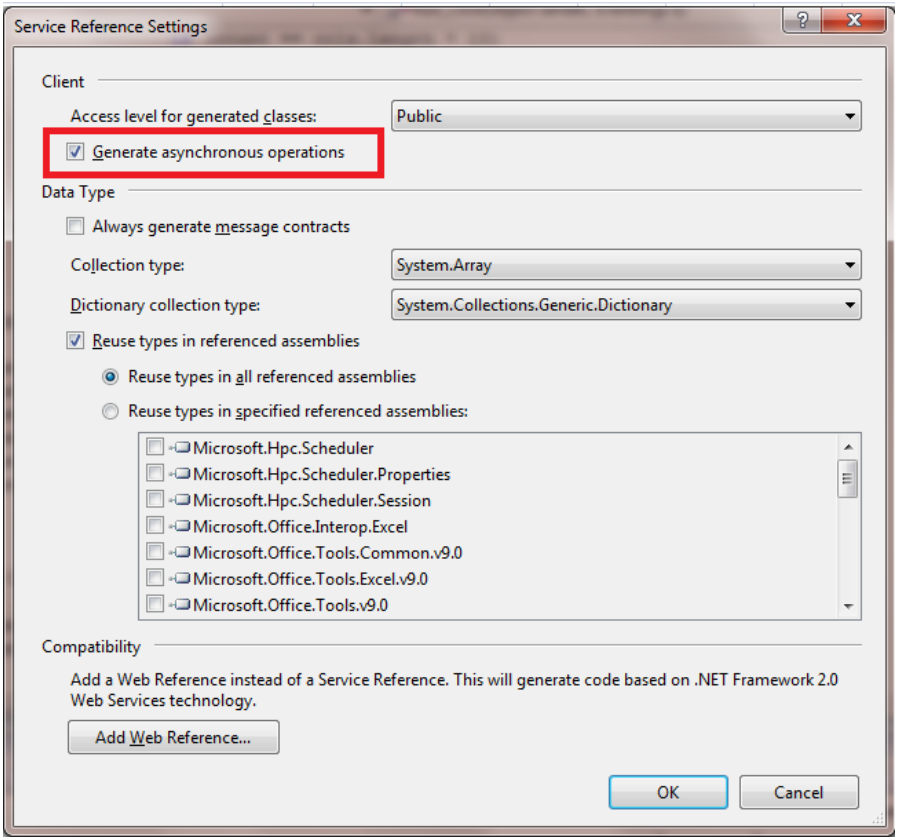



*Figure 4-8 Configure the AsianOptionsService service reference*

4. Add references to HPC 2008 R2 SDK. To do this, on the Solution Explorer pane, right-click the **AsianOptions** project and select **Add Reference...** On the **Add Reference** dialog, click the **Browse** tab, navigate to **%HPCPack2008R2SDK%\Bin**, and select the **Microsoft.Hpc.Scheduler.dll**, **Microsoft.Hpc.Scheduler.Properties.dll**, and **Microsoft.Hpc.Scheduler.Session.dll** assemblies, as shown in Figure 4-9, and then click **OK**.

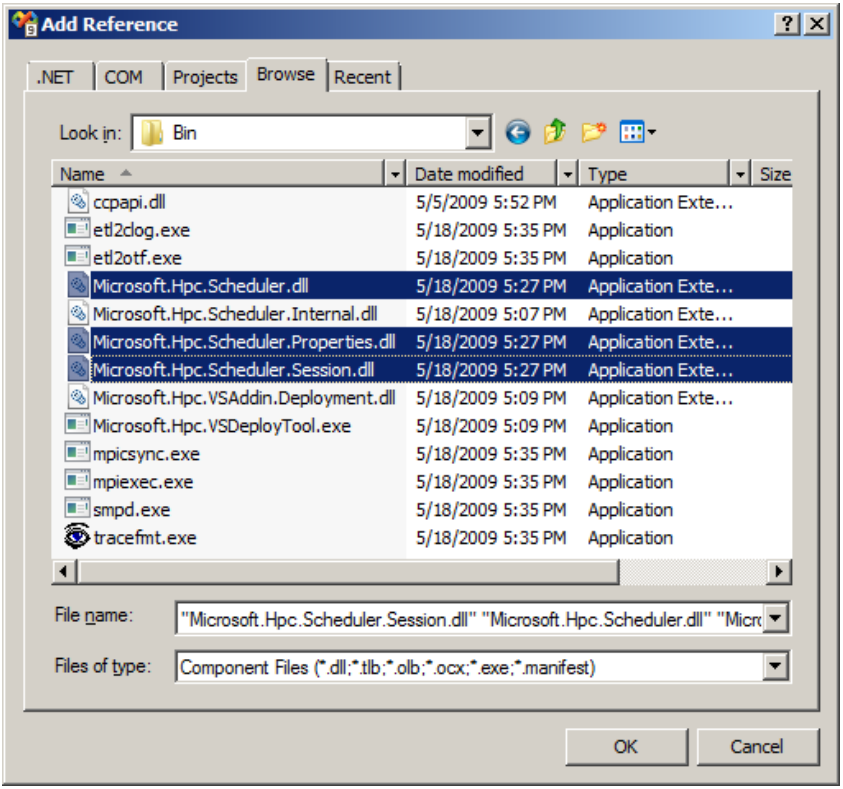



*Figure 4-9 Add Reference to HPC 2008 R2 SDK assemblies*

> **Help**
>
> **%HPCPack2008R2SDK%** is the installation folder you selected during the Microsoft HPC Pack 2008 SDK installation process.

5. Set the names of the buttons on the **Sheet1** Excel sheet. To do this, right-click the **Sheet1.cs** file on Solution Explorer and select the **View Designer** option. Once the Excel spreadsheet is shown, set the following properties to the two existing buttons, as shown in Figure 4-10.

    a. For the top button, set:

        i. **Name:** Run

        ii. **Text:** Run

    b. For the bottom button, set:

        i. **Name:** Clear

        ii. **Text:** Clear

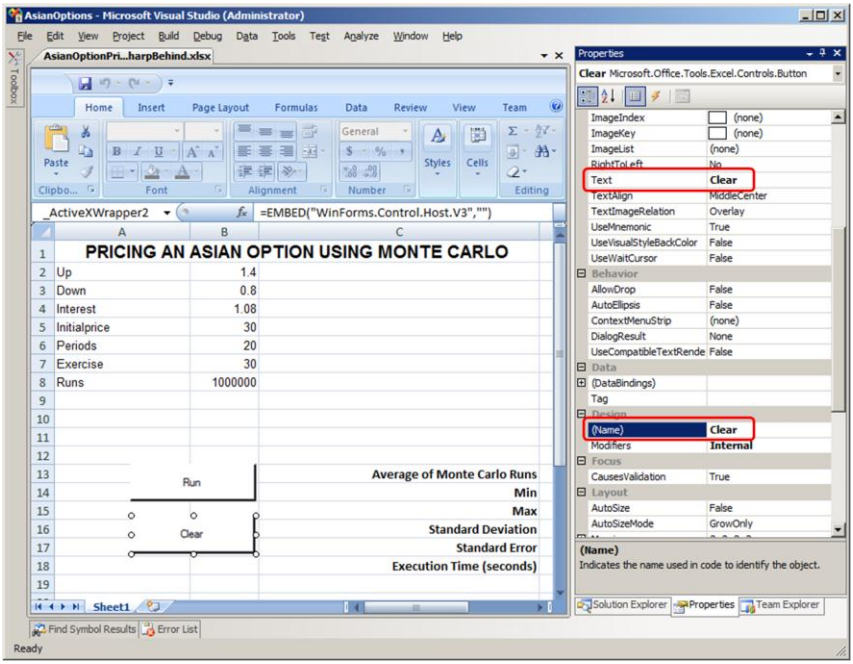



*Figure 4-10 Configure Sheet1 button settings*

6. Right-click **Sheet1** in Solution Explorer and select **View Code** to open the code behind it. When the text editor displays the code behind **Sheet1**, add the namespace directive statements, as shown in Code 4-7.

   (Code Snippet – *IntroToParallelComputing - Sheet1 namespace references*)

```
C#
```

```
using System.Threading;
using System.Diagnostics;
using AsianOptions.AsianOptionsService;
using Microsoft.Hpc.Scheduler.Session;
using System.ServiceModel;
```

*Code 4-7 Declaring namespaces*

7. Add the code (in bold) to the **Sheet1** class, as shown in Code 4-8.

   (Code Snippet – *IntroToParallelComputing - Sheet1 Range fields*)

```
C#
```

```
public partial class Sheet1
{
    private static Excel.Range rngUp, rngDown, rngInitial, rngExercise,
rngInterest, rngPeriods, rngRuns, rngAsianCallValue, rngHeadNode;

    ...
```

*Code 4-8 Variables denoting Asian Option parameter ranges in the Excel workbook*

8. Initialize the fields just created on the **Sheet1** class. To do this, add the following code (in bold) on the **Sheet1_Startup** method, as shown in Code 4-9.

   (Code Snippet – *IntroToParallelComputing - Sheet1 Sheet1_Startup implementation*)

**C#**

```
private void Sheet1_Startup(object sender, System.EventArgs e)
{
    rngUp = this.Range["B2", this.missing];
    rngDown = this.Range["B3", this.missing];
    rngInterest = this.Range["B4", this.missing];
    rngInitial = this.Range["B5", this.missing];
    rngPeriods = this.Range["B6", this.missing];
    rngExercise = this.Range["B7", this.missing];
    rngRuns = this.Range["B8", this.missing];
     rngAsianCallValue = this.Range["B9", this.missing];
     rngHeadNode = this.Range["B10", this.missing];
}
```

*Code 4-9 Initializing Sheet1*

9. Add handling for the buttons on the Sheet1 Excel worksheet. To do this, add the methods shown in Code 4-10 to the **Sheet1** class**.**

   (Code Snippet – *IntroToParallelComputing - Sheet1 buttons click handling implementation*)

**C#**

```
private void Run_Click(object sender, EventArgs e)
{
            double initial = (double)rngInitial.Value2;
            double exercise = (double)rngInitial.Value2;
            double up = (double)rngUp.Value2;
            double down = (double)rngDown.Value2;
            double interest = (double)rngInterest.Value2;
            int periods = Convert.ToInt32(rngPeriods.Value2);
            int runs = Convert.ToInt32(rngRuns.Value2);

            double sumPrice = 0.0;
            double sumSquarePrice = 0.0;
            double min = double.MaxValue;
            double max = double.MinValue;
            double stdDev = 0.0;
            double stdErr = 0.0;

            // Run for a number of iterations
```

```
            string[] cols = { "D", "E", "F", "G", "H", "I", "J", "K",
"L", "M" };

            AutoResetEvent finishedEvt = new AutoResetEvent(false);
            int count = 0;

            SessionStartInfo info = new SessionStartInfo((string)
rngHeadNode.Value2, "AsianOptionsService");

            Stopwatch timer = null;

            using (Session session = Session.CreateSession(info))
            {
                NetTcpBinding binding = new NetTcpBinding();
                // Disable encryption
                binding.Security.Transport.ProtectionLevel =
ProtectionLevel.None;
   using (AsianOptionsClient client = new AsianOptionsClient(binding,
session.EndpointReference))
                {
                    timer = Stopwatch.StartNew();

                    // Set time out to MaxValue so that we'll not have
timeout exceptions
                    client.InnerChannel.OperationTimeout = new
TimeSpan(1, 0, 0);

                    foreach (string col in cols)
                    {
                        for (int i = 2; i <= 11; i++)
                        {
                            client.BeginPriceAsianOptions(
                                initial,
                                exercise,
                                up,
                                down,
                                interest,
                                periods,
                                runs,
                                (IAsyncResult result) =>
                                {
                                    double price =
client.EndPriceAsianOptions(result);
```

```
                                    // Populate the cell: Cell Id is
stored in result.AsyncState

this.Range[(string)result.AsyncState, missing].Value2 = price;

                                    Interlocked.Increment(ref count);

                                    min = Math.Min(min, price);
                                    max = Math.Max(max, price);

                                    sumPrice += price;
                                    sumSquarePrice += price * price;
                                    stdDev = Math.Sqrt(sumSquarePrice -
(sumPrice * sumPrice) / count) / ((count == 1) ? 1 : count - 1);
                                    stdErr = stdDev / Math.Sqrt(count);

                                    if (count == cols.Length * 10)
                                    {
                                        finishedEvt.Set();
                                    }
                                },
                                string.Format("{0}{1}", col, i));
                            }
                        }
                        finishedEvt.WaitOne();
                    }
                }

                timer.Stop();

                this.Range["D13", missing].Value2 = sumPrice / count;
                this.Range["D14", missing].Value2 = min;
                this.Range["D15", missing].Value2 = max;
                this.Range["D16", missing].Value2 = stdDev;
                this.Range["D17", missing].Value2 = stdErr;
                this.Range["D18", missing].Value2 =
timer.Elapsed.TotalMilliseconds / 1000.0;
}

private void Clear_Click(object sender, EventArgs e)
{
    this.Range["D2", "M11"].Clear();
    this.Range["D13", "D18"].Clear();
}
```

*Code 4-10 Button Event callback method*

10. Add the following code (in bold) on the **InternalStartup** method to set the button click's handlers, as shown in Code 4-11.

    (Code Snippet – *IntroToParallelComputing - Sheet1 buttons click handling*)

    **C#**

```
private void InternalStartup()
{
    this.Run.Click += new System.EventHandler(this.Run_Click);
    this.Clear.Click += new System.EventHandler(this.Clear_Click);
    this.Shutdown += new System.EventHandler(this.Sheet1_Shutdown);
    this.Startup += new System.EventHandler(this.Sheet1_Startup);
}
```

    *Code 4-11 Assign button event handlers*

11. (11) Add a **Session** property to hold the session on the scheduler. To do this, open ThisWorkbook.cs code-behind and add the following code (in bold), as shown in Code 4-12.

    (Code Snippet – *IntroToParallelComputing - ThisWorkbook ThisWorkbook_Startup implementation*)

    **C#**

```
private void ThisWorkbook_Startup(object sender, System.EventArgs e)
{
    Globals.Sheet1.Run.Enabled = true;
    Globals.Sheet1.Clear.Enabled = true;
}
```

    *Code 4-12 Workbook startup method*

12. Type CTRL+SHIFT+B to build the solution. We have completed the development of the client application. We should now be able to deploy the service.

### 4.2.3 Deploy the service

**1.** Create a new folder, **AsianOptionsService**, on each compute node under **C:\.**

> **Help**
>
> If you prefer to deploy the service on a different path, you will need to change the service element's assembly attribute in the AsianOptionsService.config file and on the following step.

2. Right-click the **bin** folder under the **AsianOptionsService** project and click **Open Folder in Windows Explorer**.
3. Locate and copy **AsianOptionsService.dll** to every Compute Node under the folder **C:\AsianOptionsService**.
4. Right-click **AsianOptionsService.config** file under the **AsianOptionsService** project and click **Open Folder in Windows Explorer**.
5. Locate and copy **AsianOptionsService.config** to every compute node and broker node of the Microsoft HPC 2008 R2 cluster in the **%CCP_HOME%\ServiceRegistration** folder.

**Help**

**%CCP_HOME%** is an environment variable of Microsoft HPC Server 2008 R2.

### 4.2.4 Run the application

1. Create a new folder, **TestAsianOptions**, on the head node under **C:**.

**Help**

You might ignore the next steps if you are running Visual Studio on the head node. If so, you should press CTRL+F5, then go to the step 13 and continue from there.

2. In Visual Studio, under Solution Explorer, right-click the **bin** folder under the **AsianOptions** project and click **Open Folder in Windows Explorer**.
3. Browse the **Debug** folder, then select all the files in that directory and copy them to the recently created folder **C:\TestAsianOptions** on the head node.
4. Switch back to the head node and browse **C:\TestAsianOptions**, as shown in Figure 4-11.

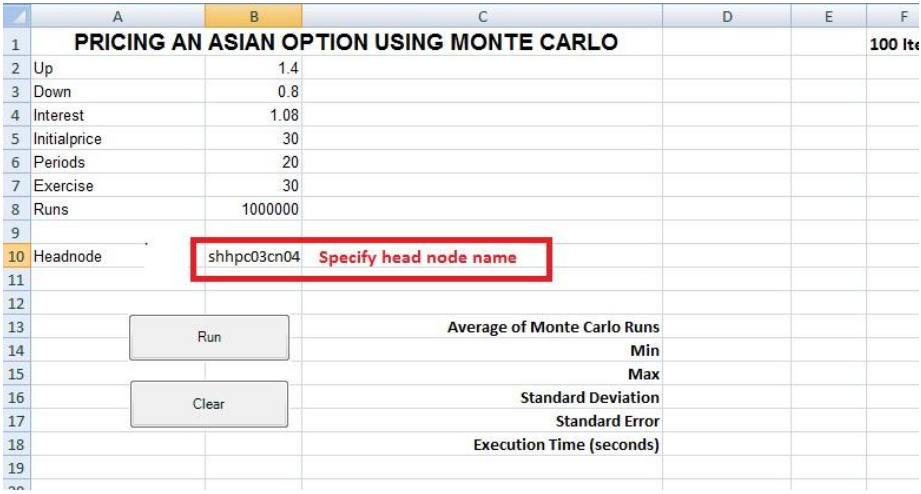



*Figure 4-11 AsianOptionPricingCSharpBehind Excel upon start-up*

5. Enter the name of the head node in cell B1, and then click **Run**. The results in Figure 4-12 are displayed.

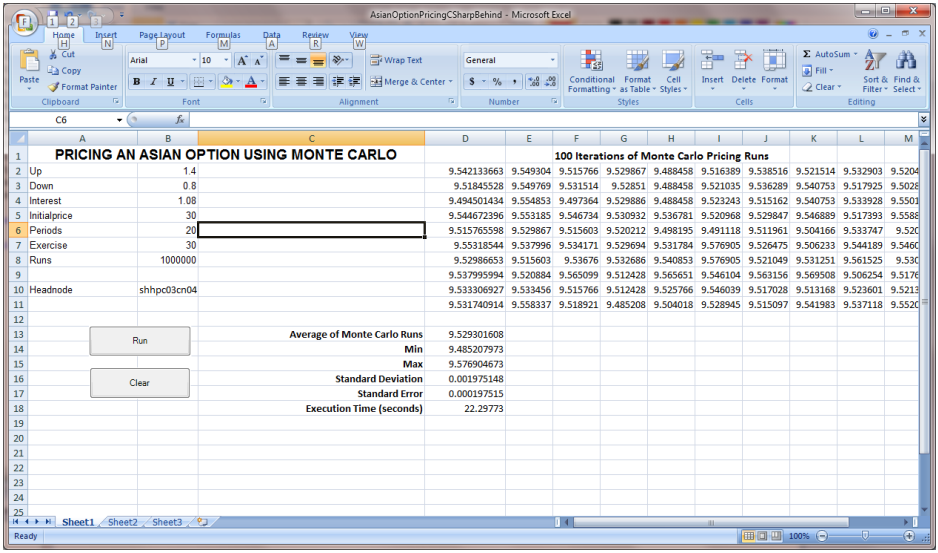



*Figure 4-12 After running the AsianOptionPricingCSharpBehind Excel workbook*

# 4.3 Create a Fire and Recollect Client Application

## 4.3.1 Create a new Sheet2

Following step 1-7 under 4.2.2 to create Sheet2, add three buttons, **Submit**, **Clear**, and **Retrieve** on Sheet2, as shown in Figure 4-13.

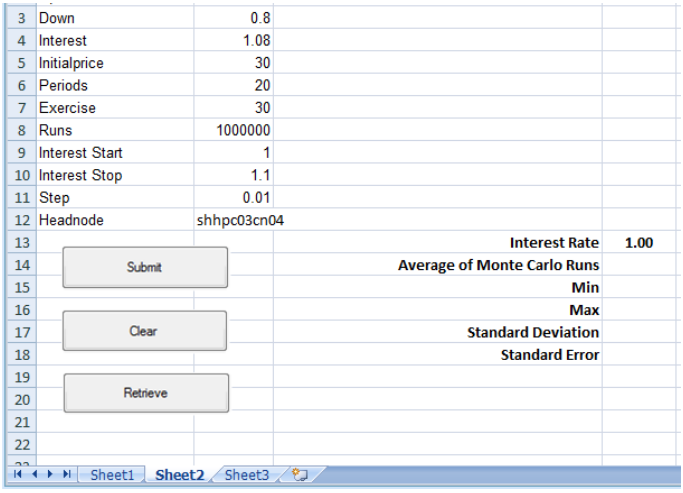



*Figure 4-13 Submit, Clear, and Retrieve Buttons*

1. **Add service reference to AsianOptions:** In Solution Explorer, right-click the **AsianOptions** project and select the **Add Service Reference...** menu item to launch the **Add Service Reference** dialog window, as shown in Figure 4-14. Click **Discover**, select the **AsianOptions.svc** service in the service list and change the namespace to **FRAsianOptionsService**.

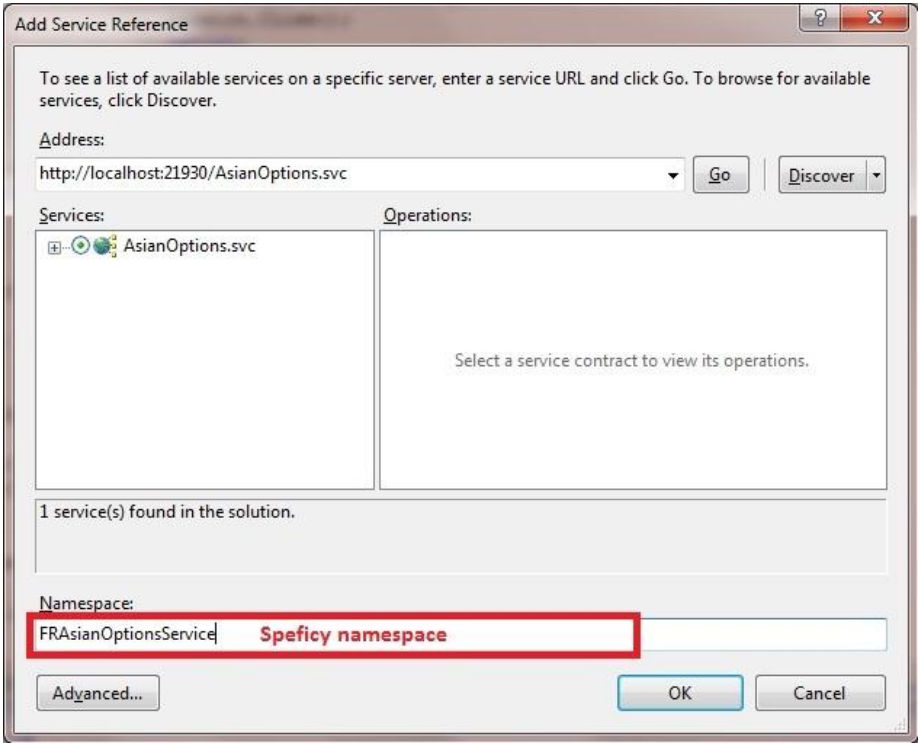



*Figure 4-14 Add Service reference*

2. Click **Advanced...** to display the **Service Reference Settings** dialog window. In this window, select **Always generate message contracts**, as shown in Figure 4-15, and then click **OK**.

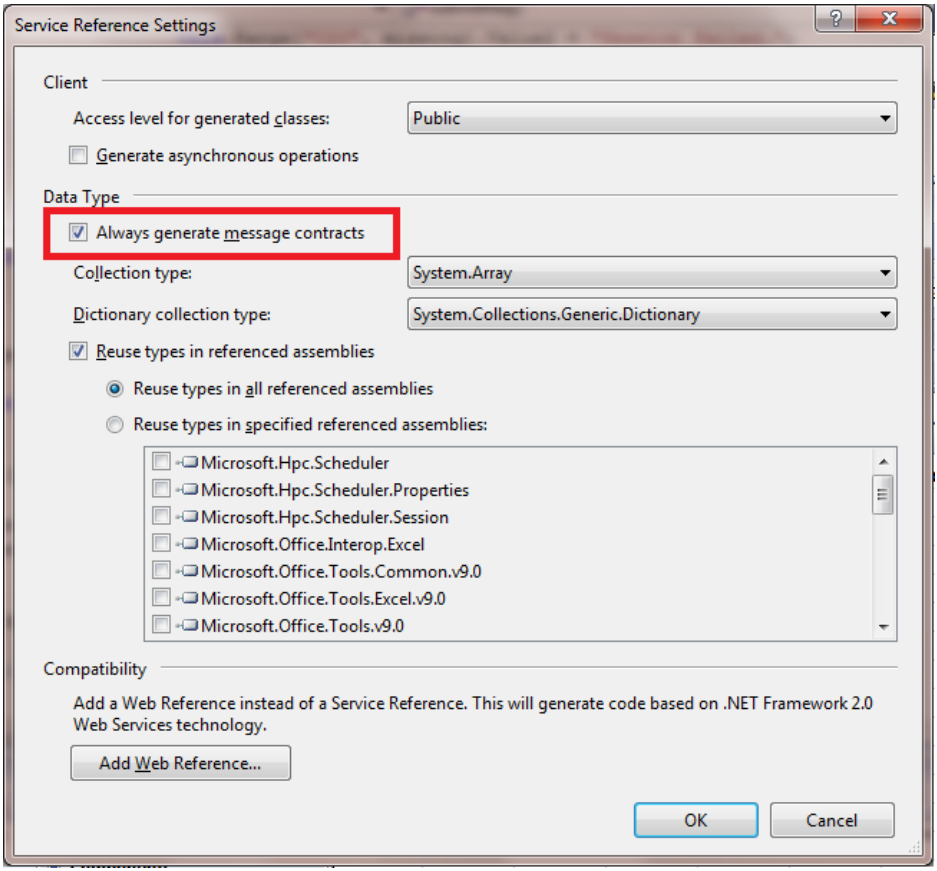



*Figure 4-15 Configure AsianOptionsService service reference*

3. Right-click **Sheet2** in Solution Explorer, and then click **View Code** to run the code behind Sheet2. As the text editor shows the code, add to the Sheet2 class the namespace declarations shown in Code 4-13.

   (Code Snippet – *IntroToParallelComputing – Sheet2 namespace references*)

**C#**

```
using System.Threading;
using Microsoft.Hpc.Scheduler.Session;
using AsianOptions.FRAsianOptionsService;
using System.ServiceModel;
```

*Code 4-13 Declaring namespaces*

4. Add to the Sheet2 class code (in bold) shown in Code 4-14.

(Code Snippet – *IntroToParallelComputing – Sheet2 Range fields*)

C#

```
public partial class Sheet2
{
        [Serializable]
         class cellContext
         {
            public string range;
            public int iteration;
         }

         class result
         {
            public double sumPrice;
            public double sumSquarePrice;
            public double average;
            public double min;
            public double max;
            public double stdDev;
            public double stdErr;
            public int count;

            public result()
            {
                sumPrice = 0;
                sumSquarePrice = 0;
                average = 0;
                min = double.MaxValue;
                max = double.MinValue;
                stdDev = 0;
                stdErr = 0;
                count = 0;
            }
        }

        readonly string[] cols = { "D", "E", "F", "G", "H", "I", "J",
"K", "L", "M" };

        private static Excel.Range rngUp, rngDown, rngInitial,
rngExercise, rngInterest, rngPeriods, rngRuns, rngInterestStart,
rngInterestEnd, rngStep, rngHeadnode;
    ...
```

*Code 4-14 Variables that denote Asian Option parameter ranges in the Excel workbook*

5. Initialize the fields just created on the **Sheet2** class. To do this, add the following code (in bold) on the **Sheet2_Startup** method, as shown in Code 4-15.

   (Code Snippet – *IntroToParallelComputing – Sheet2 Sheet2_Startup implementation*)

```
C#
```

```
private void Sheet2_Startup(object sender, System.EventArgs e)
{
    rngUp = this.Range["B2", missing];
    rngDown = this.Range["B3", missing];
    rngInterest = this.Range["B4", missing];
    rngInitial = this.Range["B5", missing];
    rngPeriods = this.Range["B6", missing];
    rngExercise = this.Range["B7", missing];
    rngRuns = this.Range["B8", missing];
    rngInterestStart = this.Range["B9", missing];
    rngInterestEnd = this.Range["B10", missing];
    rngStep = this.Range["B11", missing];
    rngHeadnode = this.Range["B12", missing];
}
```

*Code 4-15 Initializing Sheet2*

6. Add handling for the buttons on the Sheet2 Excel sheet. To do this, add the following methods to the **Sheet2** class**,** as shown in Code 4-16, 4-17, and 4-18.

   (Code Snippet – *IntroToParallelComputing – Sheet2 buttons click handling implementation*)

```
C#
```

```
private void Submit_Click(object sender, EventArgs e)
        {
            ThreadPool.QueueUserWorkItem((x) => submitReq());
        }

        private void submitReq()
        {
            #region Initialization
            double initial = (double)rngInitial.Value2;
            double exercise = (double)rngInitial.Value2;
            double up = (double)rngUp.Value2;
            double down = (double)rngDown.Value2;
            double interest = (double)rngInterest.Value2;
            int periods = Convert.ToInt32(rngPeriods.Value2);
            int runs = Convert.ToInt32(rngRuns.Value2);
            double interestStart = (double)rngInterestStart.Value2;
            double interestEnd = (double)rngInterestEnd.Value2;
```

```
            double interestStep = (double)rngStep.Value2;
            //Config.headNode = (string)rngHeadnode.Value2;
            #endregion

            #region fire request

            SessionStartInfo info = new
SessionStartInfo((string)rngHeadnode.Value2, "AsianOptionsService");
            info.Secure = false;
            info.BrokerSettings.SessionIdleTimeout = 12 * 60 * 60 *
1000;  // 12 hours

            DurableSession.SetInterfaceMode(false, IntPtr.Zero); //set
interface mode to non console

            using (DurableSession session =
DurableSession.CreateSession(info))
            {
                this.Range["C20", missing].Value2 = "Session Created.";
                this.Range["D20", missing].Value2 = session.Id;
                Thread.Sleep(1000);
                this.Range["C20", missing].Value2 = " Sending Req...";

                NetTcpBinding binding = new NetTcpBinding();
                // Disable encryption
                binding.Security.Transport.ProtectionLevel =
ProtectionLevel.None;

                using
(BrokerClient<AsianOptions.FRAsianOptionsService.IAsianOptions> client
= new
BrokerClient<AsianOptions.FRAsianOptionsService.IAsianOptions>(session,
binding))
                {
                    int count = 0;
                    for (double interestIdx = interestStart;
interestIdx < interestEnd; interestIdx += interestStep, count++)
                    {
                        this.Range["C20", missing].Value2 =
string.Format("Session Created. Sending Req Batch {0}", count);

                        foreach (string col in cols)
```

```
                        {
                            for (int i = 2; i <= 11; i++)
                            {
                                PriceAsianOptionsRequest priceRequest =
new PriceAsianOptionsRequest(initial, exercise, up, down, interestIdx,
periods, runs);
                                cellContext ctx = new cellContext();
                                ctx.range = string.Format("{0}{1}",
col, i);
                                ctx.iteration = count;
                                try
                                {

client.SendRequest<PriceAsianOptionsRequest>(priceRequest, ctx);
                                }
                                catch (Exception)
                                {
                                    // Populate the cell with an error
message
                                    this.Range[ctx.range,
missing].Value2 = "#SendErr#";
                                    this.Range["C20", missing].Value2 =
"Session failed.";
                                    this.Range["D20", missing].Clear();
                                    session.Close();
                                    return;
                                }
                            }
                        }
                    }
                    client.EndRequests();
                    this.Range["C20", missing].Value2 = "Closing.";
                }
            }

            this.Range["C20", missing].Value2 = "Request sent.";

            #endregion
        }
```

*Code 4-16 Event handler method for the Submit button*

C#

```
private void Retrieve_Click(object sender, EventArgs e)
        {
            ThreadPool.QueueUserWorkItem((x) => retrieveResp());
        }

        private void retrieveResp()
        {
            #region Init
            result[] results = new result[10];
            for (int i = 0; i < 10; i++)
                results[i] = new result();
            #endregion


            SessionAttachInfo attachInfo = new
SessionAttachInfo((string)rngHeadnode.Value2,
Convert.ToInt32(this.Range["D20", missing].Value2));

            using (DurableSession session =
DurableSession.AttachSession(attachInfo, 20 * 1000))
            {
                NetTcpBinding binding = new NetTcpBinding();
                // Disable encryption
                binding.Security.Transport.ProtectionLevel =
ProtectionLevel.None;

                using
(BrokerClient<AsianOptions.FRAsianOptionsService.IAsianOptions> client
= new
BrokerClient<AsianOptions.FRAsianOptionsService.IAsianOptions>(session,
binding))
                {

                    foreach (BrokerResponse<PriceAsianOptionsResponse>
response in client.GetResponses<PriceAsianOptionsResponse>())
                    {
                        cellContext idx =
response.GetUserData<cellContext>();
                        double price =
response.Result.PriceAsianOptionsResult;
```

```
                        Interlocked.Increment(ref
results[idx.iteration].count);

                        this.Range[idx.range, missing].Value2 = price;

                        results[idx.iteration].min =
Math.Min(results[idx.iteration].min, price);
                        results[idx.iteration].max =
Math.Max(results[idx.iteration].max, price);

                        results[idx.iteration].sumPrice += price;
                        results[idx.iteration].sumSquarePrice += price
* price;

                        results[idx.iteration].stdDev =
Math.Sqrt(results[idx.iteration].sumSquarePrice -
results[idx.iteration].sumPrice * results[idx.iteration].sumPrice /
results[idx.iteration].count) / ((results[idx.iteration].count == 1) ?
1 : results[idx.iteration].count - 1);
                        results[idx.iteration].stdErr =
results[idx.iteration].stdDev /
Math.Sqrt(results[idx.iteration].count);

                        if (results[idx.iteration].count == 100)
                        {
                            int i = idx.iteration;
                            this.Range[string.Format("{0}14", cols[i]),
missing].Value2 = results[i].sumPrice / results[i].count;
                            this.Range[string.Format("{0}15", cols[i]),
missing].Value2 = results[i].min;
                            this.Range[string.Format("{0}16", cols[i]),
missing].Value2 = results[i].max;
                            this.Range[string.Format("{0}17", cols[i]),
missing].Value2 = results[i].stdDev;
                            this.Range[string.Format("{0}18", cols[i]),
missing].Value2 = results[i].stdErr;
                        }
                    }
                }
                session.Close();
            }

            #region Summarize
```

```
            for (int i = 0; i < 10; i++)
            {
                this.Range[string.Format("{0}14", cols[i]),
missing].Value2 = results[i].sumPrice / results[i].count;
                this.Range[string.Format("{0}15", cols[i]),
missing].Value2 = results[i].min;
                this.Range[string.Format("{0}16", cols[i]),
missing].Value2 = results[i].max;
                this.Range[string.Format("{0}17", cols[i]),
missing].Value2 = results[i].stdDev;
                this.Range[string.Format("{0}18", cols[i]),
missing].Value2 = results[i].stdErr;
            }
            #endregion

        }
```

*Code 4-17 Event handler method for the Retrieve button*

```
private void Clear_Click(object sender, EventArgs e)
{
    this.Range["D2", "M11"].Clear();
    this.Range["D13", "M18"].Clear();
}
```

*Code 4-18 Event handler method for the Clear button*

## 4.3.2 Run the application

1. In the Visual Studio window, click **Debug**, and then click **Start Without Debugging** to run the application, as shown in Figure 4-16.

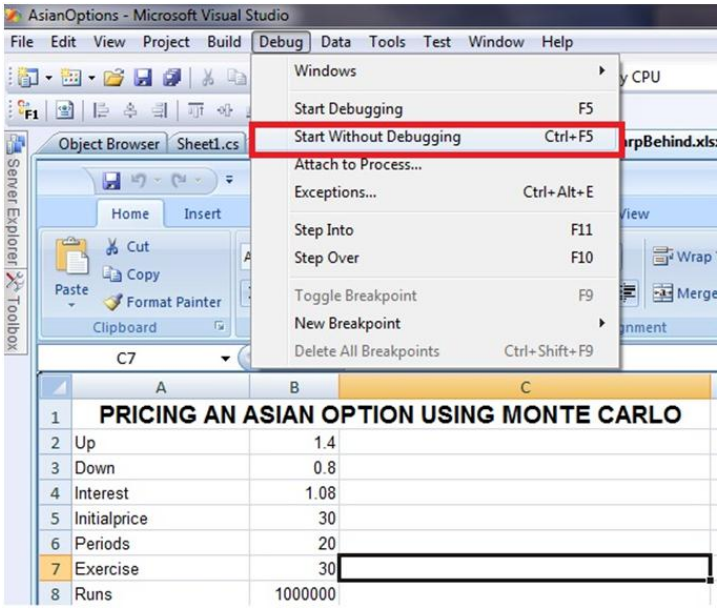



*Figure 4-16 Run the client application from Visual Studio*

2. The application opens an Excel window, as shown in Figure 4-17.

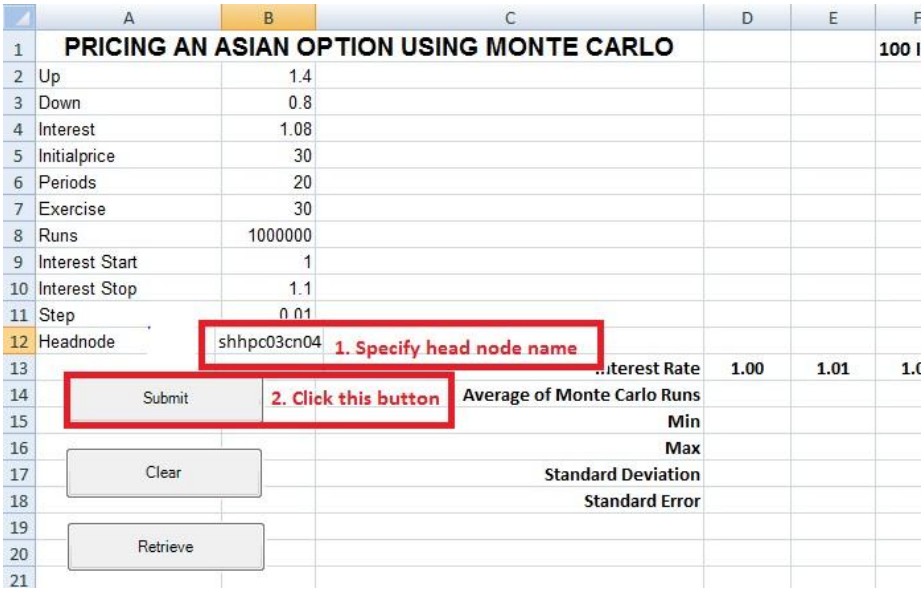



*Figure 4-17 Excel Workbook*

3. Enter the head node name into cell B12 and click **Submit**. The progress information is displayed in cell C20. When all the requests are submitted, cell C20 shows "Request sent" and cell D20 shows the Job ID, as shown in Figure 4-18.

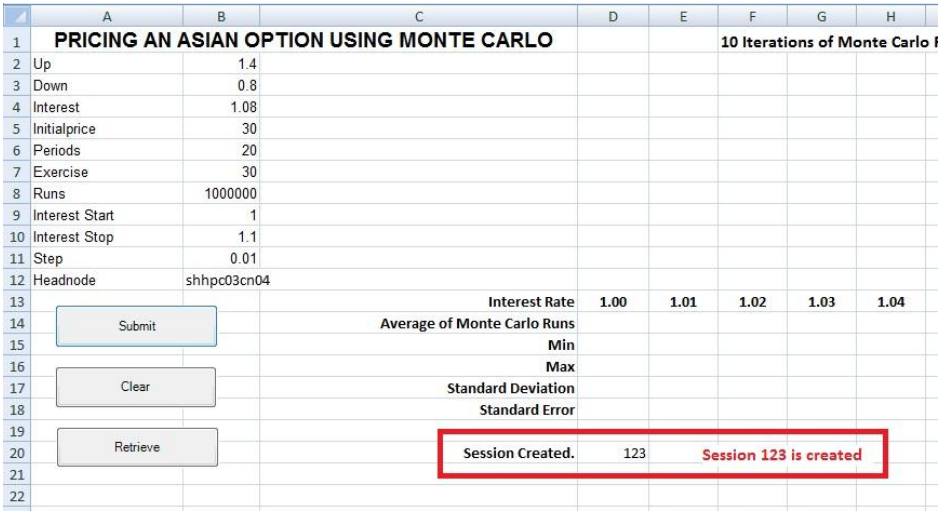



*Figure 4-18 Submit a session job from the Excel workbook*

4. Exit Excel. In the file saving dialog, click **Yes**, as shown in Figure 4-19.

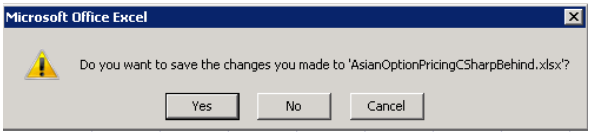



*Figure 4-19 Save the workbook*

5. Return to the Job Manager to monitor the job status, until the job finishes, as shown in Figure 4-20.

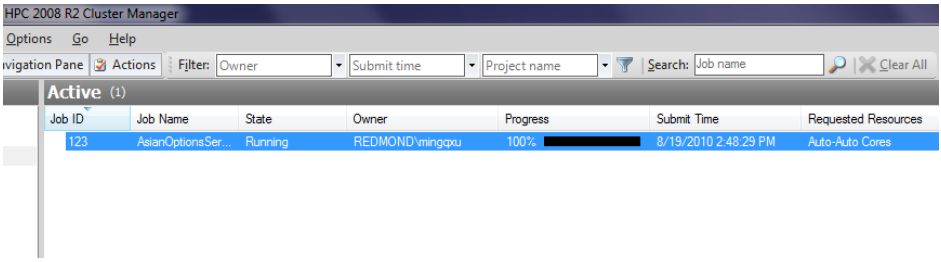



*Figure 4-20 Monitoring the session job's status*

6. Restart the application by beginning with Step 1 in this procedure, as shown in Figure 4-21.

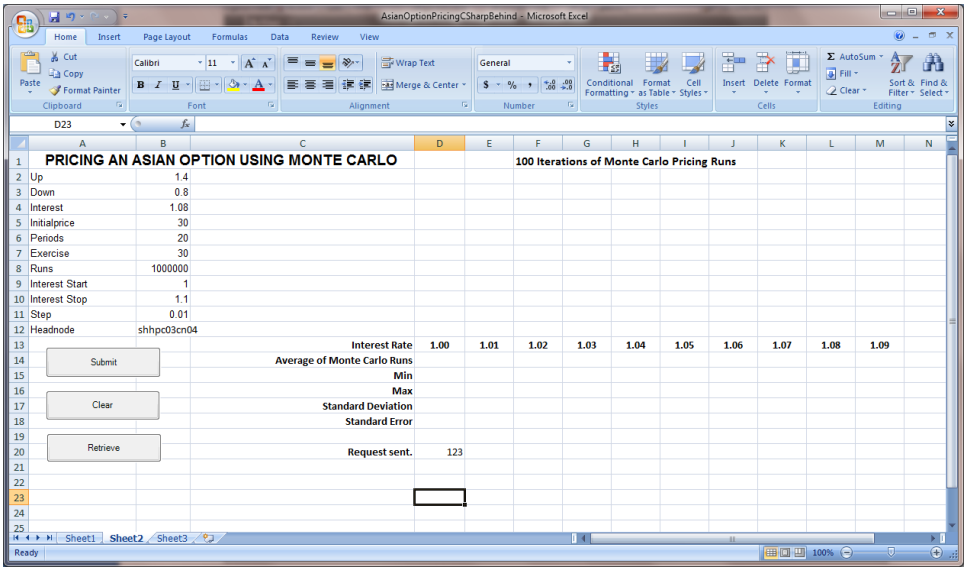



*Figure 4-21 Restart the application; cell D20 stores the job ID*

7. Click **Retrieve** button to retrieve the results, as shown in Figure 4-22.

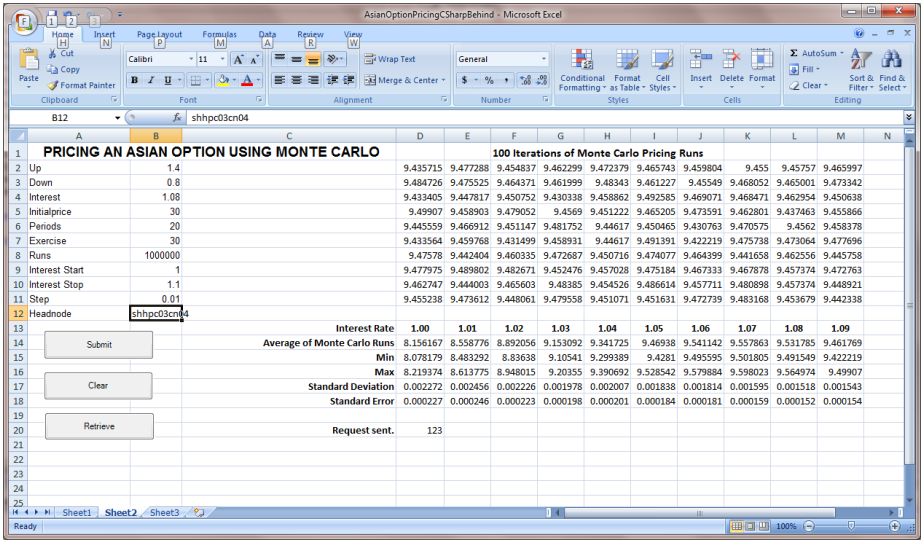



*Figure 4-22 Retrieve results*

8. Switch to Sheet3, and you see the curve showing the price of the Asian Options as a function of the interest rate, as shown in Figure 4-23.

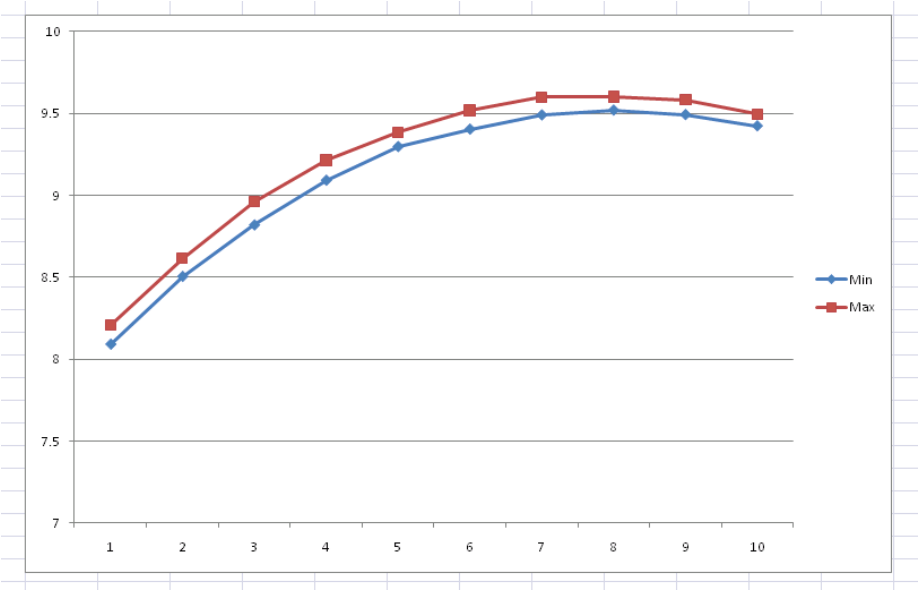



*Figure 4-23 Min/Max AsianOptions price as a function of interest rate*

# Chapter 5 Programming MPI-based HPC Applications

In Chapter 4, we introduced a problem that is easy to parallelize. The reason that the problem is easy is that the work units can be easily partitioned and the processing elements aren't required to perform synchronization among themselves during the calculation. In this chapter, we will introduce a traditional HPC problem, one that requires a more complicated work unit partition, runtime data distribution/aggregation, and communication among the processing elements. We will present a classic HPC development scenario centered around data parallelism, using Visual C++, MPI, and HPC Server 2008 to develop high-performance, parallel solutions.

Chapter Highlights:

- Understand how to use MPI to develop data parallel applications
- Learn how to perform domain decomposition
- Understand the parallel programming methods for interprocess parallelization and synchronization

## 5.1 Problem Domain

Image processing is a compute-intensive domain. Given the size of today's images, and the wide range of special effects, it is not uncommon to consume hours of CPU time in the processing of a single image. A representative example of a problem in this domain is *contrast stretching*, where contrast is enhanced by lightening or darkening pixels based on neighboring pixels. Figure 5-1 shows an image before processing and we cannot see whether there are other boats in the harbor. How does contrast stretching help? For every pixel P, the typical approach is to determine the min and max of its eight neighbors, and then adjust P upward/downward based on the ratio of lightness to darkness in relation to P.

*Figure 5-1 Image before processing*

The best result is obtained by adjusting the image slowly, and repeating until either (a) the image *converges* (no longer changes from one iteration to the next), or (b) the desired effect has been achieved (by performing a specified number of iterations). For example, consider the images to your right. The upper image is the original, capturing a sailboat in a South Pacific harbor at sunset. The lower image is the equivalent image after contrast stretching for 75 iterations. Notice in Figure 5-2 that the stretching reveals more clearly the presence of other boats in the harbor, in particular their white masts. Contrast stretching is one of the many techniques used to enhance images.

*Figure 5-2 Image after processing*

## 5.2 Digital Representation of an Image

Programmatically, images are most easily treated as two-dimensional arrays of integers. For simplicity, we'll work with bitmaps (.bmp), where each pixel is stored as three distinct integers (0..255) representing the amount of blue, green, and red at that pixel. Thus, an M-by-N image contains M*N pixels, M*N*3 integers, and will be represented by a 2D array with N rows and M columns. In this approach, each element of the array denotes a single pixel, which we'll represent using a structure containing three fields, each field an unsigned char since the range is 0..255, as shown in Code 5-1.

```
typedef struct {

                    uchar  blue;   // amount of blue at this pixel (0..255, 0
=> no blue/black, 255 => max blue/white)

                    uchar  green;  // amount of green (0..255)

                    uchar  red;    // amount of red (0..255)

          } PIXEL_T;

          PIXEL_T  image[N][M];
```

*Code 5-1 Pixel array*

For example, an 800 x 600 bitmap contains 480,000 pixels, 1,440,000 bytes, and will be represented by an array of PIXEL_T with 600 rows and 800 columns, as shown in Figure 5-3.

*Figure 5-3 800 x 600 image's pixel array*

The pseudocode for the main loop is shown in Code 5-2. The algorithm repeats until convergence, or until the desired number of steps has been performed. For every step, each non-boundary pixel of the image is visited, its neighbors consulted, and the pixel P is either darkened by subtracting 1 (black is the absence of color), lightened by adding 1 (white is the presence of all colors), or left alone. The updated pixels are stored in a temporary image matrix I' so updates do not interfere with the current computation. After all the pixels have been processed, a convergence check is made (based on the total number of updated pixels), and the original image matrix I is updated by copying I' back into I.

Given a pixel P, contrast stretching is performed separately on each color value of blue, green, and red. The first step is to determine the min and max values by sorting the eight immediate neighbors. Then a ratio of lightness to darkness relative to P is computed:

ratio = (*P* – min) / (max – min);

If the ratio is small, then P is closer to the min, which implies the neighbors have more of this color relative to P. To enhance contrast, this implies P's value should be reduced. If the ratio is large, then P is closer to the max, and the reverse is true—the neighbors have less of this color relative to P, and so P's value should be increased. Here's more complete pseudocode, as shown in Code 5-2.

```
          if (min == max)  // then all pixels have the same value, so leave P
alone:

                    ;

          else  // enhance contrast of P as appropriate:

          {

                    ratio = (P - min) / (max - min);

                    if (ratio < 0.5)      // P is closer to the min, neighbors
have more color ==> so reduce P's color

                              P--;

                    else if (ratio > 0.5) // P is closer to the max, neighbors
have less color ==> so increase P's color

                              P++;

                    else                  // equally split, we leave P alone:

                              ;

          }
```

*Code 5-2 Contrast stretching algorithm*

We'll look at this in more detail.

As an aside, it is worth noting the layout of a bitmap file. A .bmp file contains two things: a fixed-sized header with information about the image, followed by the pixel values themselves. The pixel values are stored in row-major order, starting with the row you see at the bottom of the image (when viewed on your screen) and finishing with the row you see at the top. Each pixel is stored as 3 consecutive bytes, denoting the amount of blue, green, and red (in this order). For example, if you are looking at a bitmap on the screen and the bottom-left pixel is red, then this means the first 3 bytes of image data are 0 0 255. And if the bottom-right pixel is white (assuming an 800 x 600 image), then the 2398$^{th}$, 2399$^{th}$, and 2400$^{th}$ bytes of image data are 255 255 255. Finally, if the top-right pixel is yellow, then the last 3 bytes in the image (and the file) are 0 255 255, as shown in Figure 5-4.

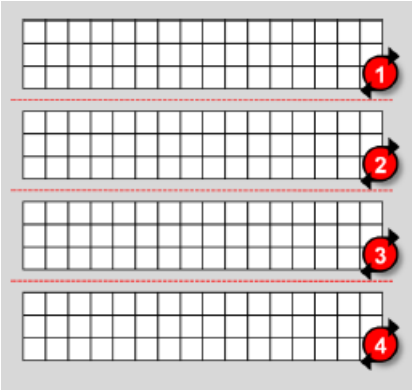



*Figure 0-4 Pixel file*

## 5.3 Data Parallelism and Contrast Stretching

Contrast stretching is a classic example of *data parallelism*—where the same computation (contrast stretching) can be performed in parallel across a dataset (the image). For example, given an image with 12 rows and a system with 4 independent execution units (see Figure 5-5), we can assign 3 rows to each execution unit and run the contrast stretching algorithm on each unit in parallel. More realistically, given an image with 600 rows and 4 execution units, we would assign 150 rows to each execution unit for parallel processing. In either case, the result is often 4 times as fast in performance.

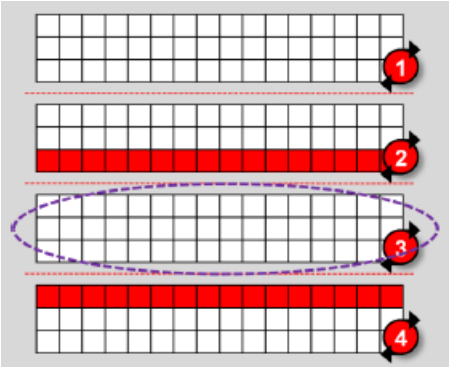



*Figure 0-5 Pixels assigned by four execution units*

There are a number of issues that must be taken into account, however. First, when an execution unit processes its first row, it will need the pixels from the previous row, which are owned by a different execution unit (this holds true for all execution units except the first—in which case the first row is an image boundary row, and not processed). Likewise, when an execution unit processes its last row, it will need the row that follows, which is owned by yet another execution unit (true for all execution units but the last). For example, consider execution unit 3: it will need the last row of unit 2 and the first row of unit 4, as shown in Figure 5-6.

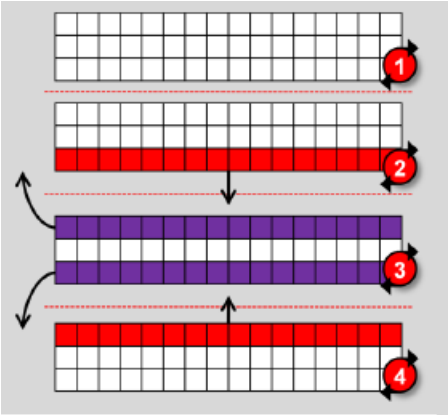



*Figure 5-6 Overlapped pixels*

In general, this implies that during each step of the contrast stretching algorithm, the execution units will need to exchange data, as shown in Code 5-3.

```
while (!converged && step < STEPS)
{
          step++;


          retrieve row preceding my first row (unless I am the first execution
unit);
          retrieve row following my last row (unless I am the last execution
unit);


          foreach (...)
                    foreach (...)
                    {
                              I'[r][c] = Stretch(...);
                              if (I'[r][c] != I[r][c]) localDiffs++;
                    }
```

*Code 5-3 Processing overlapped pixels*

Second, to properly test for convergence, at the end of each step the execution units will need to communicate their local convergence values to each other. The image has converged if and only if convergence holds across all execution units, as shown in Code 5-4.

```
diffs = localDiffs;

foreach (other execution unit E)

{

          temp = retrieve E's local diffs convergence value;

          diffs += temp;

}


converged = (diffs == 0);
```

*Code 5-4 Check convergence condition*

On a cluster, execution units do not share memory, so a distributed-memory approach must be taken. This implies the execution units must physically transfer data when they want to communicate. MPI is a good choice in this case, given its flexibility, broad availability, and potential for high performance on a wide variety of hardware. Message-passing with send/receive represents the core of MPI, and is easily used in a parallelized contrast stretching to communicate rows between execution units, as shown in Code 5-5.

```
let P = total number of execution units;

let N = my execution unit number (1<=N<=P);


while (!converged && step < STEPS)

{

          step++;


          MPI_Send my first row to execution unit N-1 (unless I am unit 1);

          MPI_Send my last row to execution unit N+1 (unless I am unit P);


          MPI_Recv preceding row from execution unit N-1 (unless I am unit 1);

          MPI_Recv trailing row from execution unit N+1 (unless I am unit P);


          foreach (...)

                    foreach (...)

                    {

                              I'[r][c] = Stretch(...);

                              if (I'[r][c] != I[r][c]) localDiffs++;

                    }
```

*Code 5-5 Cluster solution pseudocode*

For example, suppose you are execution unit 3 in Figure 5-6. The neighboring execution units need your first and last rows (shown in purple) to perform their contrast stretching, so you send the first row to unit 2 and the last row to unit 4. Next, you (unit 3) need the rows preceding and following your own, so you receive these rows (shown in red) from execution units 2 and 4, respectively. Since every execution unit is running the same algorithm, all sends and receives complete, and the contrast stretching proceeds in parallel. Once all pixels have been processed, the execution units will need to communicate their convergence values to determine if the image has converged. While this can be done through a series of sends and receives, MPI offers a much more efficient, safer, and readable *collective* operation for summing across the execution units, as shown in Code 5-6.

```
MPI_Allreduce(localDiffs, diffs, ..., MPI_SUM, ...);  // sum all the
localDiffs, and distribute final diffs

converged = (diffs == 0);
```

*Code 5-6 Allreduce solution*

## 5.4 A Sequential Version of the Contrast Stretching Application

An important first step in developing a parallel version of an application is to create a sequential version, as shown in Figure 5-7.

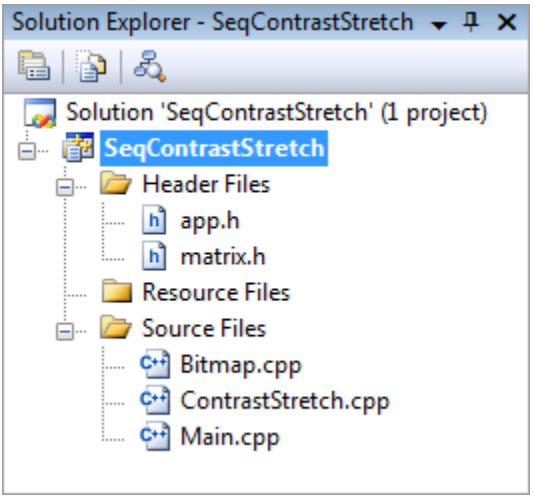



*Figure 0-1 Sequential version solution*

A sequential version allows us to gain a better understanding of the problem, provides a vehicle for correctness testing against the parallel versions, and forms the basis for performance measurements. Performance is often measured in terms of *speedup*, that is, how much faster the parallel version executed in comparison to the sequential version. More precisely:

$$speedup = \frac{\text{Sequential}_{\text{time}}}{Parallel_{\text{time}}}$$

For example, if the sequential version runs in 80 seconds and the parallel version runs in 20 seconds, then the speedup is 4. If the parallel version was run on 4 execution units, this is a very good result—the sequential workload was perfectly parallelized across the execution units with no measurable overhead. If, however, the parallel version was run on 16 execution units, then the quality of the result depends on your expectations. A time of 20 seconds might be considered good if the application was difficult to parallelize. If, on the other hand, the application was considered well-written and highly-parallel, then it's a bad result—the time should have been closer to 5 seconds.

Why 5 seconds? Generally, the goal of parallelizing an application is *linear* speedup: given N execution units, the parallel version should run N times faster than the sequential one (see graph on right). Linear speedup implies the parallel application is fully using all N execution units in meaningful computation, as shown in figure 5-8. Given a sequential application running in 80 seconds, the parallel version should run in 40 seconds on 2 execution units, 20 seconds on 4 units, 10 seconds on 8 units, and 5 seconds on 16 units. Linear speedup is one of the Primary goalsof HPC.

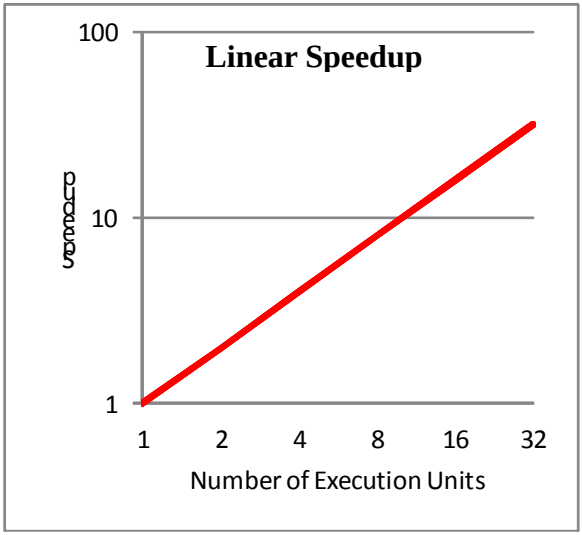



*Figure 0-2 Speedup*

### 5.4.1 Sequential version architecture

Open the sequential solution in Visual Studio 2008: *Chapter 5\Solutions\Sequential\SeqContrastStretch*. The solution consists of two include files and three source code files:

| | |
|---|---|
| App.h | application-wide definitions |
| Matrix.h | function templates for creating efficient 2D matrices |
| Bitmap.cpp | code for reading and writing .bmp files |
| ContrastStretch.cpp | implements the algorithm for contrast stretching |
| Main.cpp | main program (startup, argument processing, file I/O, and cleanup) |

Let's start by running the program to get a feeling for how it behaves. First, select the platform for your local workstation (Win32 or x64) via the drop-down in Visual Studio's standard toolbar, as shown in Figure 5-9.

*Figure 5-9 Choose platform*

Now run the application by pressing Ctrl+F5 (run without debugging). By default, the application will run until the image converges, as shown in Figure 5-10.

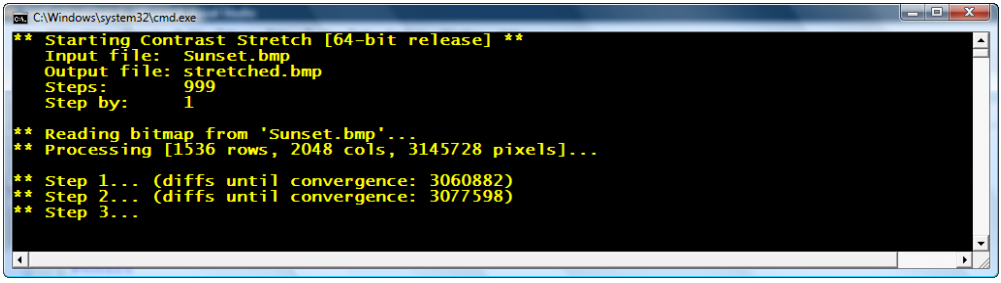



*Figure 5-10 Sequential version run result*

This may take a while, so feel free to quit early (Ctrl+C). The application is controlled via four command-line arguments: input file, output file, number of steps to perform, and value to step by. These default to Sunset.bmp, Stretched.bmp, 999, and 1, respectively. By supplying your own arguments, you can configure the application for shorter runs, which is more conducive to debugging and testing. For example, as a quick test run, I often perform 5 steps and step by 10. You can configure Visual Studio as follows to supply command-line arguments: locate your Solution Explorer window (see Figure 5-7), right-click the **SeqContrastStretch** project (highlighted in Figure 5-7), and select **Properties**. In the window that opens, under **Configuration Properties**, click **Debugging**, and enter the desired values into **Command Arguments**, as shown in Figure 5-11.

Sunset.bmp  temp.bmp  5  10

The output file will be placed in *Chapter 5\Solutions\Sequential\SeqContrastStretch\SeqContrastStretch*. Note that the input file is being read from this folder as well.

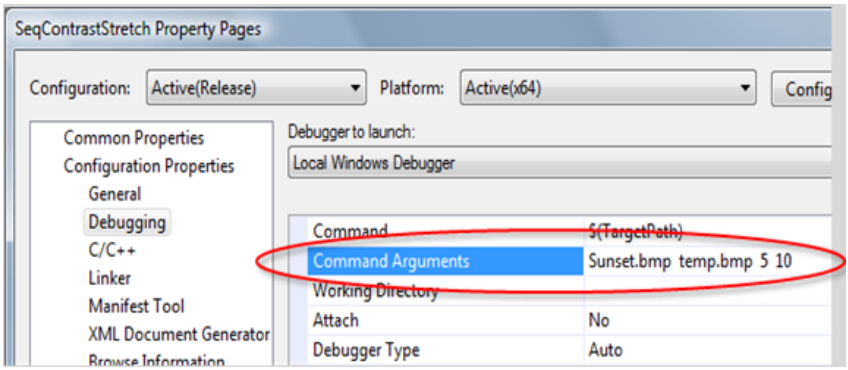



*Figure 5-11 Project property dialog*

Open the main program Main.cpp and skim over the code. You'll see that the main function performs four main steps: processing of command-line arguments, inputting of the bitmap file, contrast stretching, and outputting the result. Notice that the main file #includes only App.h. If you open App.h, you'll see that this file includes the necessary header files, as well as defining PIXEL_T and function prototypes.

The file Bitmap.cpp defines two functions, **ReadBitmapFile** and **WriteBitmapFile**. The read function opens a .bmp file, reads the header and image data, and returns the image data as a newly allocated 2D array. The first row of the array (image[0]) is the bottom-most row of the image when viewed on the screen, and the last row of the array (image[rows-1]) is the top-most row of the image when viewed. The write function takes a header and 2D array, and writes it out as a .bmp file. It is the caller's responsibility to deallocate the 2D array of image data.

Last, the file ContrastStretch.cpp implements the contrast stretching algorithm. It defines a single public function **ContrastStretch** called by the main program, as shown in Code 5-7.

```
PIXEL_T **ContrastStretch(PIXEL_T **image, int rows, int cols, int steps, int
stepby)

         {

                    PIXEL T **image2 = New2dMatrix<PIXEL T>(rows, cols);  //
temp array for updated pixel values:


                    bool converged = false;

                    int  step      = 0;

                    long long  diffs;


                    while (!converged && step < steps)  // perform contrast
stretching:

                    {

                              step++;


                              diffs = 0;

                              for (int row = 1; row < rows-1; row++)  //
foreach non-boundary row:

                              {

                                        for (int col = 1; col < cols-1; col++)
// foreach non-boundary column:

                                        {

                                                  image2[row][col].blue  =
StretchColor(image[row-1][col-1].blue, ...);

                                                  image2[row][col].green =
StretchColor(image[row-1][col-1].green, ...);

                                                  image2[row][col].red   =
StretchColor(image[row-1][col-1].red, ...);


                                                  if ((image2[row][col].blue
!= image[row][col].blue) || ...)  // did pixel change?

                                                            diffs++;

                                        }

                              }

                              converged = (diffs == 0);  // did we converge?
```

```
                                for (int row = 1; row < rows-1; row++)   // copy
updated image back into original:
                                          for (int col = 1; col < cols-1; col++)
                                                    image[row][col] =
image2[row][col];
                        }


                        Delete2dMatrix<PIXEL T>(image2);  // Done!  Delete temp
memory and return image:
                        return image;
            }
```

*Code 5-7 ConstrastStretch function*

Given an *image* of size *rows* by *cols*, it stretches the image at most *steps* times, stretching by *stepby* each time. The image is updated in place, and returned by the function. The file defines internal functions for sorting neighboring pixels (*median*) and computing new pixel values (*StretchColor*), as shown in Code 5-8.

```
uchar median(uchar a, uchar b, uchar c, uchar d, uchar e, uchar f, uchar g,
uchar h, uchar i, uchar &min, uchar &max)
            { /* sort the 9 pixel values, computing min and max, returning
median */ }


            uchar StretchColor(uchar UL, uchar UP, uchar UR, uchar L, uchar P,
uchar R, uchar DL, uchar DW, uchar DR, int stepby)
            {
                        uchar newp, med, min, max;
                        double ratio;


                        med = median(UL, UP, UR, L, P, R, DL, DW, DR, min, max);


                        if (min == max)  // then all pixels have the same value,
so leave P alone:
                                    newp = min;
                        else
                        {
                                    // do neighbors overall contain more of this
color, or less?
                                    ratio = (P - min) / (double) (max - min);


                                    if (ratio < 0.5)  // P is closer to the min,
neighbors have more color ==> so reduce P's color
                                    {
                                              if (P > stepby) // beware of
underflow:
```

```
                                                    newp = P - stepby;
                                          else
                                                    newp = 0;
                                }
                                else if (ratio > 0.5)  // P is closer to the
max, neighbors have less color ==> so increase P's color
                                {
                                          if (P < 255-stepby)  // beware of
overflow:
                                                    newp = P + stepby;
                                          else
                                                    newp = 255;
                                }
                                else  // P is exactly in the middle relative to
min & max, so leave pixel alone:
                                          newp = P;
                      }


                      return newp;
           }
```

*Code 5-8 Median function*

### 5.4.2 Using MPI to parallelize the contrast stretching algorithm

The solution is in *Chapter 5\Solutions\MPI\MPIContrastStretch*. The steps are as follows.

1. Open the application in *Chapter 5\Solutions\MPI\MPIContrastStretch*. This is a parallel contrast stretching program. Switch to your workstation's target platform (Win32 or x64), then compile the project. Add references to MPI and Winsock2 libraries as follows : right-click the project name (**SQL_MPIContrastStretch**) in Solution Explorer, expand **Configuration Properties**, expand **Linker**, and select **Input**. In the configuration page that appears, enter the string value "msmpi.lib Ws2_32.lib" in the **Additional Dependencies** text field, as shown in Figure 5-12.

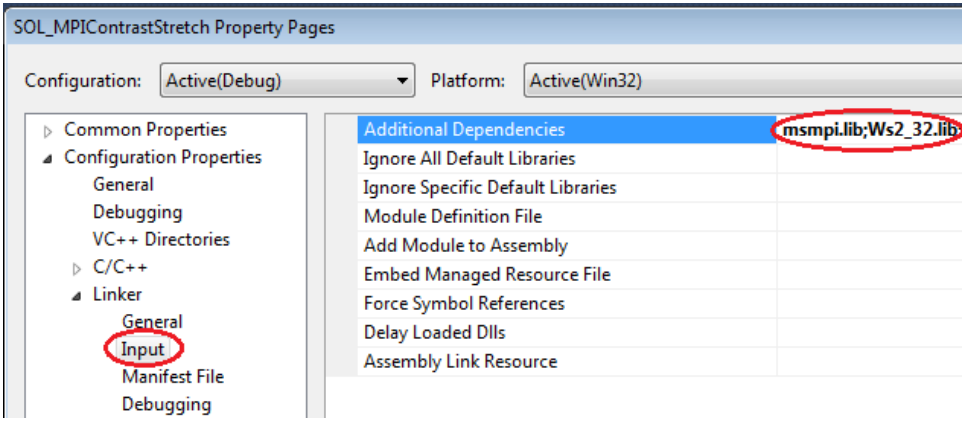



*Figure 5-12 Add MPI library*

Then expand **C/C++** and select **Code Generation**. In the Runtime Library under the Configuration Pane, select **Multi-threaded (/MT)** (the default is **Multi-threaded Debug DLL (/MDd)**), as shown in Figure 5-13.

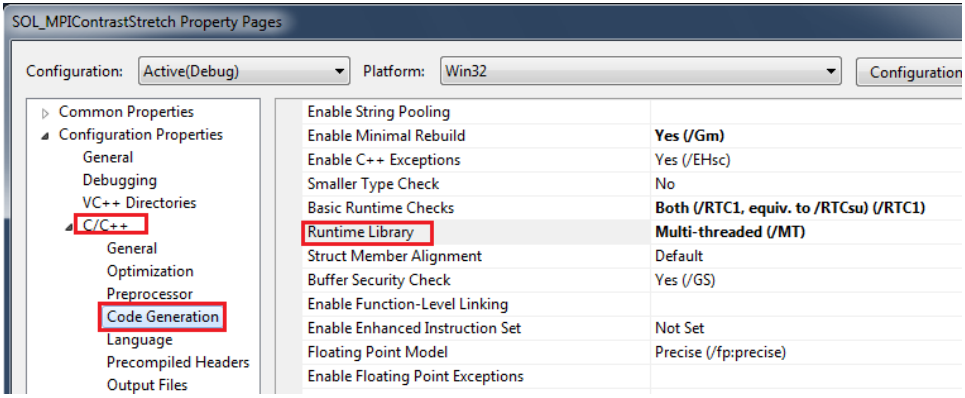



*Figure 5-13 Modify Compiling option*

The program should now build successfully. Note that this must be done for each project type: Win32 Debug, Win32 Release, x64 Debug, and x64 Release.

The second step is to configure Visual Studio to locate MSMPI during compilation and linking; this needs to be done only once. Start Visual Studio, click **Tools**, and then click **Options** to start the **Options** dialog. Under the **Projects and Solutions** category, select **VC++ Directories**, as shown in Figure 5-14.

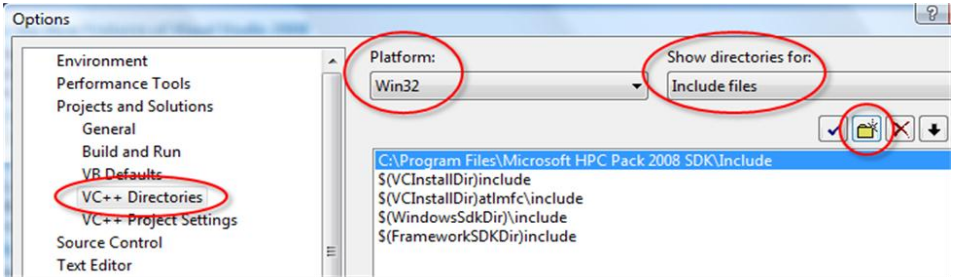



*Figure 5-14 Add Include files*

Let's assume the SDK install folder is C:\Program Files\Microsoft HPC Pack 2008 R2 SDK\. Select the Win32 platform, show directories for either "Include files" or "Library files", and click the small folder icon to add the following configuration values:

| | |
|---|---|
| Win32 \| Include files: | C:\Program Files\Microsoft HPC Pack 2008 R2\Include |
| Win32 \| Library files: | C:\Program Files\Microsoft HPC Pack 2008 R2\Lib\i386 |

Now switch to the x64 platform, and repeat with these values:

| | |
|---|---|
| x64 \| Include files: | C:\Program Files\Microsoft HPC Pack 2008 R2\Include |
| x64 \| Library files: | C:\Program Files\Microsoft HPC Pack 2008 R2\Lib\amd64 |

If the platform drop-down doesn't include "x64", exit Visual Studio, install the 64-bit compilers for Visual C++, and try again.

At the start of main (before any of the command-lines arguments are processed), add calls to **MPI_Init**, **MPI_Comm_size**, **MPI_Comm_rank**, and **gethostname**. For ease of debugging, declare the associated variables as globals, as shown in Code 5-9.

```
int   myRank;
int   numProcs;
char  host[256];

int main(int argc, char *argv[])
{
          MPI_Init(...);
          MPI_Comm_size(...);
          MPI_Comm_rank(...);
          gethostname(...);

          .
          .
          .

          MPI_Finalize();
          return 0;
}
```

*Code 5-9 Major statements in main function*

At the end of main (that is, just before the return), add the call to **MPI_Finalize**. Build, fix any syntax errors, and build again.

Now modify the startup sequence to output information about your run-time environment, as shown in Code 5-10.

```
cout << "   Processes:   " << numProcs << endl;
cout << "   Process " << myRank << " is running on '" << host << "'" <<
endl;
```

*Code 5-10 Output statements in the main function*

Build and run from within Visual Studio (Ctrl+F5), creating just one MPI process as shown in Figure 5-15. Record the first five convergence values; make a note that these are the correct values.

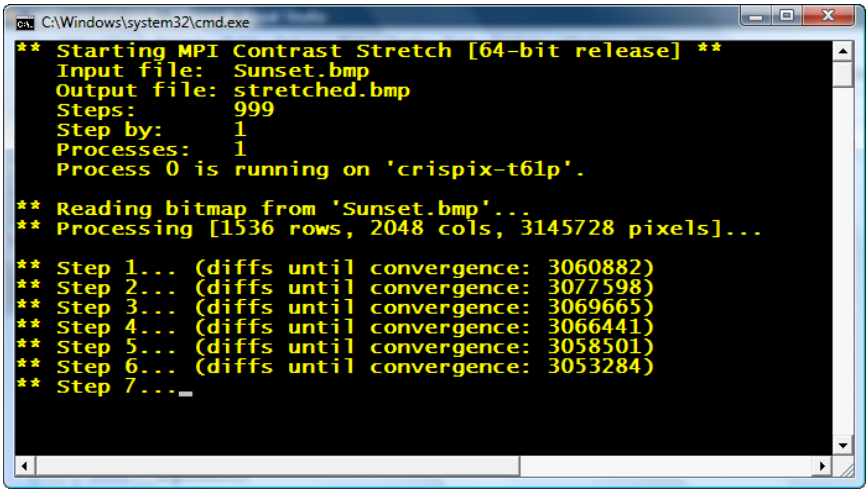



*Figure 5-15 Sequential version run result*

2. The MPI version will follow a standard data parallel design: (1) distribute the data, (2) process the data in parallel (communicating only when necessary), and (3) collect the results. Applying this design to our contrast stretching application (see Figure 5-16), the master process reads the image file and evenly distributes the image matrix to the worker processes, keeping a chunk to itself for processing. All processes then stretch their image chunk in parallel. As a worker completes, it sends its chunk back to the master, who inserts the chunk into the proper location of the resulting image. Finally, after all chunks have been collected, the master writes the newly stretched image back to disk.

   In a perfect world, parallel applications are developed step-by-step, starting with a correct solution and maintaining correctness as we go. We'll take this approach here, using WinDiff to compare the resulting image with the expected image.

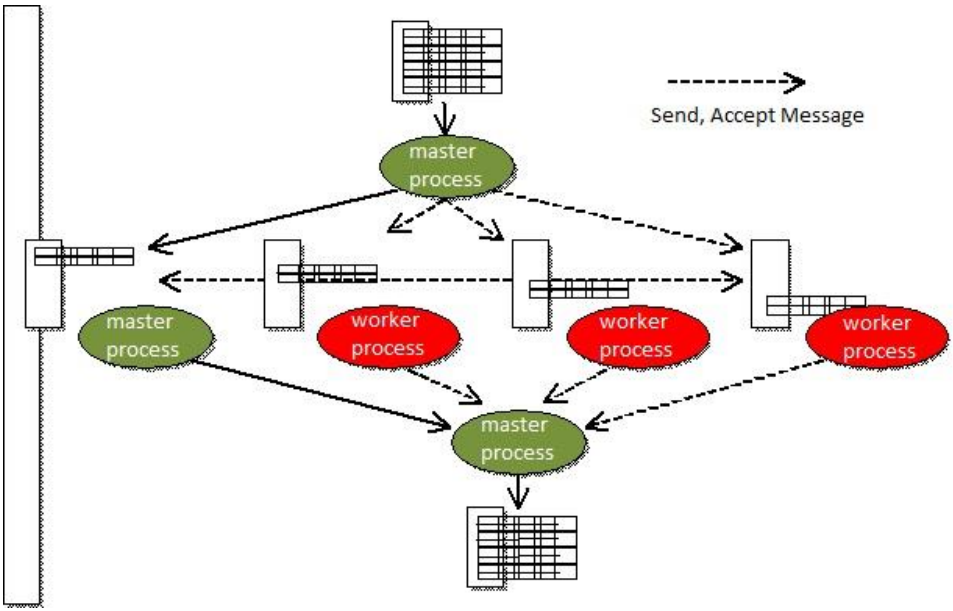



*Figure 5-16 MPI distributes data, parallel processes data, and collects results*

The first step is to distribute the image among the workers. To test for correctness, we'll skip the contrast stretching, and simply collect the image back on the master. If all is well, the resulting image file will be byte-for-byte identical to the original image file.

The master and workers will use **MPI_Send** and **MPI_Recv** to distribute and collect the image matrix. Recall that one of the parameters to **MPI_Send** and **MPI_Recv** is a datatype, conveying the type of elements being sent and received. This type is the struct **PIXEL_T**, defined in App.h. To use structured types with **MPI_Send** and **MPI_Recv**, the type must first be defined as an **MPI_Datatype** by all processes. For simplicity, a function to perform this duty is defined at the bottom of Main.cpp. Somewhere after the call to **MPI_Init**, add a call to this function **CreateMPIPixelDatatype** to define a new MPI datatype called **MPI_PIXEL_T**, as shown in Code 5-11.

```
// Create an MPI-based datatype to represent our PIXEL_T struct:

MPI_Datatype  MPI_PIXEL_T = CreateMPIPixelDatatype();
```

*Code 5-11 Create MPI pixel data type*

Just before the call to **MPI_Finalize**, all processes should free this datatype with a call to **MPI_Type_free**, as shown in Code 5-12.

```
MPI_Type_free(&MPI_PIXEL_T);

MPI_Finalize();

return 0;
```

*Code 5-12 Release pixel data*

3. We are now ready to distribute the image matrix across the worker processes. Start by locating in main the call to *ContrastStretch(…)*. Immediately above this line, we're going to add code to call a function to distribute the matrix. Immediately after, we're going to call a function to collect the results. Finally, we'll comment out the call to *ContrastStretch* to skip this step for now. Code 5-13 shows what you should enter.

```
          PIXEL_T  **chunk = NULL;

          int        myrows = 0;

          int        mycols = 0;


          chunk = DistributeImage(image, rows, cols, myrows, mycols,
MPI_PIXEL_T);  // distribute image, get back a chunk:


          assert(chunk != NULL);  // every process should have work to do
now:

          assert(rows > 0);   assert(cols > 0);

          assert(myrows > 0); assert(mycols > 0);


          // chunk = ContrastStretch(chunk, myrows, mycols, steps, stepby,
MPI_PIXEL_T);


          image = CollectImage(image, rows, cols, chunk, myrows, mycols,
MPI_PIXEL_T);  // collect chunks to get image:
```

*Code 5-13 Image distribution and collection*

Even though it is commented out, note that the call to *ContrastStretch* is now parameterized in terms of a chunk, not the entire matrix.

Next, add two source code files to your project, Distribute.cpp and Collect.cpp. In Distribute.cpp, stub out a definition of our **DistributeImage** function, as shown in Code 5-14.

```
          #include "app.h"


          PIXEL T **DistributeImage(PIXEL T **image, int &rows, int &cols,
int &myrows, int &mycols, MPI_Datatype MPI_PIXEL_T)

          {

                    return NULL;

          }
```

*Code 5-14 DistributeImage function*

In Collect.cpp, stub out our **CollectImage** function, as shown in Code 5-15.

```
          #include "app.h"


          PIXEL T **CollectImage(PIXEL T **image, int rows, int cols,
PIXEL_T **chunk, int myrows, int mycols, MPI_Datatype MPI_PIXEL_T)

          {

                    return NULL;

          }
```

*Code 5-15 CollectImage function*

Finally, define prototypes for these functions by adding the following declarations to App.h, as shown in Code 5-16.

```
          PIXEL_T **DistributeImage(PIXEL_T **image, int &rows, int &cols,
int &myrows, int &mycols, MPI_Datatype MPI_PIXEL_T);

          PIXEL_T **CollectImage(PIXEL_T **image, int rows, int cols,
PIXEL_T **chunk, int myrows, int mycols, MPI_Datatype MPI_PIXEL_T);
```

*Code 5-16 Function prototype definitions*

Build, and fix any syntax errors.

4. Before we start implementing these functions, let's finish our modifications to the main function. The crucial observation is that N processes are now executing the program, 1 master and N-1 workers. Do we really want all N processes opening and reading the input file? Outputting to stdout? Starting and stopping the clock? Writing the output file? No. Generally, these tasks are performed by the master process only[1]. By convention, the master process is the first process launched—the rank 0 process. Go back through main line by line, and decide whether all processes should execute a particular line of code, or just the master process. For example, all processes must initialize MPI, process the command-line arguments, and define the MPI datatype **MPI_PIXEL_T**. On the other hand, only the master process should read the input file, as shown in Code 5-17.

```
          BITMAPFILEHEADER bitmapFileHeader;

          BITMAPINFOHEADER bitmapInfoHeader;

          PIXEL_T **image = NULL;

          int       rows = 0, cols = 0;


          if (myRank == 0)

          {

                    cout << "** Reading bitmap from '" << infile << "'..."
<< endl;

                    image = ReadBitmapFile(infile, bitmapFileHeader,
bitmapInfoHeader, rows, cols);

                    if (image == NULL)

                    {

                              cout << endl;

                              cout << "** Failed to open image file,
halting..." << endl;

                              return 1;

                    }

          }
```

*Code 5-17 Master process initializes reading the input file*

[1] This is certainly not the only possible design. For example, we could have all processes read the input file and thus avoid the data distribution step. The potential disadvantage is that if the input file is on a network share, the share becomes a bottleneck. We could solve this by assuming the input file is local to each process, but then we complicate application deployment to the cluster. These are common trade-offs in the design of cluster-wide, HPC applications.

The problem of distributing the image matrix is now encapsulated within **DistributeImage**. This function takes the image and its size (*rows* and *cols*), and returns the chunk (of size *myrows* and *mycols*) that the calling process should stretch, as shown in Code 5-18.

```
chunk = DistributeImage(image, rows, cols, myrows, mycols, MPI_PIXEL_T);
```

*Code 5-18 DistributeImage*

All processes are involved in this step, and should be calling this function from main. However, the role that a process plays during distribution depends on whether they are the master or a worker. More subtle is the fact that the *validity* of the parameters to **DistributeImage** depends on the process. Since the master process reads the input file, when the master calls **DistributeImage**, the parameters *image*, *rows*, and *cols* contain meaningful values. For the worker processes, these parameters reflect their initialized values of NULL, 0, and 0, respectively.

Within **DistributeImage**, the master process is responsible for first distributing the size of the matrix, since this is unknown to the workers. Then it distributes the individual chunks to each of the workers. This is accomplished through a series of calls to **MPI_Send**. Finally, the master makes a copy of the chunk it is supposed to process—the first chunk of the original matrix. Code 5-19 shows the code you should enter.

```
          PIXEL_T **chunk = NULL;  // best practice with MPI: initialize
all variables

          int       tag   = 0;

          int       params[2] = {0, 0};


          cout << myRank << " (" << host << "): Distributing image..." <<
endl;


          if (myRank == 0)  // Master:

          {

                    int rowsPerProc  = rows / numProcs;

                    int leftOverRows = rows % numProcs;


                    params[0] = rows;

                    params[1] = cols;


                    for (int dest=1; dest < numProcs; dest++)  // send
each worker the size of the image:

                              MPI_Send(params,
sizeof(params)/sizeof(params[0]), MPI_INT, dest, tag, MPI_COMM_WORLD);


                    for (int dest=1; dest < numProcs; dest++)  // now send
the actual data chunk (skip over first chunk + leftover):

                              MPI_Send(image[leftOverRows +
dest*rowsPerProc], rowsPerProc*cols, MPI_PIXEL_T, dest, tag,
MPI_COMM_WORLD);
```

```
                    // okay, master is responsible for the first chunk +
any leftover rows:

                    myrows = rowsPerProc + leftOverRows;

                    mycols = cols;

                    chunk = New2dMatrix<PIXEL_T>(myrows+2, mycols);  //
why 2 extra rows?  See contrast stretching function

                    memcpy_s(chunk[1], myrows*mycols*sizeof(PIXEL_T),
image[0], myrows*mycols*sizeof(PIXEL_T));

          }
```

*Code 5-19 Master process assigns the image to worker processes and itself*

The master retains the first chunk for itself to process, including any extra rows if the number of processes does not evenly divide the size of the matrix. You'll notice that the allocated chunk contains 2 extra rows; we'll explain when we discuss the MPI-based contrast stretching algorithm.

5. During distribution, the worker processes receive the size of the image, compute their chunk size, allocate memory, and then receive the actual chunk data. This can be coded as shown in Code 5-20.

```
else  // Workers:

          {

                    MPI_Status status;


                    MPI_Recv(params, sizeof(params)/sizeof(params[0]),
MPI_INT, 0 /*master*/, tag, MPI_COMM_WORLD, &status);


                    rows = params[0];

                    cols = params[1];

                    myrows = rows / numProcs;  // size of our chunk to
process:

                    mycols = cols;


                    chunk = New2dMatrix<PIXEL_T>(myrows+2, mycols);  //
why 2 extra rows?  See contrast stretching function


                    MPI_Recv(chunk[1], myrows*mycols, MPI_PIXEL_T, 0
/*master*/, tag, MPI_COMM_WORLD, &status);

          }


          return chunk;
```

*Code 5-20 Worker process code*

All processes finish the distribution step by returning their chunk back to the caller. Build, and fix any syntax errors. If you get errors complaining of undefined symbols for *myRank*, *numProcs*, or *host*, recall that these are defined in Main.cpp but you also need to inform the compiler. Define these as external symbols by adding the following to App.h, as shown in Code 5-21.

```
extern int  myRank;

extern int  numProcs;

extern char host[256];
```

*Code 5-21 Global variables*

The build should now succeed without error.

6. We are almost ready to run and test our handiwork. What remains is to implement the **CollectImage** function, in which the workers send their chunks to the master, and the master collects and builds the resulting image. I'm going to leave the details to you, but in short the idea is to reverse the steps you performed in **DistributeImage**. The workers are simple: they send their chunk back to the master process, though keep in mind that the data starts at chunk[1], not chunk[0] (due to those 2 extra rows we haven't explained yet). The master is likewise straightforward: it receives each worker chunk back into *image*, the location of which depends on the worker's rank. The master must also copy their own chunk back into *image* (likewise starting at chunk[1]). Once the image is complete, the master returns the image back to the caller; the workers should return NULL. If you need help, recall that a complete solution is available in *Chapter 5\Solutions\MPI\MPIContrastStretch*.
7. Now it's time to run and see what happens. To adequately test our work, we need to run with multiple processes—first with say 2, then 4, and then 7 (an odd number of processes will trigger the case of the image not dividing evenly and thus *leftover* rows). As discussed earlier, we'll need to minimize Visual Studio, open a console window ("black screen"), and launch the application using Mpiexec. Go ahead and open a console window (Start, cmd.exe), and navigate to the directory containing your .exe (for example, *Exercises\MPI\MPIContrastStretch\x64\release*). You'll need an image to process, so copy Sunset.bmp from the *MPIContrastStretch* subdirectory. Now run and start 2 processes. The application will run quickly since we commented out the stretching step, as shown in Figure 5-17.

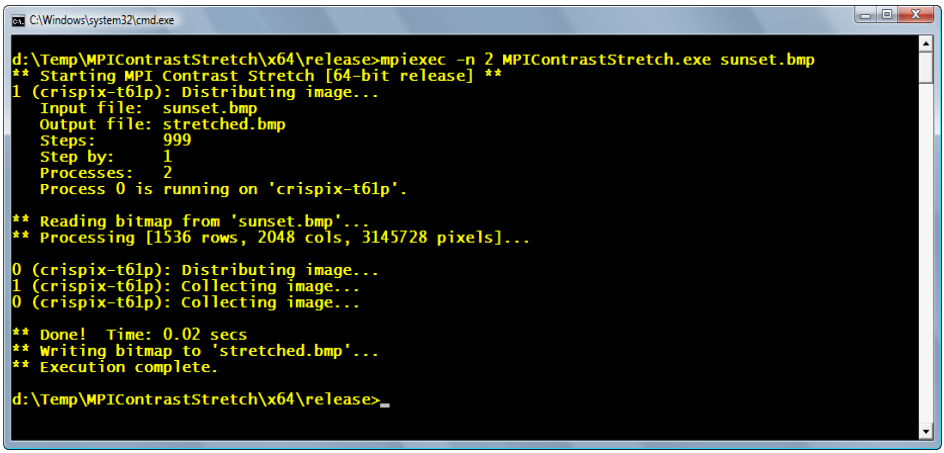



*Figure 5-17 Run results*

To ensure that the distribution and collection are working properly, compare the input file Sunset.bmp to the output file Stretched.bmp. Do a byte-for-byte comparison, using a tool such as WinDiff (available in *Misc*\). There should be no reported differences. Repeat for 4 and 7 processes.

8. The last step of the conversion to MPI is the contrast stretching portion of the application. In Main.cpp, uncomment the call to ContrastStretch, as shown in Code 5-22.

```
chunk = ContrastStretch(chunk, myrows, mycols, steps, stepby,
MPI_PIXEL_T);
```

*Code 5-22 Uncomment the call to ContrastStretch in Main.cpp*

Fix the function prototype in App.h to match the new definition (note the additional parameter at the end), as shown in Code 5-23.

```
PIXEL T **ContrastStretch(PIXEL T **image, int rows, int cols, int steps,
int stepby, MPI_Datatype MPI_PIXEL_T);
```

*Code 5-23 Add ContrastStretch function prototype*

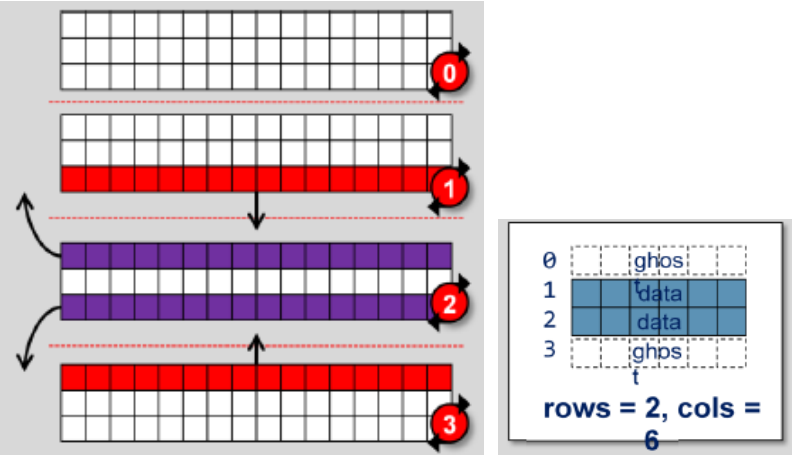
Now we'll focus our attention on the implementation in ContrastStretch.cpp. Open the source code file, and update the function header to match the new prototype. Add an output statement for debugging purposes, and update the allocation of the temporary matrix to include 2 extra rows, as shown in Code 5-24.

```
           PIXEL T **ContrastStretch(PIXEL T **image, int rows, int cols,
int steps, int stepby, MPI_Datatype MPI_PIXEL_T)

           {

                     cout << myRank << " (" << host << "): Processing " <<
rows << " rows, " << cols << " cols..." << endl;

                     PIXEL_T **image2 = New2dMatrix<PIXEL_T>(rows+2, cols);
```

*Code 5-24 Add an output statement for debugging purposes, and update the allocation of the temporary matrix to include 2 extra rows*

Okay, so why the 2 extra rows? Recall the basics of the contrast stretching algorithm: for each pixel P, we update P based on its 8 surrounding neighbors. Based on this algorithm, the first row must be treated as a special case, since the first row has no neighbors to the North. Likewise, the last row has no neighbors to the South, the first column no neighbors to the West, and the last column no neighbors to the East. The solution is to ignore processing of these rows and columns.

When we distribute the image matrix, we run into the same problem with each chunk. In particular, for each process P, P is unable to process their first and last rows since the neighboring rows are in other processes. And unlike the first and last columns, processing of the first and last rows cannot be skipped without changing the semantics of the algorithm. This holds true for all processes except the first (rank 0) and the last (rank numProcs-1). The implication is that during each step of the algorithm, processes must swap their first and last rows with their neighboring processes, as shown in Figure 5-18.




*Code 5-18 Ghost rows between processes*

For example, consider process 2 in a run with 4 processes. As shown to the right, process 2 will need to send its first row to process 1, and its last row to process 3. Likewise, process 2 will need to receive the last row from process 1, and the first row from process 3. These two extra rows are typically referred to as *ghost* rows, and will need to be stored locally on process 2. Thus, when we allocate the memory for process 2's chunk, we include room for the 2 ghost rows; likewise for allocation of the matching temporary matrix. For example, if process 2 was responsible for 2 rows and 6 columns of data, its chunk would be 4 x 6 in size (as shown to the left), with the data starting in row 1.

9. Locate the while loop, and add code at the top of the loop body to exchange ghost rows. First, for each process, send its last row "down" and receive the corresponding row from its neighbor as its upper ghost row, as shown in Code 5-25.

```
          if (myRank < numProcs-1)  // everyone send last row down (except
last):

                    MPI Send(image[rows], cols, MPI PIXEL T, myRank+1, 0
/*tag*/, MPI_COMM_WORLD);

          if (myRank > 0)  // now everyone receive that ghost row from
above (except first):

                    MPI_Recv(image[0], cols, MPI_PIXEL_T, myRank-1, 0
/*tag*/, MPI_COMM_WORLD, &status);
```

*Code 5-25 Receive the corresponding row from its neighbor as its upper ghost row*

The second step is for each process to send its first row "up" and receive the corresponding row from its neighbor as its lower ghost row, as shown in Code 5-26.

```
          if (myRank > 0)  // everyone send first row up (except first):

                    MPI Send(image[1], cols, MPI PIXEL T, myRank-1, 0
/*tag*/, MPI_COMM_WORLD);

          if (myRank < numProcs-1)  // and everyone receive that ghost row
from below (except last):

                    MPI_Recv(image[rows+1], cols, MPI_PIXEL_T, myRank+1, 0
/*tag*/, MPI_COMM_WORLD, &status);
```

*Code 5-26 Receive the corresponding row from its neighbor as its lower ghost row*

Add the necessary variable declarations (for example, *status*), and build. But don't run just yet.

10. With the ghost rows now set, we are ready to perform one pass of the contrast stretching algorithm. However, the nested for loops that process each pixel require a subtle update. Every process should start processing at the first data row of its chunk—that is, row 1—except the first process. The first process has no data in row 0 (its upper ghost row), since row 1 is the start of the image. So the first process should start processing at row 2. Similarly, every process should process up to and including the last row of its chunk—that is, up to and including the index *rows*. The last process, however, should stop processing at *rows-1*, since rows denotes the last row of the image. Let's compute this explicitly for each process, and update the for loops accordingly, as shown in Code 5-27.

```
          int firstRow = 1;  // this holds true for everyone but master &
last worker:

          int lastRow  = rows;


          if (myRank == 0)   // master: does not process first row

                    firstRow = 2;


          // NOTE: master & last worker could be the same if numProcs ==
1, so don't tie if tests together.


          if (myRank == numProcs-1)  // last worker: does not process last
row

                    lastRow = rows-1;


          for (int row = firstRow; row <= lastRow; row++)  // from first
row to last row, inclusive:

          {

                    for (int col = 1; col < cols-1; col++)

                    {
```

*Code 5-27 Contrast stretching MPI algorithm*

Okay, just one more issue before we run and test. When the for loops end, a convergence test is performed to see if the stretching has reached a steady state (in which case the algorithm can terminate), as shown in Code 5-28.

```
                    }

          }


          cout << " (diffs until convergence: " << diffs << ")" << endl;

          converged = (diffs == 0);
```

*Code 5-28 Convergence test*

Since the image is distributed, the computed *diffs* is now a local value. But the algorithm calls for a global value, terminating if and only if the total number of differences is 0. This implies that the processes need to communicate their local diffs to the master, who will sum and redistribute the total value. This is going to require a number of sends and receives, as shown in Code 5-29.

```
          if (myRank > 0)  // workers:

          {

                    MPI_Send(&diffs, 1, MPI_LONG_LONG, 0 /*master*/, 0
/*tag*/, MPI_COMM_WORLD);

                    MPI_Recv(&diffs, 1, MPI_LONG_LONG, 0 /*master*/, 0
/*tag*/, MPI_COMM_WORLD, &status);

          }

          else  // master:

          {

                    long long  temp;


                    for (int src=1; src < numProcs; src++)  // receive
from workers:

                    {

                              MPI_Recv(&temp, 1, MPI_LONG_LONG,
MPI_ANY_SOURCE, 0 /*tag*/, MPI_COMM_WORLD, &status);

                              diffs += temp;  // add to our own diffs

                    }


                    for (int dest=1; dest < numProcs; dest++)  // now send
out total diffs to workers:

                              MPI_Send(&diffs, 1, MPI_LONG_LONG, dest, 0
/*tag*/, MPI_COMM_WORLD);

          }


          cout << " (diffs until convergence: " << diffs << ")" << endl;

          converged = (diffs == 0);
```

*Code 5-29 Collect the diffs variable*

Finally, since the nested for loops changed, we also need to update the corresponding nested for loops that copy the contents of the temporary matrix back into the image matrix. In particular, the rows to copy have changed, as shown in Code 5-30.

```
          for (int row = firstRow; row <= lastRow; row++)

                    for (int col = 1; col < cols-1; col++)

                              image[row][col] = image2[row][col];
```

*Code 5-30 Update temporary matrix copy*

Build and fix any syntax errors. If you haven't already, you should also update the output statements to include process rank and hostname.

11. We now have a complete, MPI-based, contrast stretching application. Let's run and test. Minimize Visual Studio, open up a console window, and launch the application with Mpiexec. Start with 2 processes, and run for 75 steps, as shown in Figure 5-19.

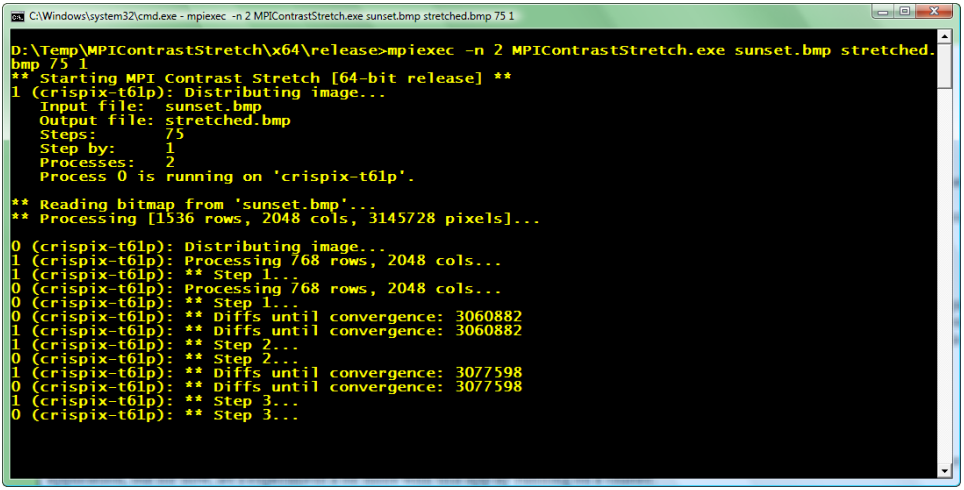



*Figure 5-19 MPI program run result*

First, make sure the first 5 convergence values reported are the same as the sequentially generated values (you wrote them down, right?). Then, when the run finishes, compare your output file Stretched.bmp to the sequentially generated image file Sunset-75-by-1.bmp in *Misc*\. There should be no differences. Repeat for 4 processes, 7 processes, and 1 process. Why test against a single MPI process? If possible, it's generally a good idea for your MPI application to run correctly with just one process; this allows correct execution if the application is run on hardware with only a single core or a cluster with only 1 core/socket/node available.

## 5.5 Running MPI Applications on Windows HPC Server 2008 R2

For the purpose of this chapter, let's assume the following hypothetical cluster:

| | |
|---|---|
| Name of head node: | headnode |
| Name of compute nodes: | compute1, compute2 … |
| Run-as credential for job execution: | domain\hpcuser |
| Network share accessible by all nodes: | \\<headnode>\Public |
| Network share on every node (mapped to C:\Apps) | \\<headnode>\Apps. \\compute1\Apps, \\compute2\\Apps. … |

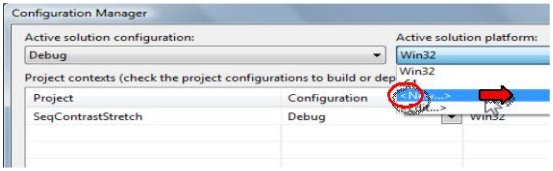
Users of the cluster are assumed to have full read/write access to these network shares, as well as access to the cluster itself (replace <headnode> with your cluster's head node name).

### 5.5.1 Submitting a job to the cluster and monitoring the job

1. If you haven't already done so, build a 64-bit release version of your application for deployment to the cluster. This is selected via Visual Studio's Standard toolbar, as shown in Figure 5-20.

*Figure 5-20 Set x64 as the target platform*

If **x64** doesn't appear in the toolbar drop-down, add this configuration to your project via the **Build** menu, **Configuration Manager** (if x64 is not an option in the **New Solution Platform** dialog (shown on the right side of Figure 5-21), then you need to exit and install the 64-bit compilers for Visual C++), as shown in Figure 5-21.



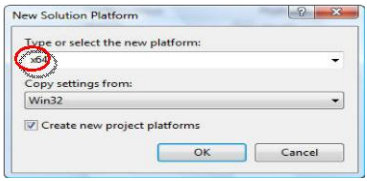




*Figure 5-21 Configure the solution*

2. After the build, your .EXE can be found in x64\release\. Deploy your application by first creating a unique subdirectory on the cluster's public share, for example, \\headnode\Public\DrJoe . Copy your 64-bit release .EXE to this subdirectory, along with any input files.

> **Note:** The use of network shares is generally the most convenient method of deployment. However, network shares can become a performance bottleneck, for example, with large input or output files. In this case, consider deploying to a local directory (C:\Apps) on each node, taking advantage of the fact that each node has a network share \\node\Apps that maps to this local directory (replace <headnode> with your cluster's head node name).

3. If you have administrative rights on the cluster, start up the Microsoft HPC Pack Cluster Manager and click the Job Management tab (as shown in the bottom-left corner of Figure 5-22); this plug-in allows you to submit jobs as well as monitor the cluster. Otherwise, assuming you have user rights on the cluster, start up the Microsoft HPC Pack Job Manager (shown in Figure 5-22). If this is the first time, connect to the cluster by supplying the head node (as shown in Figure 5-23).

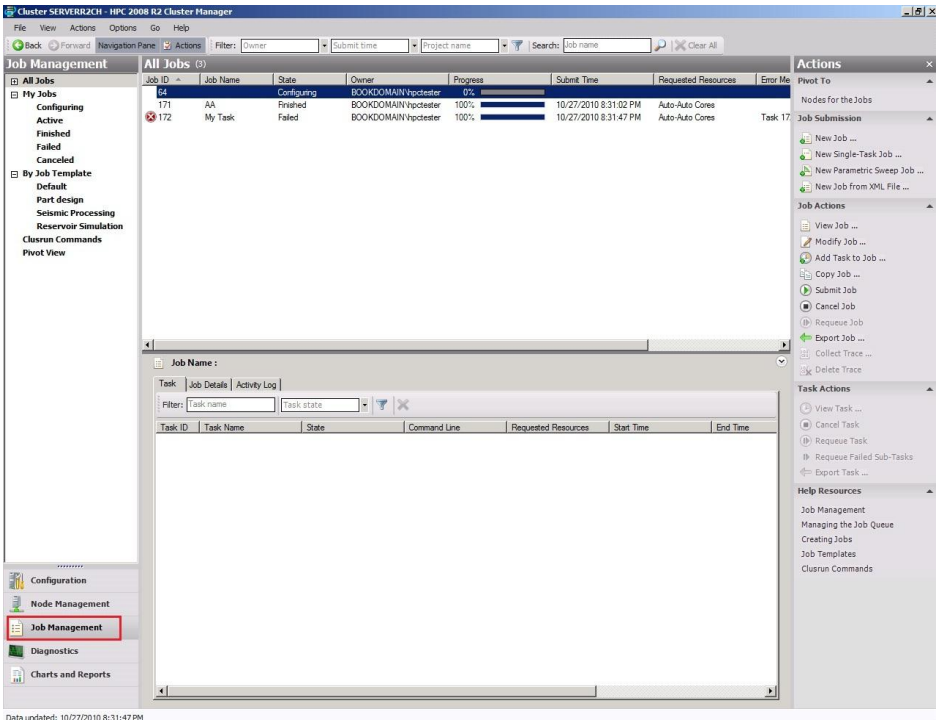



*Figure 5-22 Cluster Manager Job Management tab*

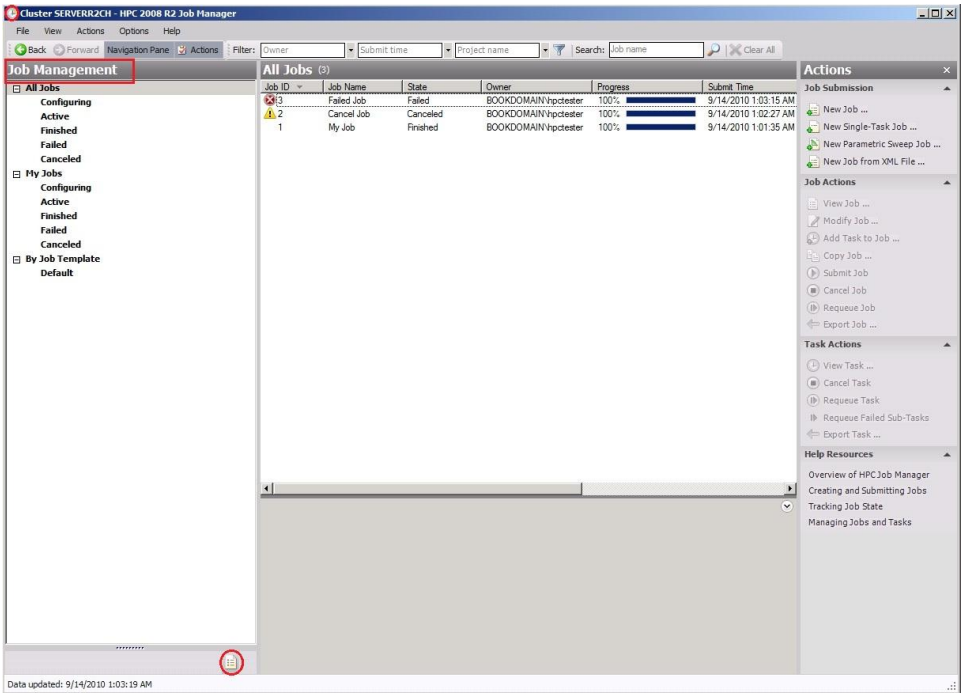



*Figure 5-23 Job Manager*

4. Create a new job via the Actions menu, Job Submission > Create New Job (dialog is shown to right, click for full-size image). Supply a job name, a project name (optional), and then decide how many execution cores you need to run your job. If this is an MPI run, set the minimum and maximum to the range of cores you can effectively use. Note that if you want to run on N cores, don't be afraid to set both the min and max to N.

   You can schedule a job by other types of resources, for example, Node, Socket, or Core. A node refers to an entire compute node. A socket refers to physical CPU chip in a node. A core refers to an execution core in a socket. For example, a *dual quadcore PC* is a single node with 2 sockets and 4 cores per socket, for a total of 8 execution cores.

   Finally, if this is a performance run, select the check box for **Use assigned resources exclusively for this job**. This maximizes your performance, but may waste resources from the perspective of other cluster users. Don't submit just yet.

5. Once the job is configured, you need to add one or more tasks—tasks are the actual executable units of work scheduled on the compute nodes. Let's assume that this is a simple job with just one task to execute your application. Click Task List in the new job window, and then click Add to add a new task to the job.

   Configure the task by assigning a name (optional), and then specifying the **Command line**, as shown in Figure 5-24. For an MPI application, you need to replace Task.exe with:

   mpiexec SQL_MPIContrastStretch.exe input.bmp output.bmp 25 5

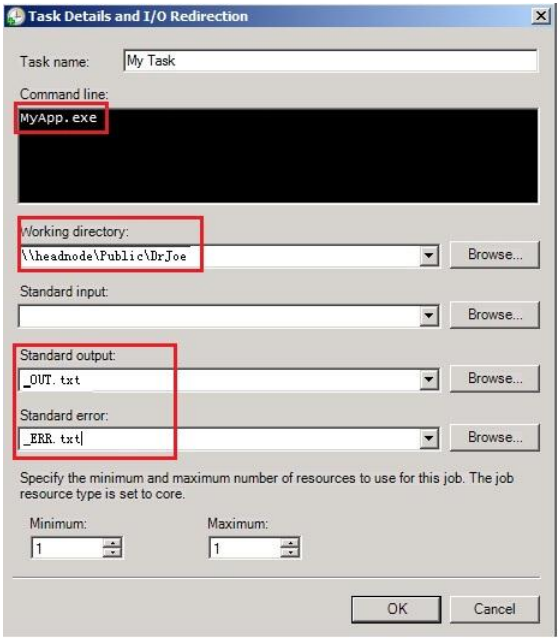



*Figure 5-24 Configure the MPI task*

Next, set the **Working directory** to the location where you deployed the .EXE (for example, \\<headnode>\Public\DrJoe, replace <headnode> with your head node name or C:\Apps if you deployed locally on each node). Redirect **Standard output** and **Standard error** to text files as shown in Figure 5-24; these capture program output and error messages, and will be created in the working directory (these files are very handy when troubleshooting). Finally, select the minimum and maximum number of execution cores to use for executing this task, as shown in Figure 5-25. The range is constrained by the values set for the overall job: use a min/max of 1 for sequential apps, the number of cores on a node for OpenMP apps, and a range of cores for MPI apps. Click **Save** to save the configuration.

6. You should now be back at the job creation window, with a job ready for submission. Now submit the job to the cluster by clicking Submit. You'll be prompted for the job's "Run-as" credentials, that is, the username and password for the identity under which this job will execute on the compute nodes. Enter the password and click OK, as shown in Figure 5-26.

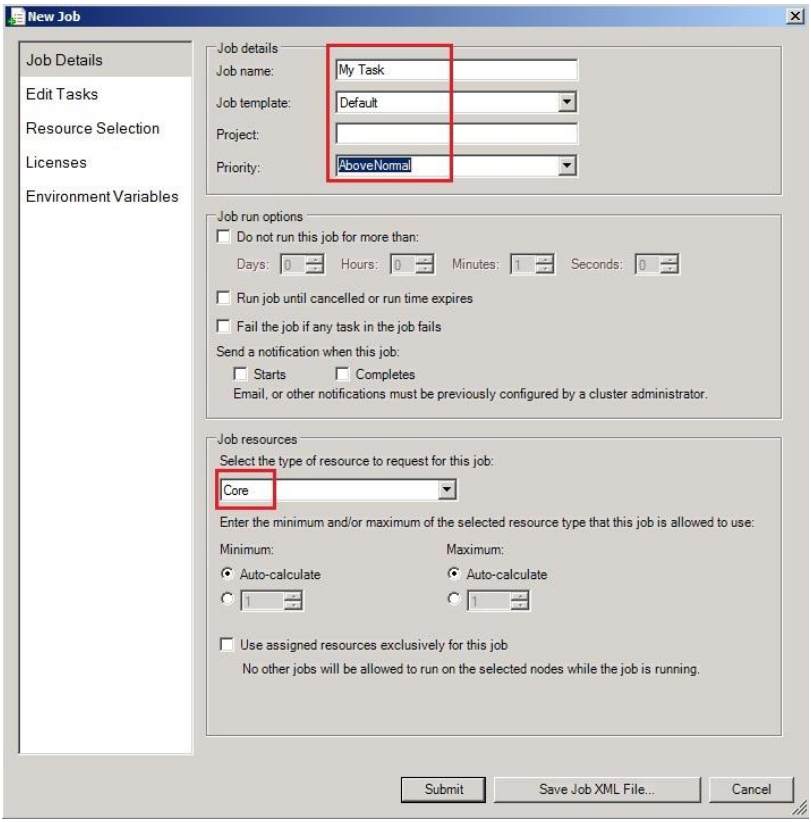



*Figure 5-25 Configure Job Detail*

*Figure 5-26 Specify User name and Password*

7. Bring up HPC Server's Heat Map so that you can visualize how resources are being used by the cluster during job execution. Submit another job, and then switch to the Node Management tab. Click Heat map, and monitor CPU usage. Here's a screen snapshot of my mini-cluster with 4 cores, running at 100%, as shown in Figure 5-27.

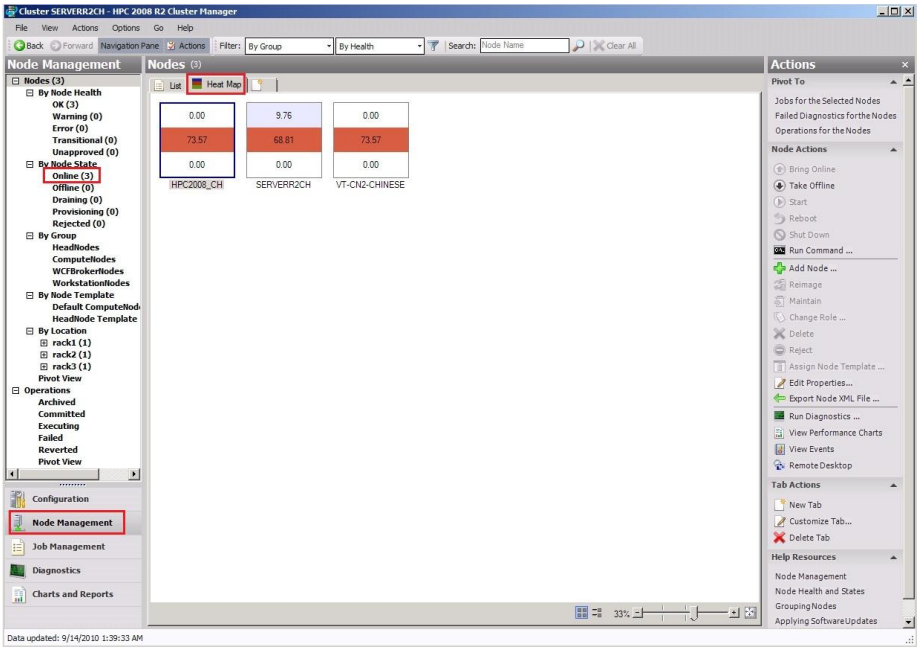



*Figure 5-27 Node Heat Map*

# Chapter 6 Manage the Windows HPC Server 2008 R2 Cluster

The previous two chapters described how to use the Windows® HPC Server 2008 R2 programming interface to write services and Message Passing Interface (MPI) applications, and how to use the job scheduling system to submit jobs. This chapter focuses on managing computer resources as follows:

- How to deploy a cluster from bare metal
- When the application is running on the cluster, how to effectively manage the cluster resources to enable the user application to run reliably and safely
- How to set the scheduling policy, to provide services under the convention in the shared environment
- How to find fault and recover the system as soon as possible
- How to use historical data to support decisions in cluster expansion planning

Chapter highlights:

- Cluster deployment
- Configuration of the scheduling system
- Monitoring of workload on the computing nodes
- Fault diagnosis
- Generation of built and custom reports

## 6.1 Introduction

Figure 6-1 shows a cluster management cycle. Administrators get the new computers, fix them on the rack, deploy the new computers, set up the cluster, and configure the cluster. When the cluster goes into production, administrators need to monitor the cluster compute nodes, head node, and broker nodes. When a fault is found, administrators needs to troubleshoot and diagnose the problem. Administrators need to periodically check the cluster and work with users to make decisions on expansion. The cycle repeats.

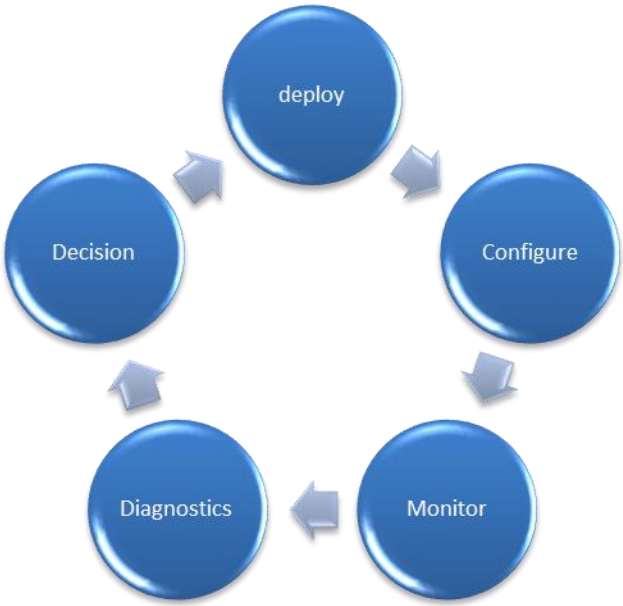



*Figure 6-1 Cluster management life cycle*

This chapter describes the common scenarios in the life cycle of cluster management, and it describes how to use Microsoft® HPC Pack 2008 R2 to complete tasks in a scenario.

## 6.2 Cluster Deployment

### 6.2.1 Deploy the Head Node

Follow the steps in Chapter 1 to install HPC Pack 2008 R2 and go through the to-do list.

### 6.2.2 Create a Node Template

The last step in the to-do list is to create a node template. Here are the demo steps to create a node template for compute nodes with operating system images:

1. Insert the Windows Server® 2008 R2 DVD.
2. Select **Create a node template** from the to-do list. The Create Node Template Wizard appears, as shown in Figure 6-2.

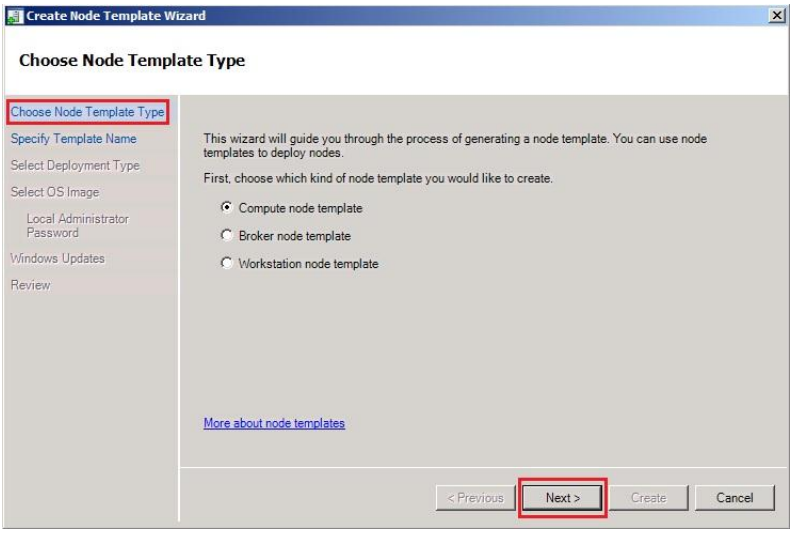



*Figure 6-2 Create a node template*

3. On the **Choose Node Template Type** page, click **Compute node template**, and then click **Next**.
4. On the **Specify Template Name** page, use the default name, and then click **Next**.
5. On the **Select Deployment Type** page, click **With operating system**, as shown in Figure 6-3.

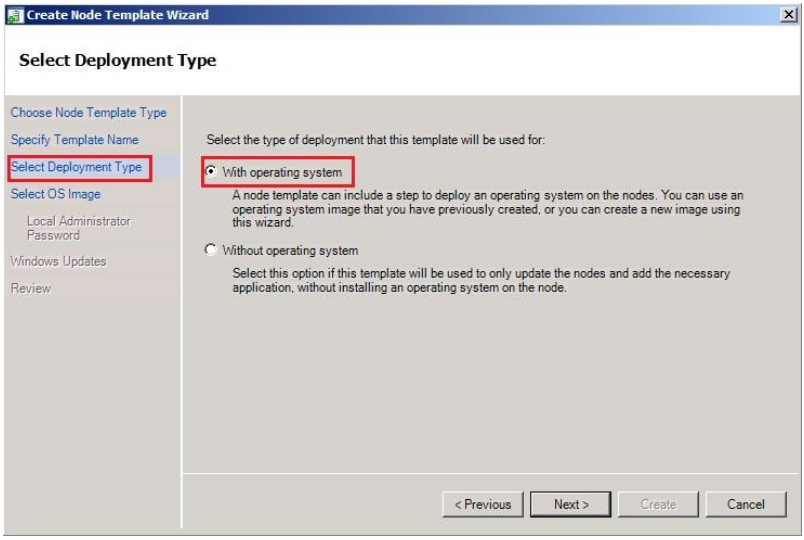



*Figure 6-3 Create a node template with operating system image*

6. Click **Next**.
7. On the **Select OS Image** page shown in Figure 6-4, click **Add Image**.

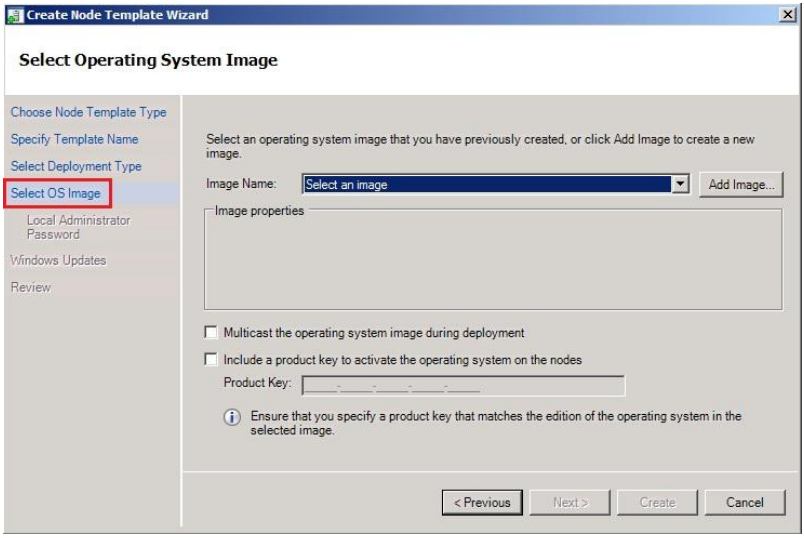



*Figure 6-4 Select operating system image*

8. In the **Add Operating System Image** dialog box shown in Figure 6-5, click **Browse**.

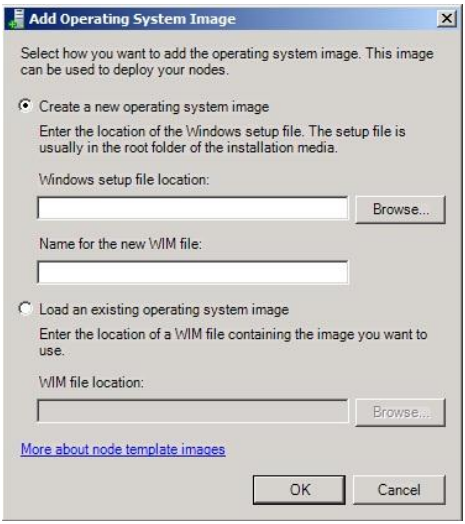



*Figure 6-5* ***Add Operating System Image*** *dialog box*

9. Browse to the setup path of Windows Server 2008 R2, select **Setup.exe**, and then click **Open**. In the text box for the Windows Imaging Format (WIM) file name, type **Windows 2008 R2**, as shown in Figure 6-6.

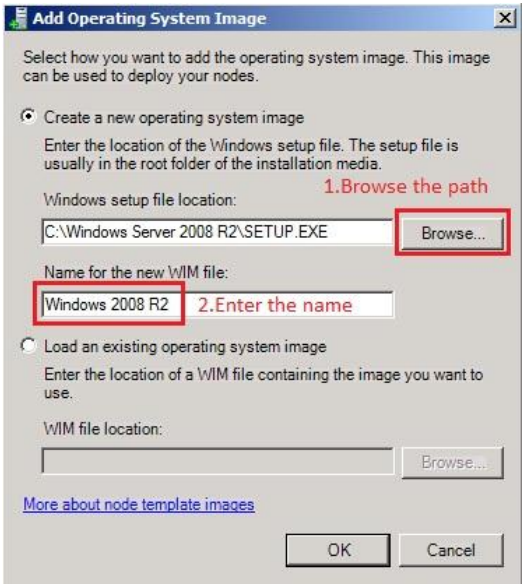



*Figure 6-6 Choose the Setup.exe location and set the WIM file name*

10. Click **OK** and the progress indicator appears, as shown in Figure 6-7.

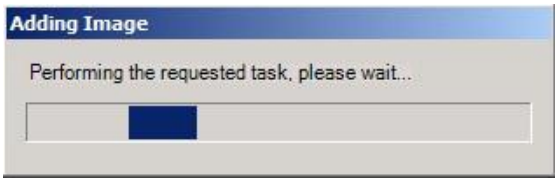



*Figure 6-7 Progress for adding an image*

11. After the process of adding an image finishes, on the **Select OS Image** page, choose the operating system image. Click the arrow, and then select **Windows 2008 R2.WIM\Windows Server 2008 R2 SERVERENTERPRISE** from the list, as shown in Figure 6-8. Note that one image file may have multiple operating system images, such as Windows 2008 R2.WIM\Windows Server 2008 R2 SERVERSTANDARD and Windows 2008 R2.WIM\Windows Server 2008 R2 SERVERSTANDARD.

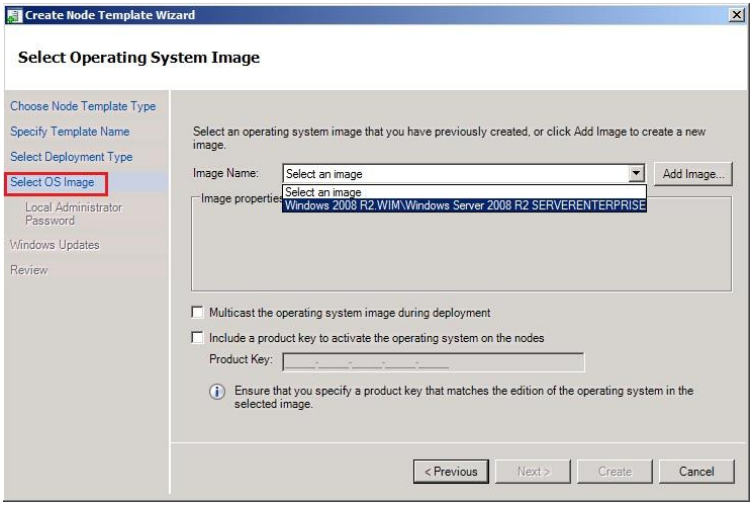



*Figure 6-8 Add the operating system image*

12. If the **Include a product key to activate the operating system on the nodes** check box is selected, enter the corresponding product key.
13. Click **Next**.

14. On the **Specify Local Administrator Password for Node** page, click **Use a specific password**, and then enter the password twice, as shown in Figure 6-9.

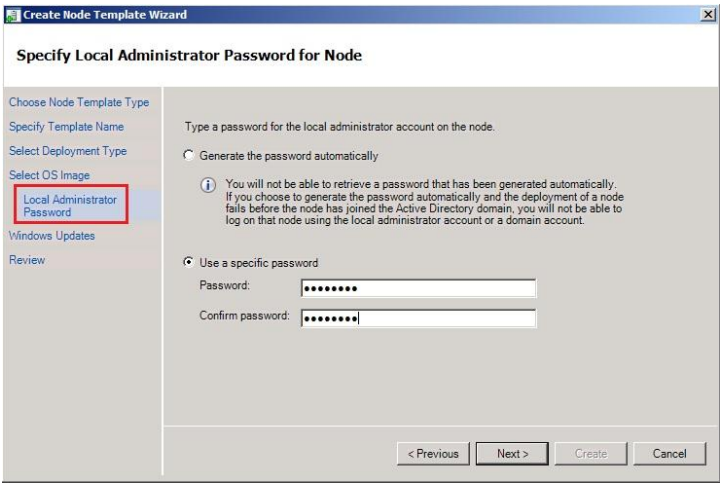



*Figure 6-9 Specify local administrator password for the node*

15. Click **Next**.
16. On the **Windows Updates** page, click **Next**.
17. On the **Review** page, check that the settings are correct, and then click **Create**, as shown in Figure 6-10.

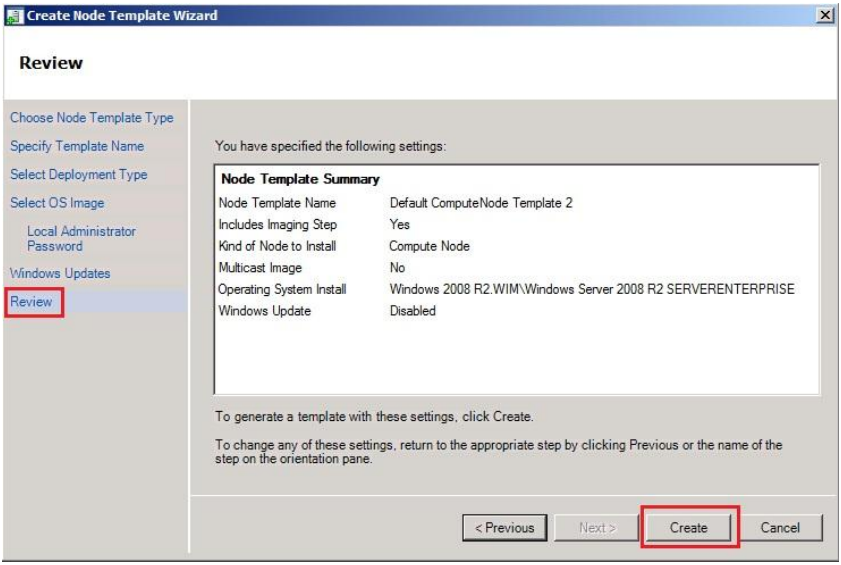



*Figure 6-10 Create Node Template Wizard*

18. After the node template is created, the navigation pane appears in the Admin Console, as shown in Figure 6-11.

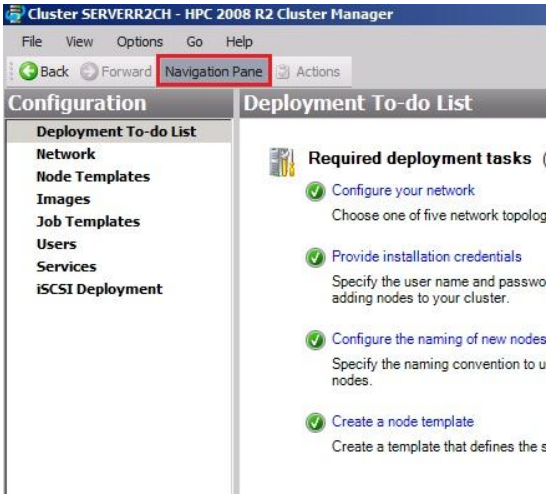



*Figure 6-11 Navigation pane appears*

> The story in the product design: Why does the navigation pane appear only after the to-do list is completed?
>
> In the previous version of Microsoft HPC Pack 2008 beta, before the to-do list is completed, users can view the navigation pane and do other configurations besides the to-do list, which may lead to invalid configurations for the cluster. To avoid such errors, the product now displays the navigation pane only after the to-do list is completed. This design follows the rule to lead customers to the path of fewest errors.

19. Click **Options**, and then click **Deployment Settings**, as shown in Figure 6-12.

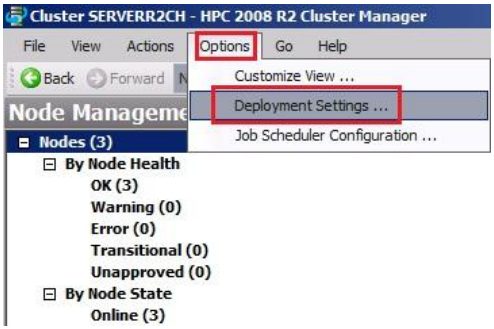



*Figure 6-12 Choose the settings for deployment*

20. In the **Deployment Settings** dialog box, click **Respond to all PXE requests**, as shown in Figure 6-13, and then click **OK**.

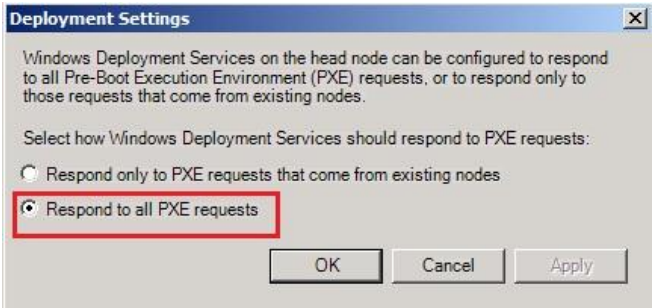



*Figure 6-13 Choose to which nodes the Preboot Execution Environment (PXE) responds*

## 6.2.3 Start Compute Nodes

> **Notice**
>
> Compute nodes must have one network interface card (NIC) connected to the private network. Network PXE should be set as the first boot option in basic input/output system (BIOS), and the nodes are in power-off state.

1. Turn on the compute nodes. The nodes enter the network boot PXE and search the Dynamic Host Configuration Protocol (DHCP) server on the network to obtain IP addresses, as shown in Figure 6-14.

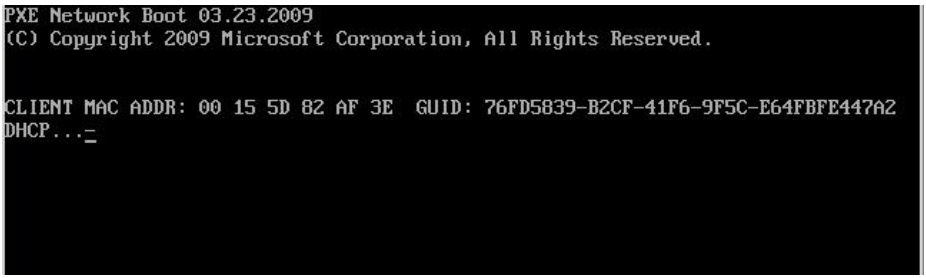



*Figure 6-14 Compute nodes enter network boot PXE and apply IP addresses*

2. After the IP address is obtained, the compute nodes wait for the head node to approve the deployment requests, as shown in Figure 6-15.

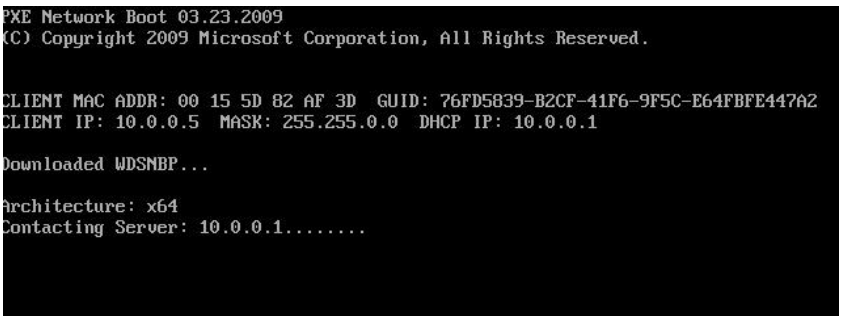



*Figure 6-15 IP address obtained as the compute node waits for the approval from the head node*

3. At this time, you can approve the deployment requests from the Admin Console. In the navigation pane, under **Node Management**, the unapproved nodes appear in the node list, categorized as **Nodes\By Node Health\Unapproved**, as shown in Figure 6-16.

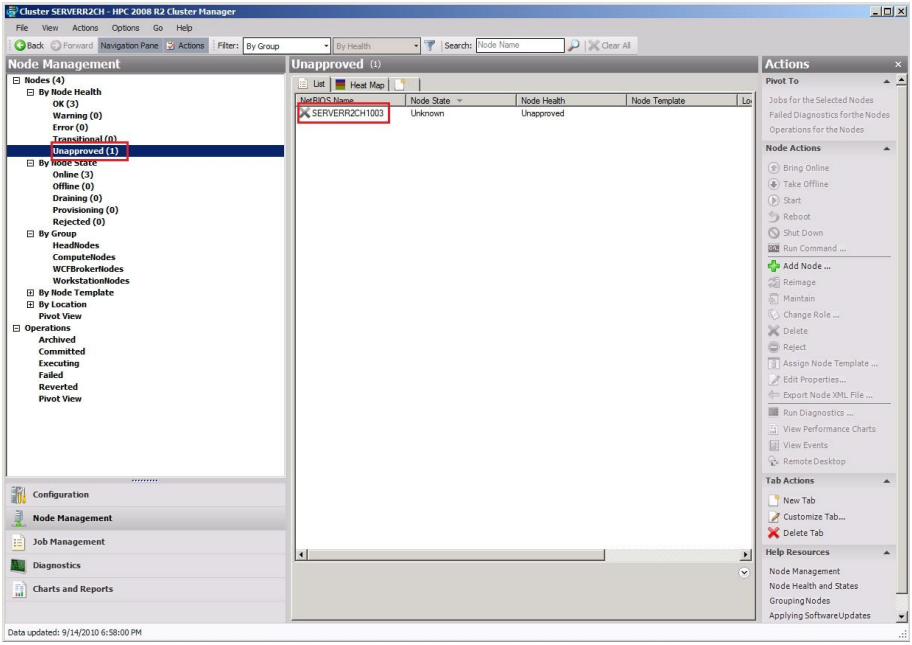



*Figure 6-16 Approve the deployment requests*

4. Select the unapproved node, right-click it, and then click **Assign Node Template**, as shown in Figure 6-17.

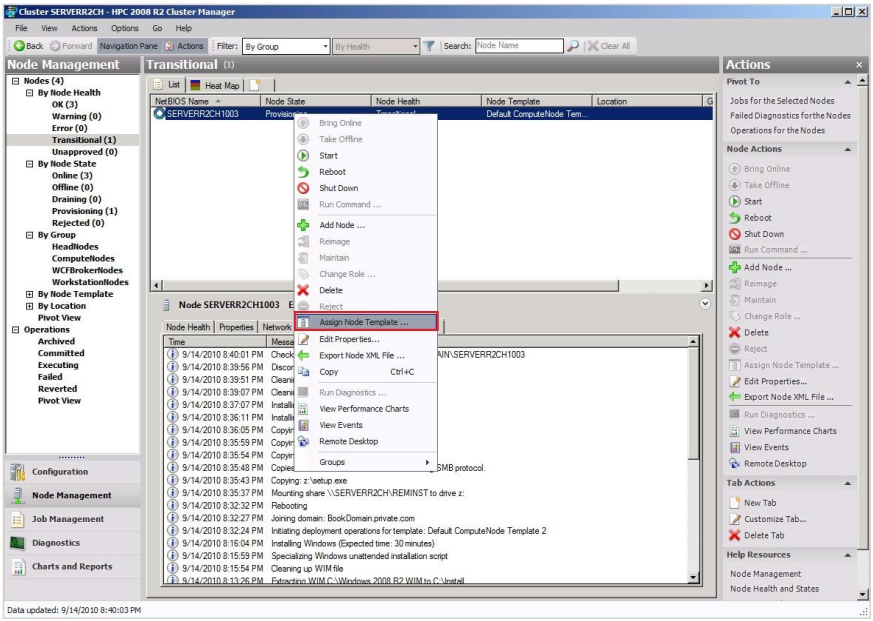



*Figure 6-17 Assign the node template*

5. In the **Assign Node Template** dialog box, select the node template created previously with the operating system image, as shown in Figure 6-18. Then, click **OK**.

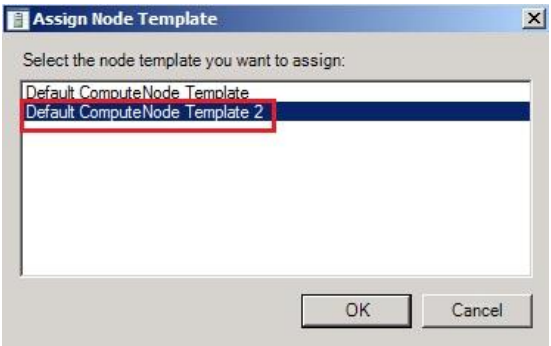



*Figure 6-18 Choose the node template*

6. In the Admin Console, the node state is changed from **Unknown** to **Provisioning**, as shown in Figure 6-19.

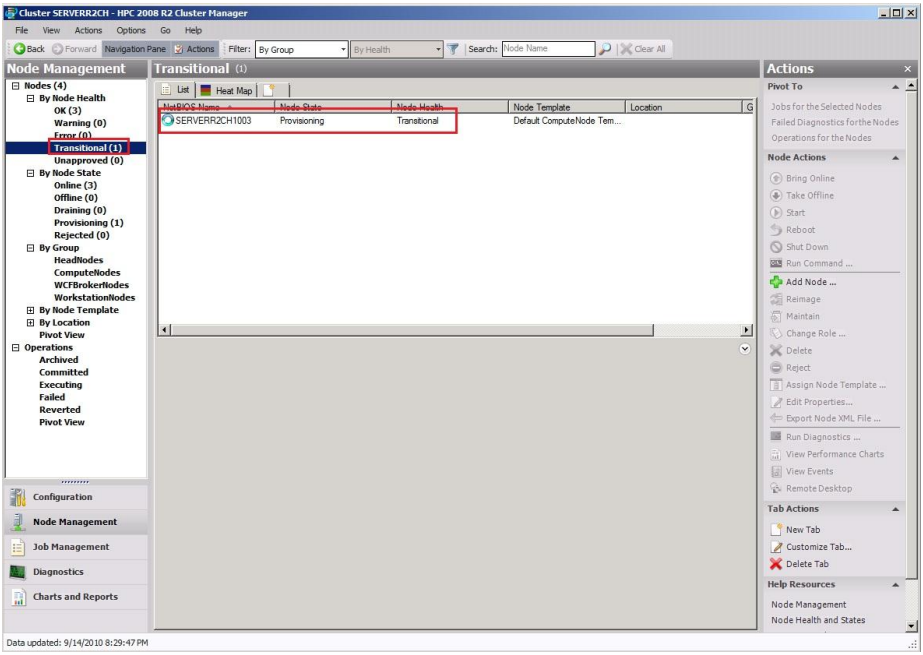



*Figure 6-19 The node is in the provisioning state*

7. Select the node. Under the node list view appears the multiple-tab node property pane. Click the **Provisioning Log** tab. The detailed deployment logs appear in the list as shown in Figure 6-20.

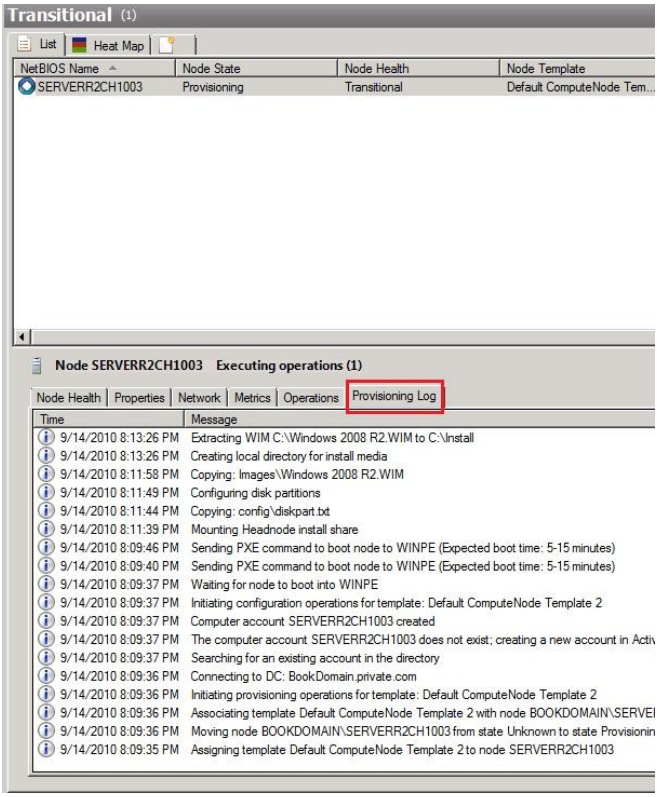



*Figure 6-20 Detailed deployment logs*

8. Go to the compute node. Windows starts to be installed on it, as shown in Figure 6-21.

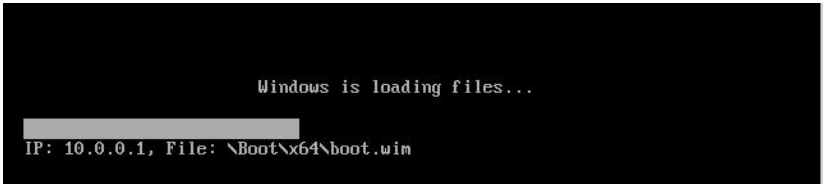



*Figure 6-21 The compute node starts installing the operating system*

9. After you enter the **Install Windows** interface as shown in Figure 6-22, the node requires around 30 minutes to complete the installation.

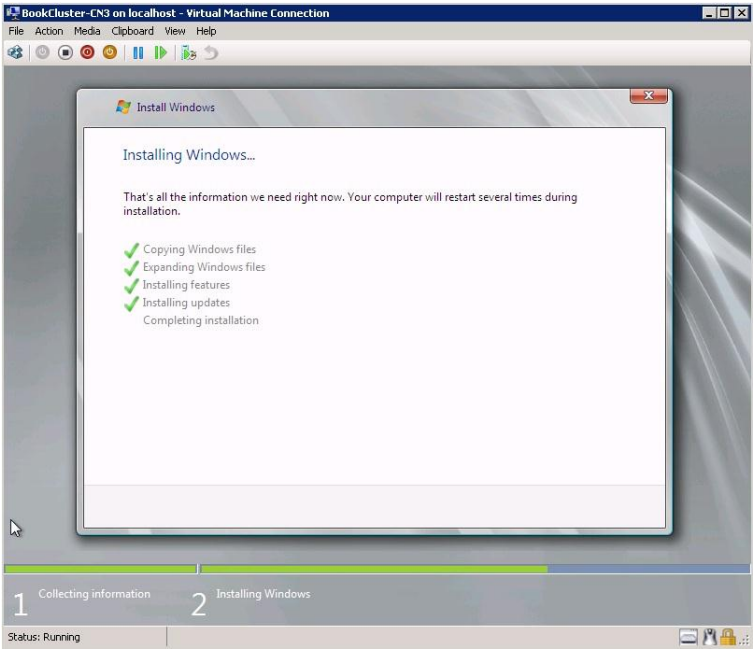



*Figure 6-22 The installation progress on the compute nodes*

10. After the deployment is complete, the compute node in the node list in the Admin Console goes into the **Offline** state, as shown in Figure 6-23.

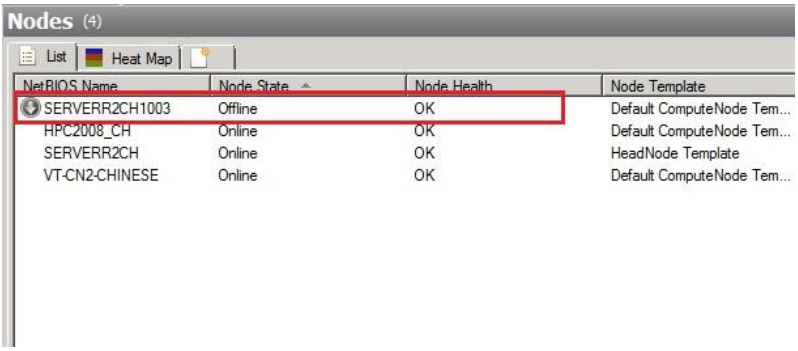



*Figure 6-23 The compute node is in the **Offline** state after the deployment is complete*

## 6.3 Configuration

Windows HPC Server 2008 R2 provides configuration functionality. By using this, administrator can configure the cluster network, node template, operating system image, job template, cluster users, and Internet small computer system interface (iSCSI) easily. Then, the jobs can get necessary resources. This section describes how to configure the job's scheduler strategy and job template. By using several common scenarios, this section introduces the functionality of the job scheduler.

## 6.3.1 Configure Immediate Pre-emption Strategy

In chapter 2, we mentioned that in using a pre-emption strategy, the higher-priority jobs can occupy necessary resources soon. Here, we submit a job with normal priority, and then submit a higher-priority job. In this way, we will show how to configure and demonstrate a pre-emption strategy.

1. Open the command prompt, and then type **job submit /numnodes:1-8 ping –t** <*headnode*> (replace <*headnode*> with the name of your head node).
2. Type **job submit /priority:Highest ping –t** <*headnode*> (replace <*headnode*> with the name of your head node).

   Now, the higher-priority job is in a queued state, as shown in Figure 6-24.

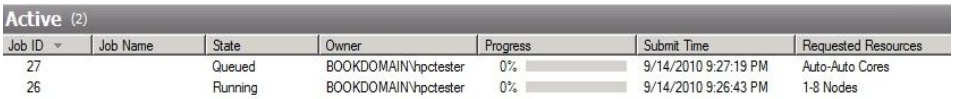



*Figure 6-24 Job list*

3. Switch to the **Configuration** navigation page, and then click **Configure job scheduler policies and settings**, as shown in Figure 6-25.

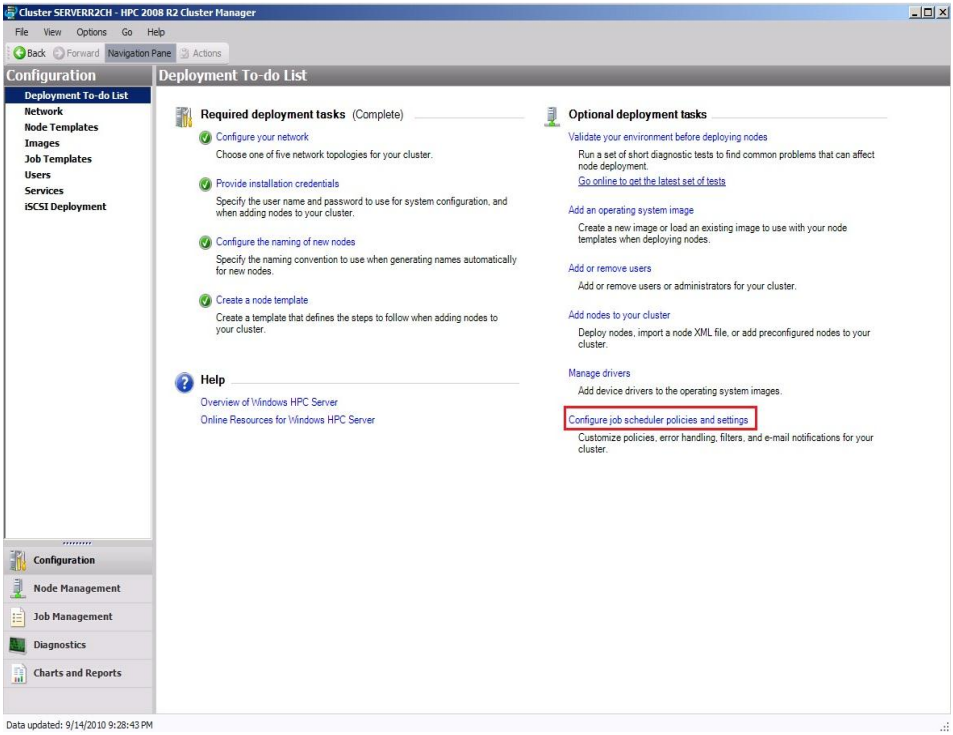



*Figure 6-25 Configure job scheduler policies and settings*

4. In the **Job Scheduler Configuration** dialog box, on the **Policy Configuration** tab, click **Immediate pre-emption** under **Job pre-emption options**, as shown in Figure 6-26.

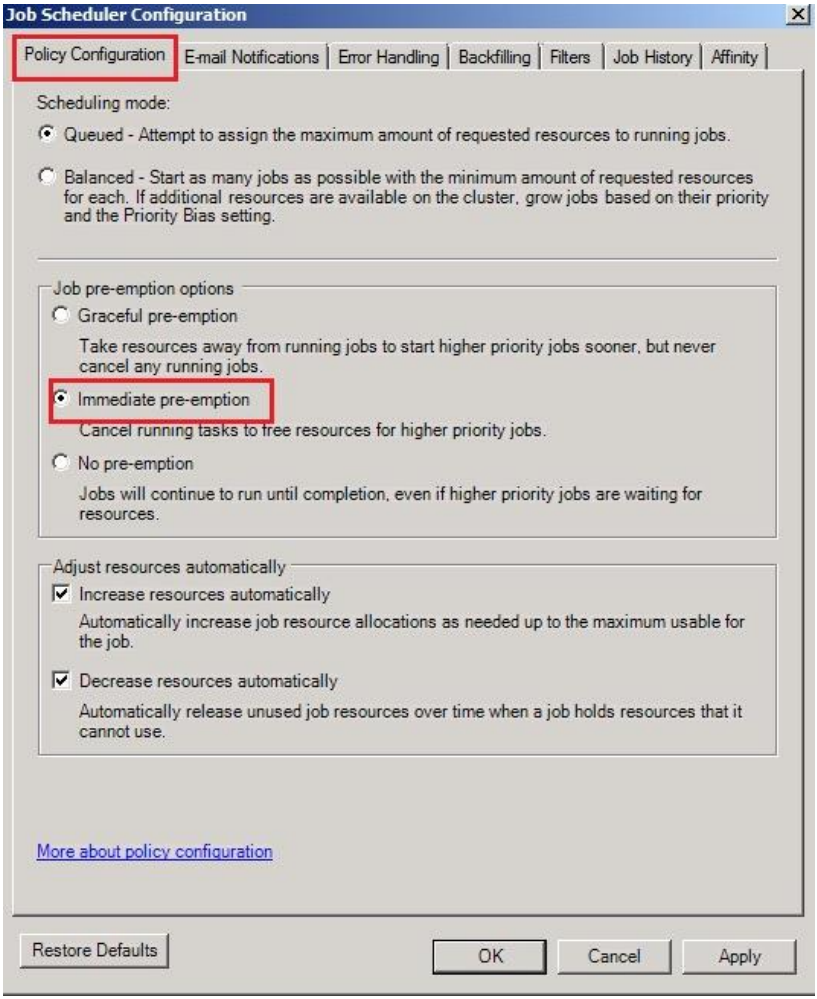
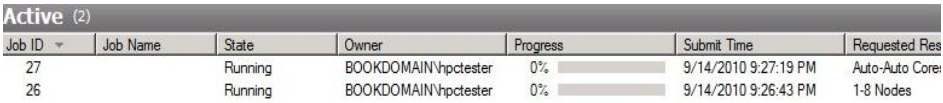



*Figure 6-26 Configure immediate pre-emption strategy*

5. The two jobs are in the **Running** state now, as shown in Figure 6-27.



Active (2)

| Job ID | Job Name | State | Owner | Progress | Submit Time | Requested Res |
|---|---|---|---|---|---|---|
| 27 | | Running | BOOKDOMAIN\hpctester | 0% | 9/14/2010 9:27:19 PM | Auto-Auto Core |
| 26 | | Running | BOOKDOMAIN\hpctester | 0% | 9/14/2010 9:26:43 PM | 1-8 Nodes |

*Figure 6-27 Higher-priority job starts to run*

6. Click the job with normal priority (your job ID may be different), and then, in the job list, click the **Activity Log** tab, as shown in Figure 6-28.

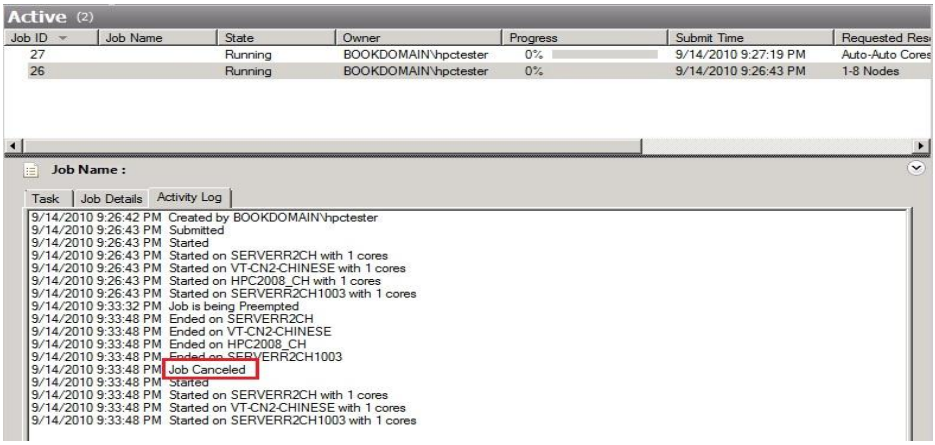



*Figure 6-28 Activity log of the normal-priority job*

Here, you can see that the job was canceled at 9:33:48 P.M. Meanwhile, the job is started. The cancel time is right after the configuration of immediate pre-emption strategy.

7. Select the higher-priority job. Its activity log appears as shown in Figure 6-29.

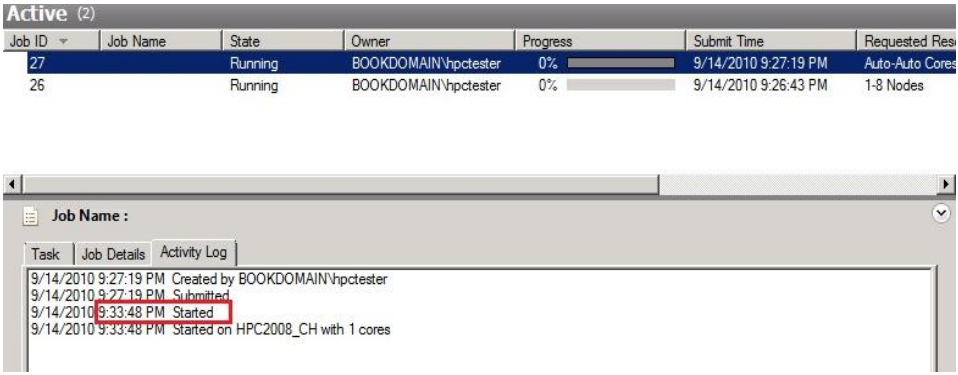



*Figure 6-29 Activity log of the higher-priority job*

Here, you can see that this job started at 9:33:48 P.M. This is the time when the normal-priority job was canceled to release node HPC2008_CH.

## 6.3.2 Configure Job Template

### 6.3.2.1 Create a Simple Job Template

Scenario: An automobile component provider sets up a cluster to help product design. The administrator of this cluster needs to create a job template for one component. By using this job template, he can help the user to submit a job and manage resource policies. Most of the jobs in this component are small ones, so the administrator needs to limit the run time to five minutes. And only the computer-aided engineering (CAE) engineer named engineer1 can use this template. Besides, the resource usage fee should be logged on the "Design" project.

1. In the **Configuration** navigation pane, click **Job Templates**, as shown in Figure 6-30.

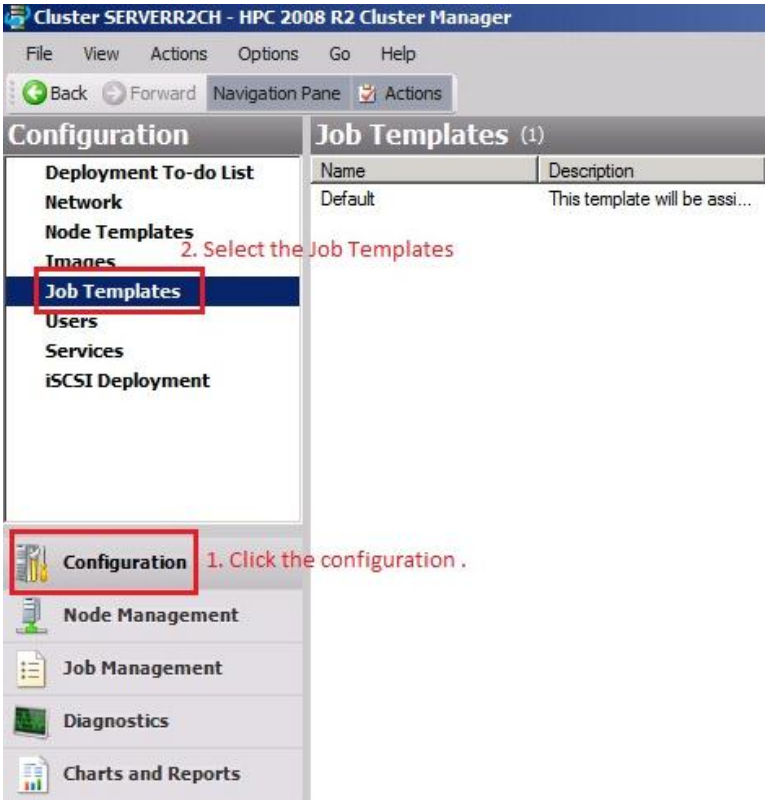



*Figure 6-30 Job templates*

You can find the default job template. If there is no job template selected while you submit jobs, the job scheduler will use the default job template to accept and verify the job's resource request.

2. In the **Action** panel, click **New**.

3. In the Generate Job Template Wizard, on the **Enter Template Name** page, type **Part design** as shown in Figure 6-31, and then click **Next**.

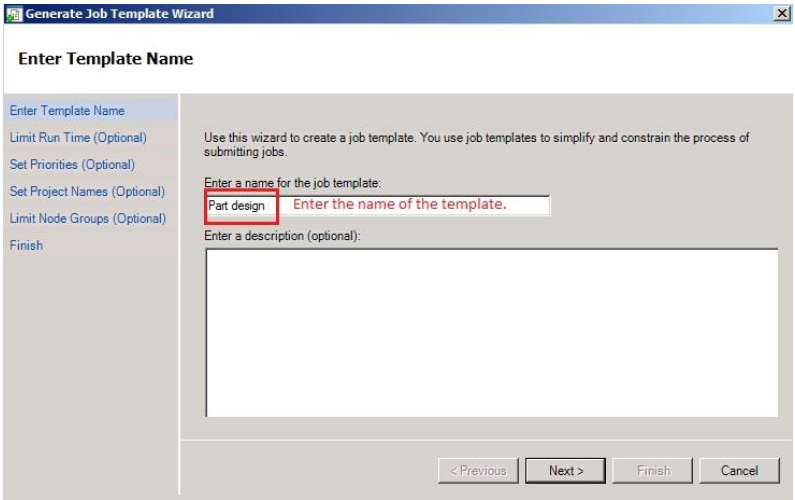



*Figure 6-31 Enter Template Name page*

4. On the **Limit Run Time** page, select the **Run jobs no longer than** check box, and then enter **5** as the number of minutes of the job run-time limit, as shown in Figure 6-32. Click **Next**.

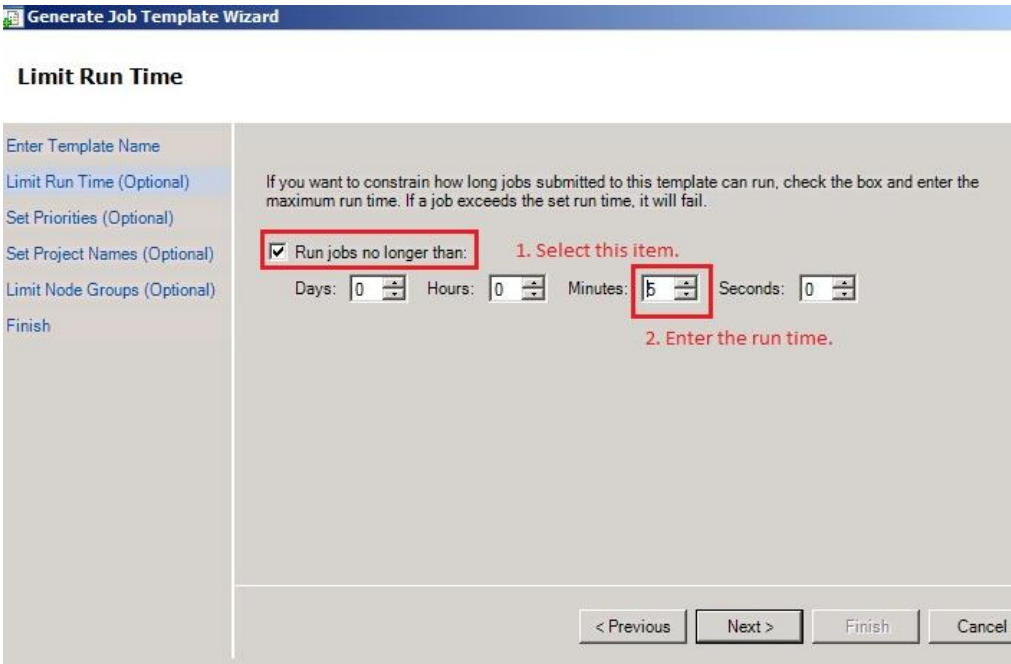



*Figure 6-32* ***Limit Run Time*** *page*

5. On the **Set Priorities** page, the default priority is **Normal**, as shown in Figure 6-33. Click **Next**.

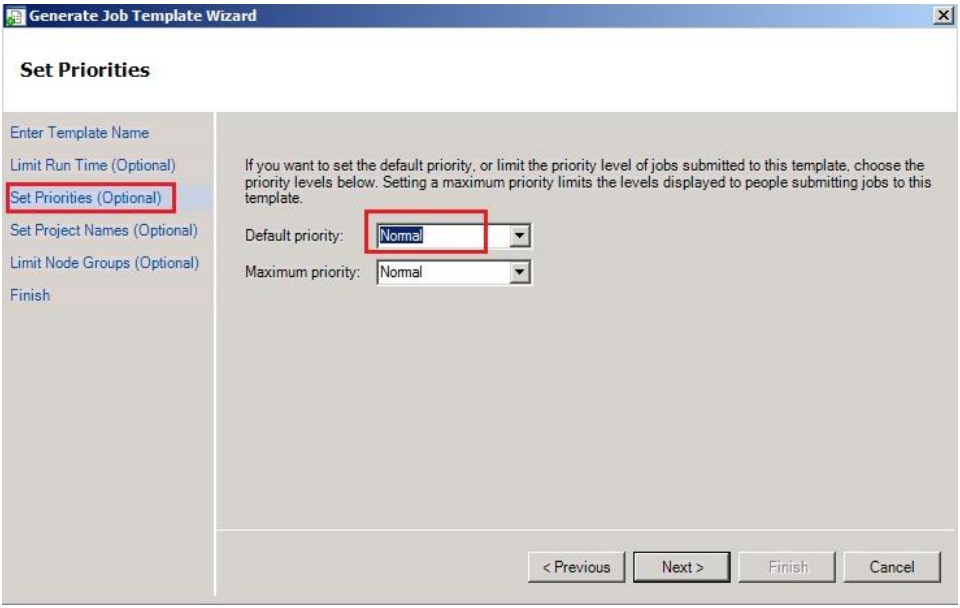



*Figure 6-33* ***Set Priorities*** *page*

6. On the **Set Project Names** page, click **Allow only the listed project names**, type **Design** as the project name, and then click **Add**, as shown in Figure 6-34. Click **Next**.

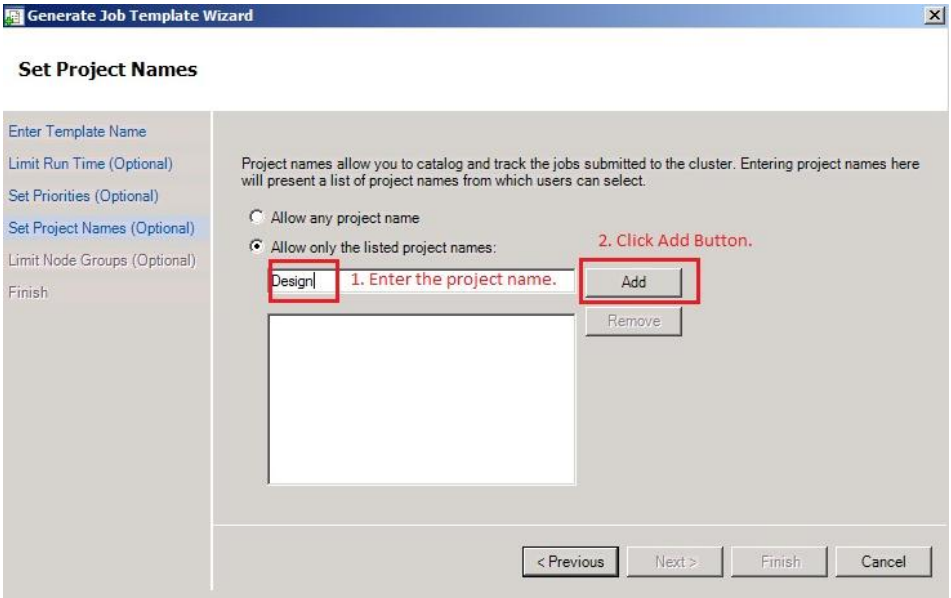



*Figure 6-34* ***Set Project Names*** *page*

7. On the **Limit Node Groups** page, don't make any limit, as shown in Figure 6-35. Click **Next**.

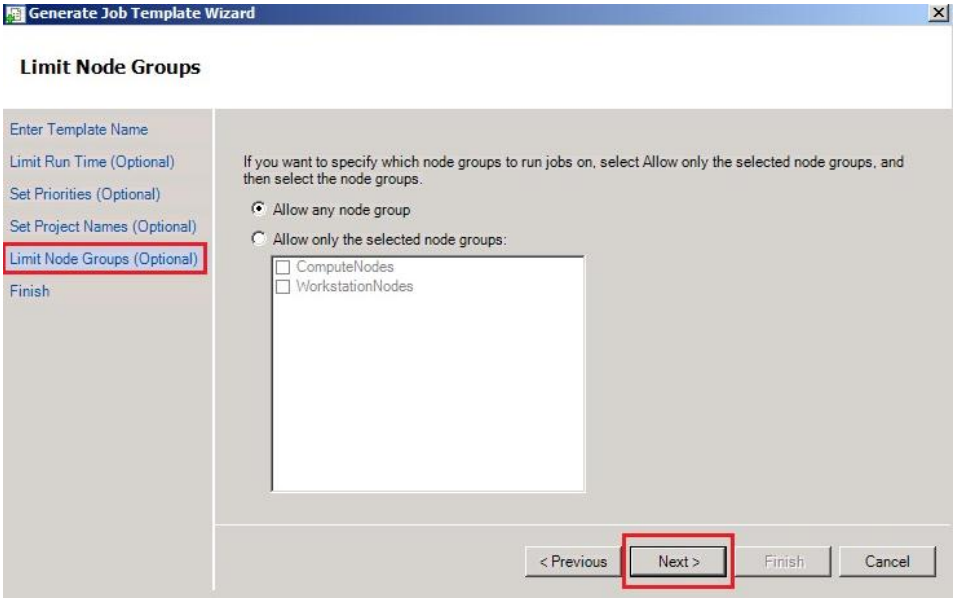



*Figure 6-35* ***Limit Node Groups*** *page*

8. On the **Finish** page, select the **Open this job template in the job template editor** check box, as shown in Figure 6-36.

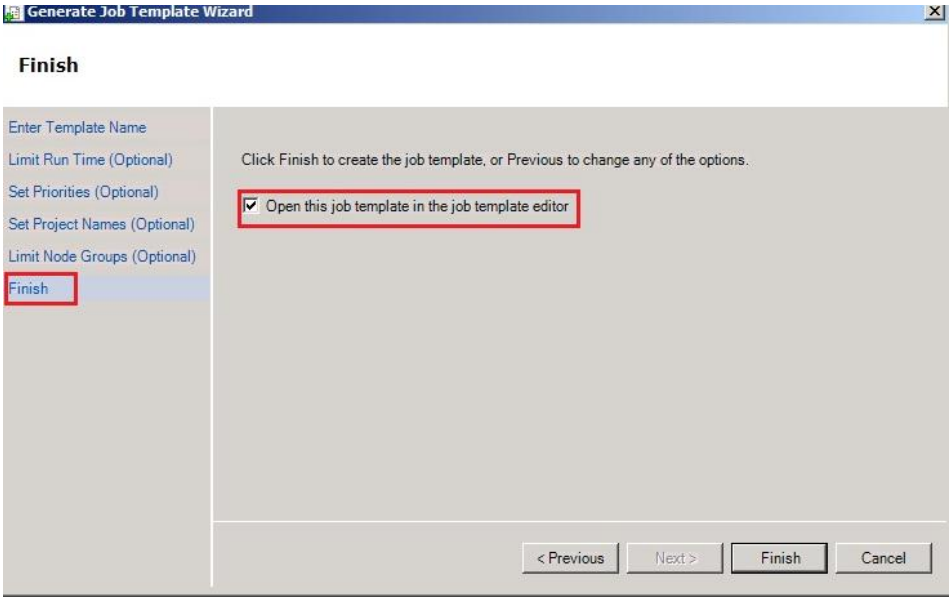



*Figure 6-36* ***Finish*** *page*

9. Click **Finish**. The **Job Template Editor** dialog box appears, as shown in Figure 6-37.

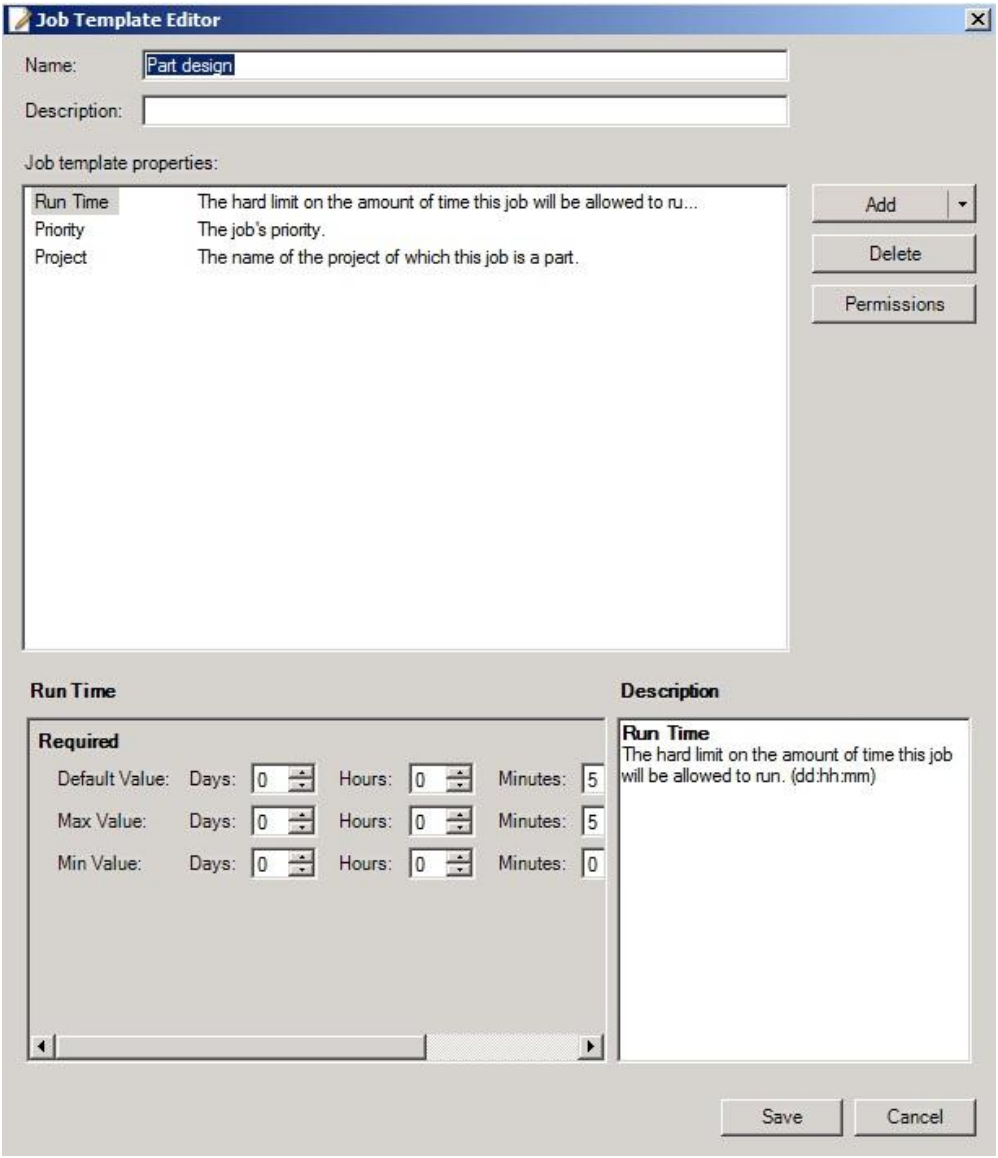



*Figure 6-37* ***Job Template Editor*** *dialog box*

10. Here, you need to set the access permissions to this job template. Click **Permissions**, as shown in Figure 6-38.

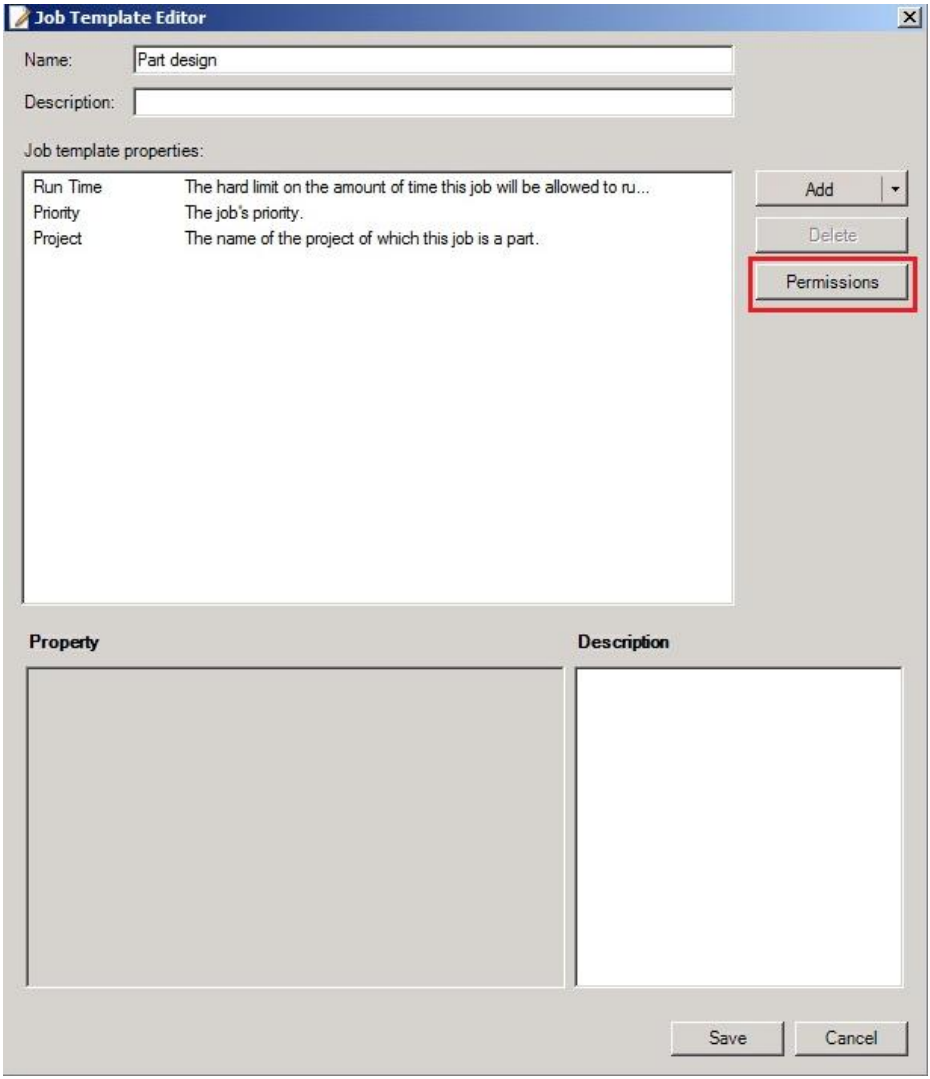



*Figure 6-38* ***Permissions*** *button*

The **Permissions for Part design** dialog box appears, as shown in Figure 6-39.

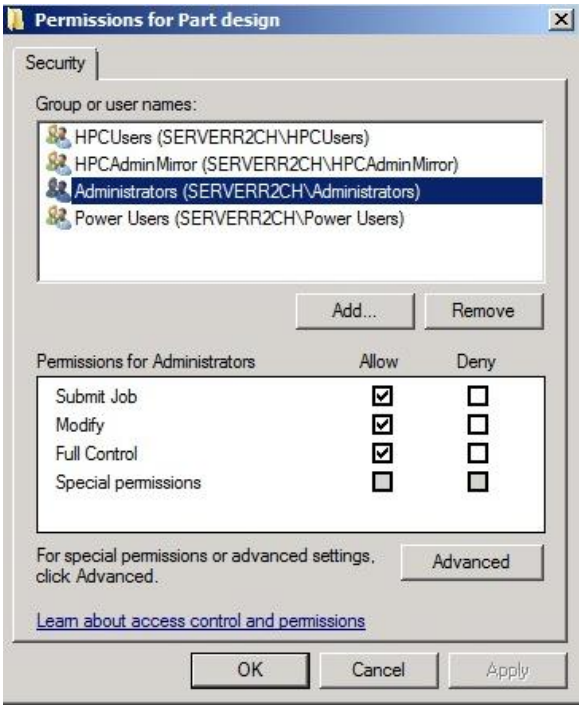



*Figure 6-39 Set permission*

Table 6-1 lists template permissions.

*Table 6-1 Template Permission Table*

| **Template permission** | **Comment** |
|---|---|
| **Submit job** | User can submit a job with this template. |
| **Modify** | User can modify this template. |
| **Full control** | User can submit jobs and can modify and delete this template. |

11. Delete **HPCUsers (**<*headnode*>**\HPCUsers)** (replace <*headnode*> with the name of your head node).
12. Click **Add**.

13. In the **Select Users, Computers, Service Accounts, or Groups** dialog box, enter the authenticated user under **Enter the object names to select**. In this cluster, set the permission to BookDomain\hpctester, as shown in Figure 6-40. Click **OK**.

*Figure 6-40 Enter user group*

14. In the **Permissions for Part deign** dialog box, you can see that BookDomain\hpctester can access to this job template, as shown in Figure 6-41. Click **OK**.

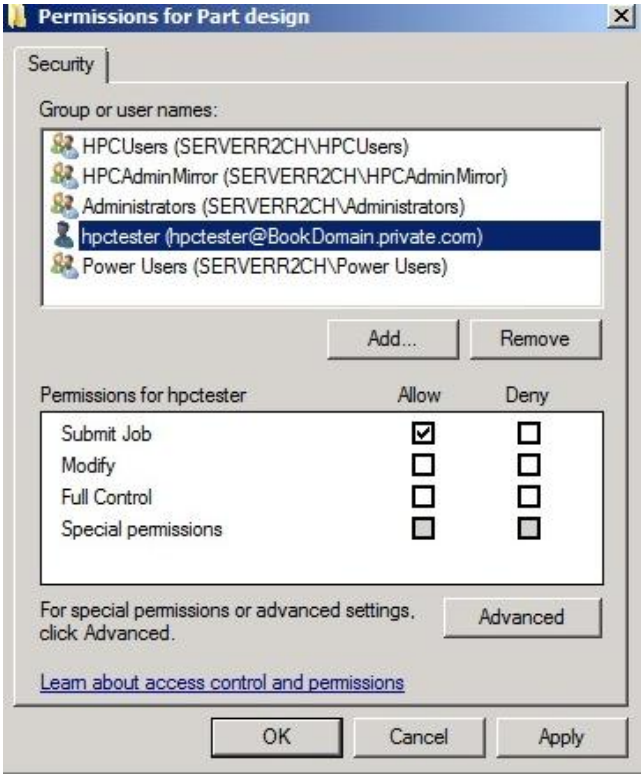



*Figure 6-41 Set permission*

15. In **Job Template Editor**, click **Save**, as shown in Figure 6-42.

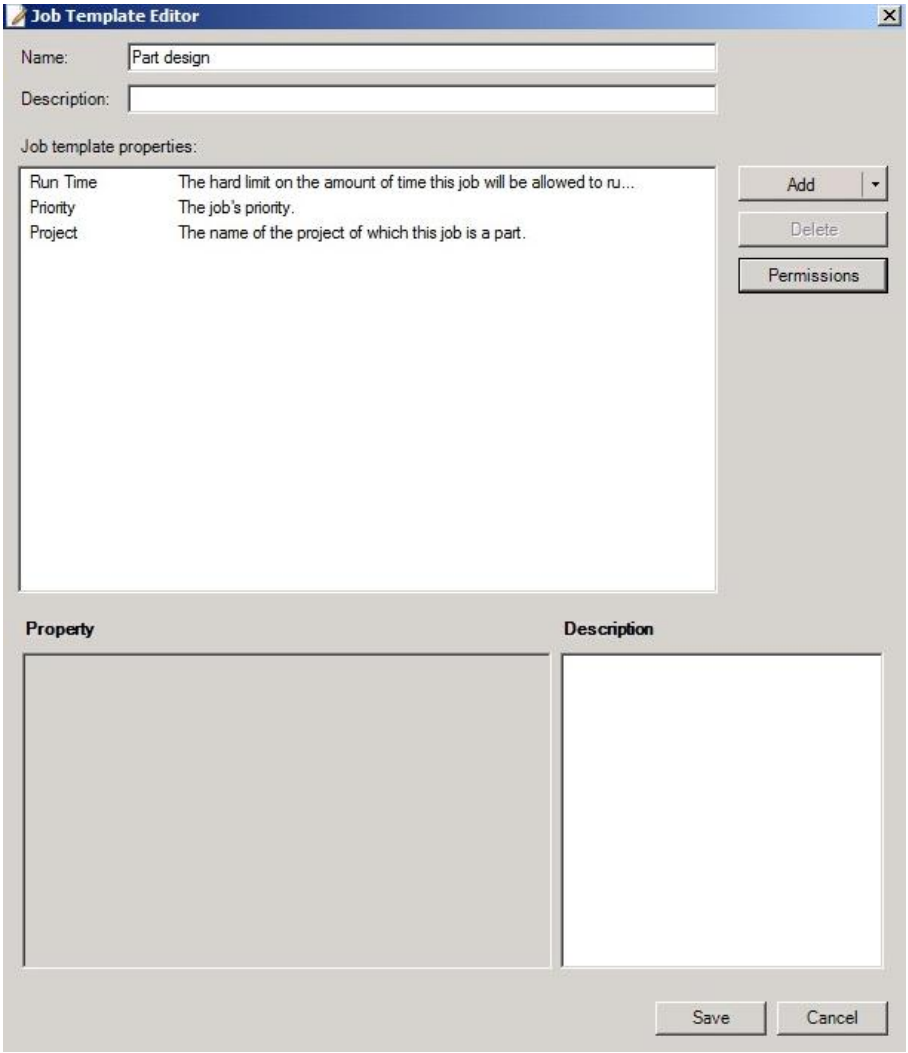



*Figure 6-42 Part design template*

16. Open a command prompt, and then type **job submit /jobtemplate:"Part Design" ping –t** <*headnode*> (replace <*headnode*>with the name of your head node). In the job scheduler list, this job is in the **Running** state, as shown in Figure 6-43.

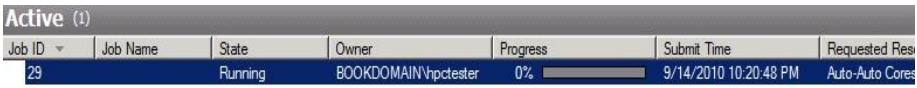



*Figure 6-43 Submit job by using the Part design template*

17. Five minutes later, job 29 is changed to the **Canceled** state, as shown in Figure 6-44. Select this job, and then click the **Job Details** tab. The error message is "Canceled by the scheduler because the job's run time expired."

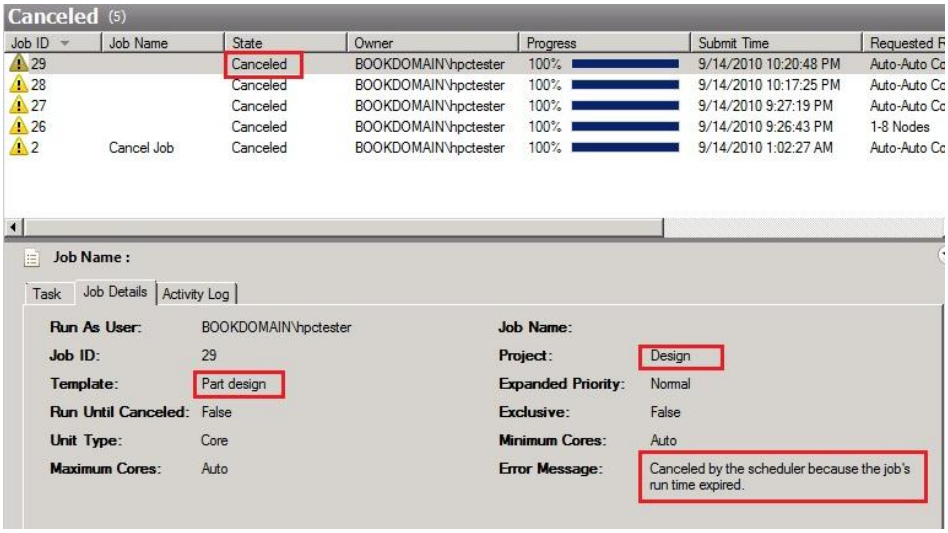



*Figure 6-44 The job is canceled because of the run-time limit of the job template*

18. Click the **Activity Log** tab, as shown in Figure 6-45. The job started at 10:20:49 P.M. and finished at 10:26:04 P.M. The total time was five minutes and 15 seconds.

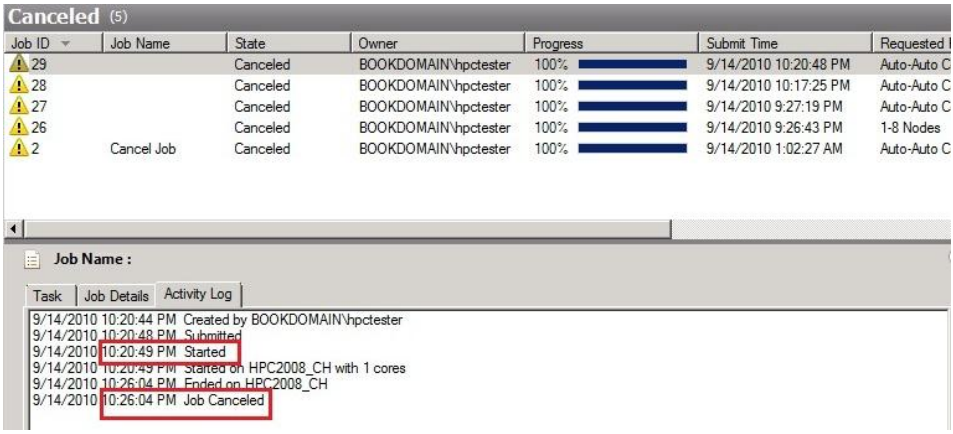



*Figure 6-45 Job activity log*

> **Why is the job not finished at five minutes? Why is there another 15 seconds?**
>
> The job scheduler has an interval to scan jobs. If the job is not expired in this interval, the scheduler will wait for another interval.

### 6.3.2.2 Resource Sharing

Scenario: A petroleum corporation has a enterprise cluster. This cluster provides computing resources for two departments: department of exploration and department of excavation. Each department has its own dedicated resources, and the two departments share some resources with each other. The shared resources will be scheduled by a first-come, first-served (FCFS) strategy. The administrator needs to set up two-node groups and job templates by completing the following steps.

## 1. Set Up Two-Node Groups

1. In Windows HPC Server 2008 R2 Cluster Manager, click **Node Management** in the left navigation. Select two nodes, right-click them, point to **Groups**, and then click **New Group**, as shown in Figure 6-46.

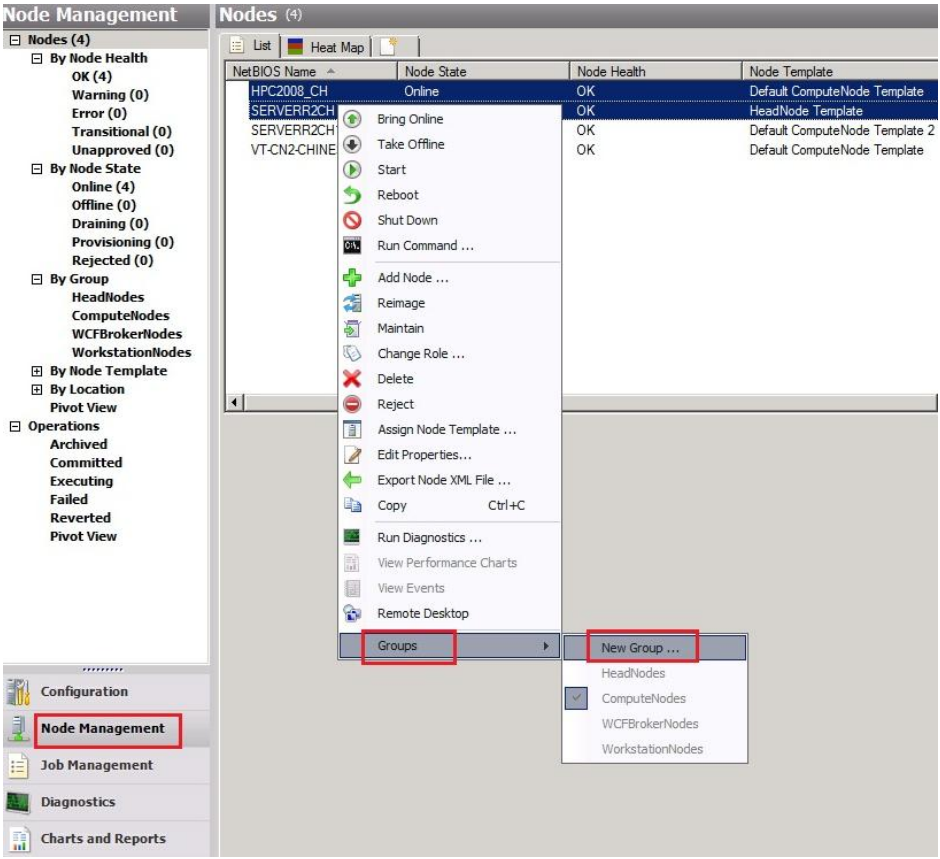



*Figure 6-46 Create new node group*

2. In the **Add Group** dialog box, type **Seismic Processing** as the name, as shown in Figure 6-47.

*Figure 6-47 Set group name*

3. Click **OK**. Now, **Seismic Processing** is in the group list of the selected two nodes, as shown in Figure 6-48.

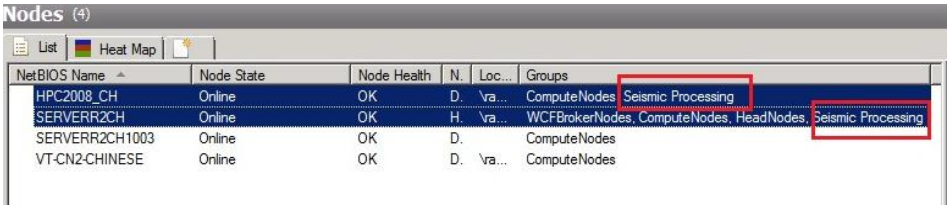



*Figure 6-48 Node group created*

4. Select nodes **HPC2008_CH** and **SERVERR2CH1003**, create a new group, and set the group name as **Reservoir Simulation**, as shown in Figure 6-49.

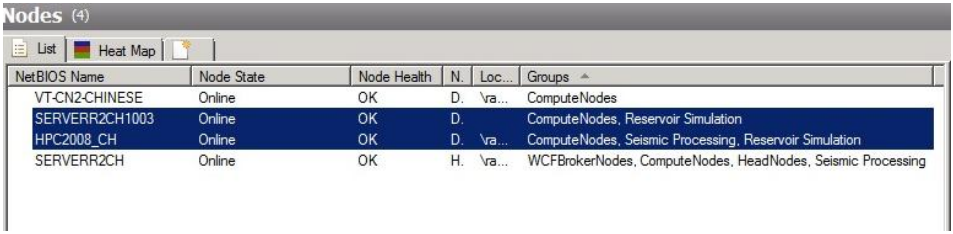



*Figure 6-49 Create Reservoir Simulation node group*

Note that two groups share the node HPC2008_CH: Seismic Processing and Reservoir Simulation.

## 2. Create Two Job Templates

1. In Cluster Manager, click **Configuration** in the left navigation, and then click **Job Templates**, as shown in Figure 6-50.

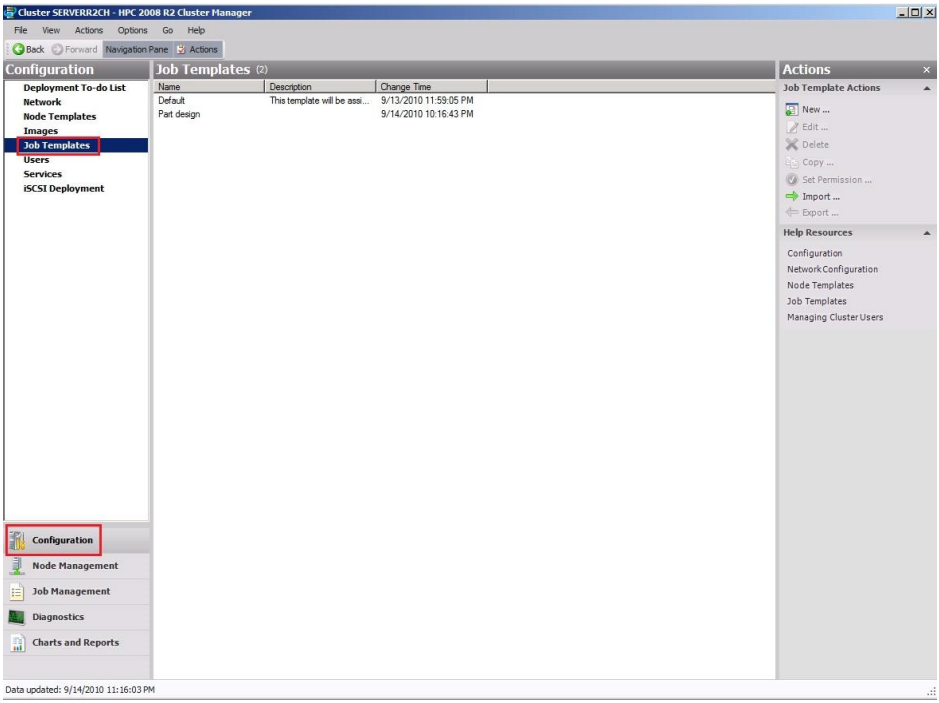



*Figure 6-50 Job template list*

2. In the **Action** panel, click **New**.

3. In the **Generate Job Template Wizard** dialog box, type **Seismic Processing** as the job template name, as shown in Figure 6-51.

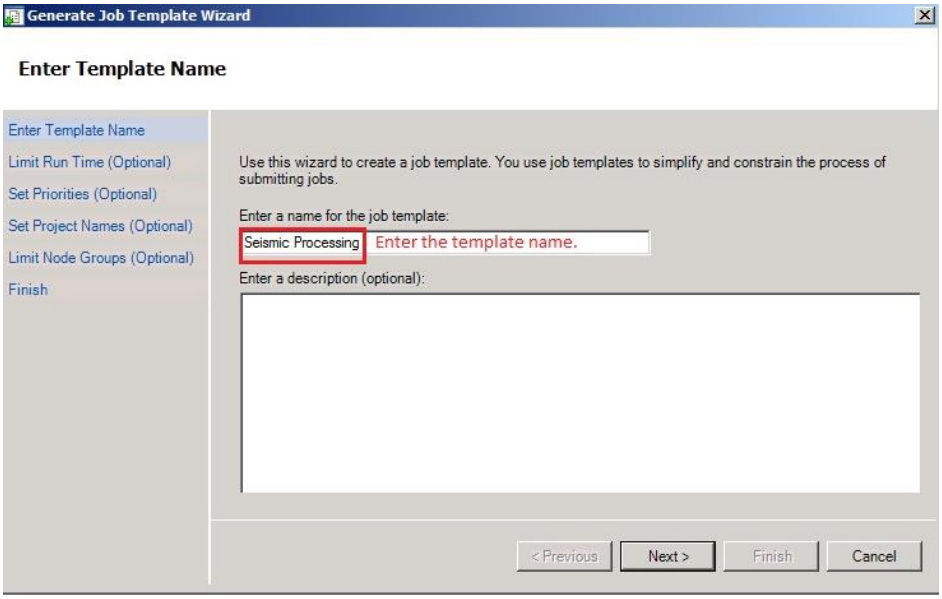



*Figure 6-51 Type **Seismic Processing** as job template name*

4. In the list of wizard pages on the left, click **Limit Node Groups**, as shown in Figure 6-52, to open the **Limit Node Groups** page.

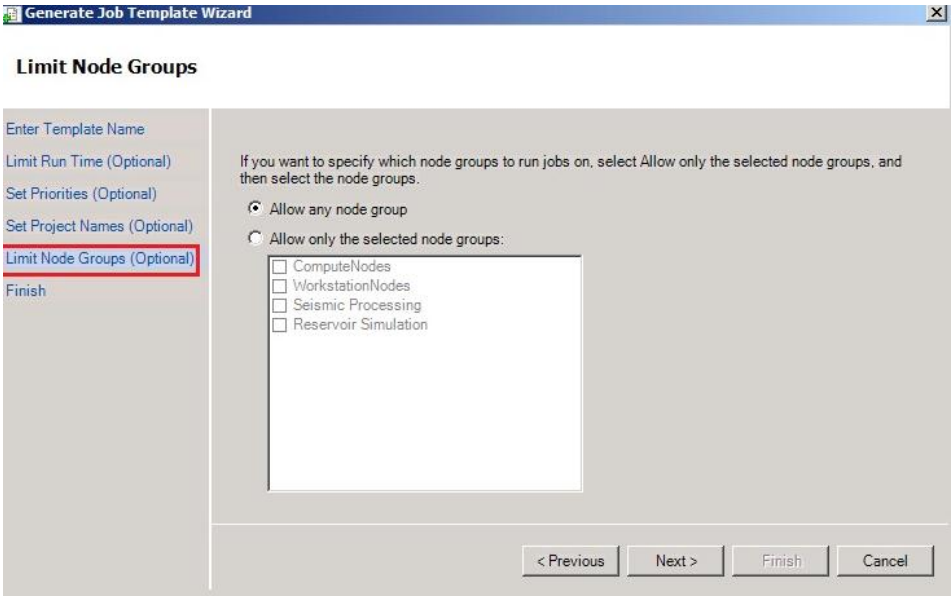



*Figure 6-52 **Limit Node Groups** page*

5. Click **Allow only the selected node groups**, and then select the **Seismic Processing** check box in the node group list, as shown in Figure 6-53.

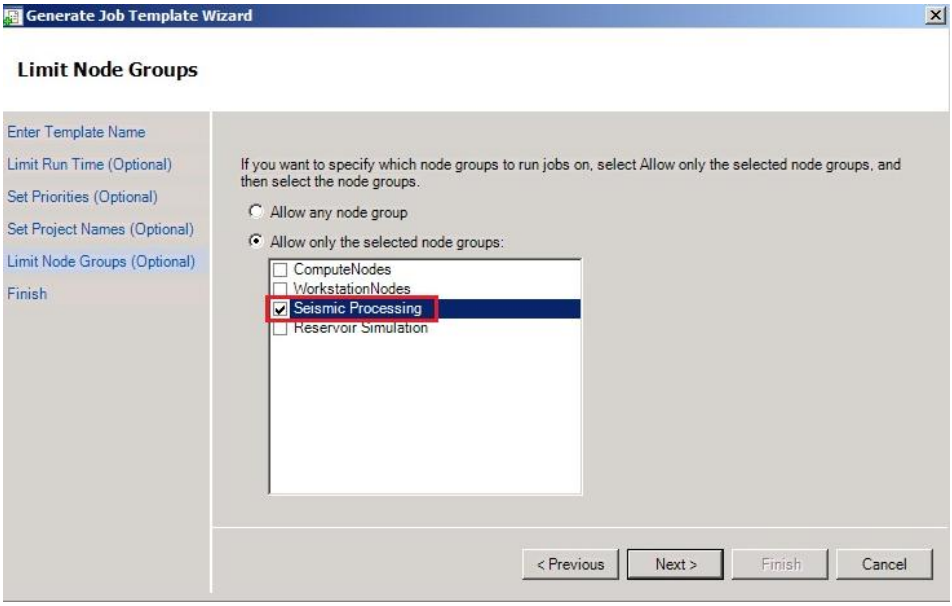



*Figure 6-53 Allow the Seismic Processing node group only*

6. In list of wizard pages on the left, click **Finish** to open the **Finish** page. Select **Open this job template in the job template editor**, and then click **Finish**.
7. In the **Job Template Editor** dialog box, click **Permissions**. Follow the same steps in section 6.3.2.1; add permission to the lob\seismic user group.
8. Repeat steps 1 through 7 to create a "Reservoir Simulation" job template. Finally, the job template list should include the job templates shown in Figure 6-54.

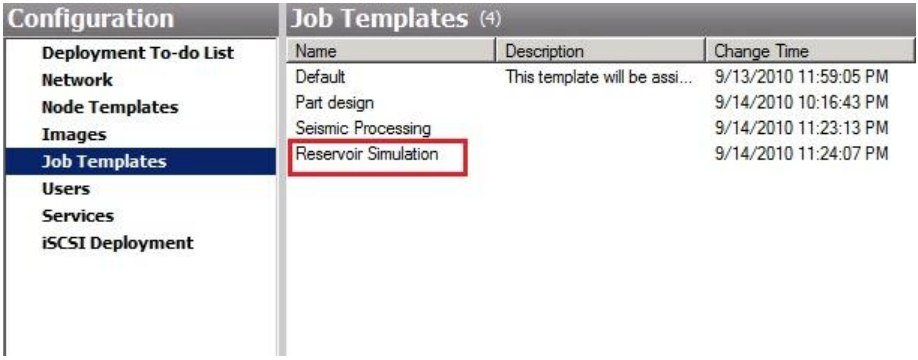



*Figure 6-54 The job template list after operation*

### 6.3.2.3 Provide Higher Priority to Access a Shared Resource to an Urgent Project

Scenario: The department of exploration of the petroleum corporation reaches the key ending phase of one project. This department needs more resources from the IT department. However, purchasing new resources takes time, and it is impossible to add new resources to the department of exploration. So, the IT department decides to leave the node group unchanged, but give the higher priority to the jobs from the department of exploration. In this way, the jobs from the department of exploration can occupy shared node HPC2008_CH with higher priority. The administrator completes the following steps to do the configuration:

1. In Cluster Manager, click **Configuration** in the left navigation, and then click **Job Templates**. In the list of job templates, select the **Seismic Processing** template, as shown in Figure 6-55.

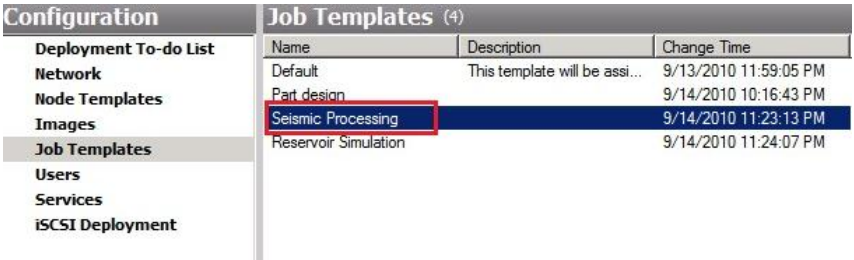



*Figure 6-55 Job template list*

2. In the **Action** panel, click **Edit**. The **Job Template Editor** dialog box appears, as shown in Figure 6-56.

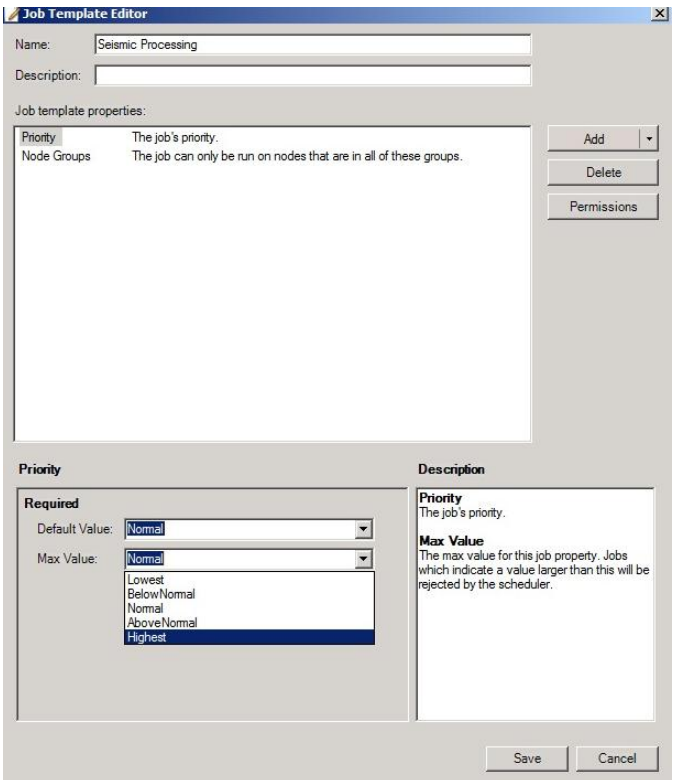



*Figure 6-56 Job templates for Seismic Processing*

3. Under **Job template properties**, click **Priority**. In the drop-down list, the administrator can define the default vale and maximum value of the job's priority. Set the default value and maximum value to **AboveNormal**, as shown in Figure 6-57.

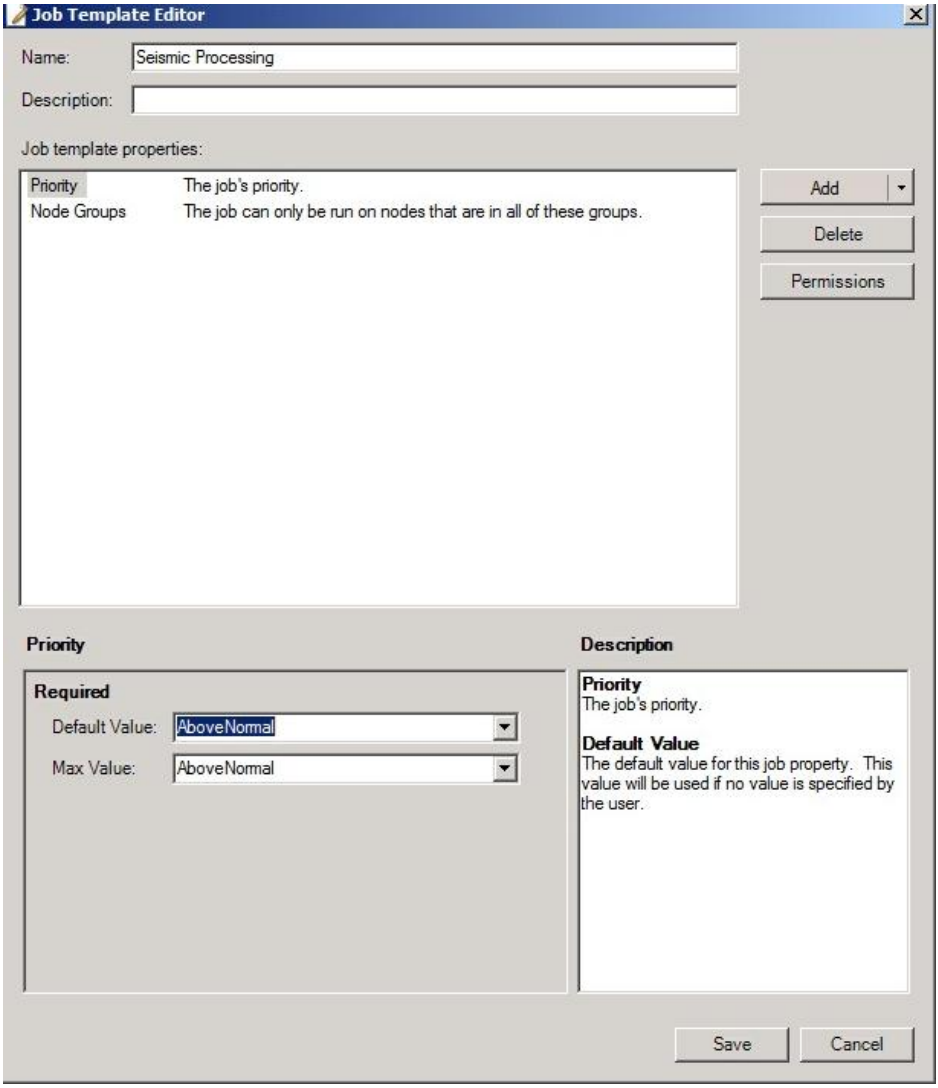



*Figure 6-57 Set the higher priority*

4. Click **Save**. While jobs are being submitted from the Seismic Processing group, the default priority is **AboveNormal**, so these jobs can access the HPC2008_CH node with higher priority.

## 6.4 Monitor the Cluster

### 6.4.1 Monitor Job Resource Usage on the Cluster Node

The computing resource is the major resource in the cluster. Administrators need to monitor the cluster status in real time. That's what the heat map provides.

1. Open the cluster Admin Console.
2. Click **Node Management**.
3. Open the appendix, go into "Chapter 6," and copy ConsumeCpu.exe to \\<*headnode*>\tools\ccpspooldir (replace <*headnode*> with the name of your head node).
4. Open the command prompt (cmd.exe), type **clusrun **<*headnode*>**\tools\ccpspooldir\ConsumeCpu.exe 100 120** (replace <*headnode*> with the name of your head node). ConsumeCpu has two parameters: One is required CPU consumption, and the second is required time (seconds) for the CPU consumption.
5. Go back to the Admin Console, click the **Heat Map** tab under **Node Management**, and you will see the heat map, as shown in Figure 6-58. The CPU usage of all the nodes is at 100 percent (in dark blue).

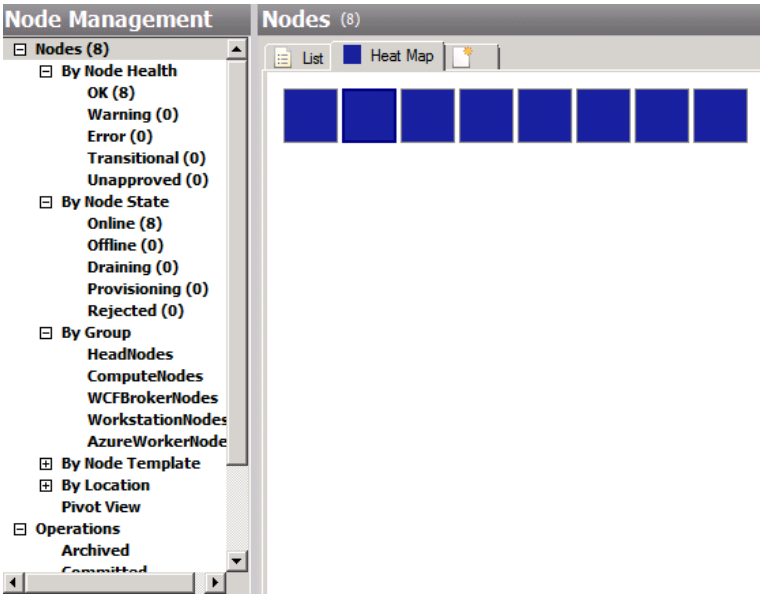



*Figure 6-58 Cluster heat map*

### 6.4.2 Monitor Memory Shortage on Nodes

Applications are not only replying on CPU, but also on memory. Some applications often have memory leaks. As a result, it is a very important task for administrators to monitor the memory usage of the cluster nodes. They expect to find the memory shortage in addition to monitoring the CPU utilizations.

1. Open the cluster Admin Console.
2. Click **Node Management**, and then switch to heat map view.

3. Click the **New Tab** tag ( ) as shown in Figure 6-59.

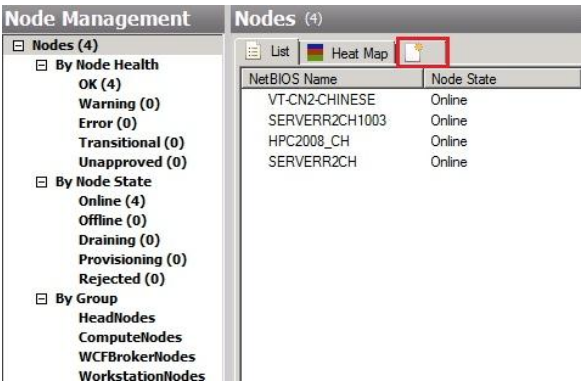



*Figure 6-59 Add a new heat map*

4. On the new tab, click **Customize Tab**, as shown in Figure 6-60.

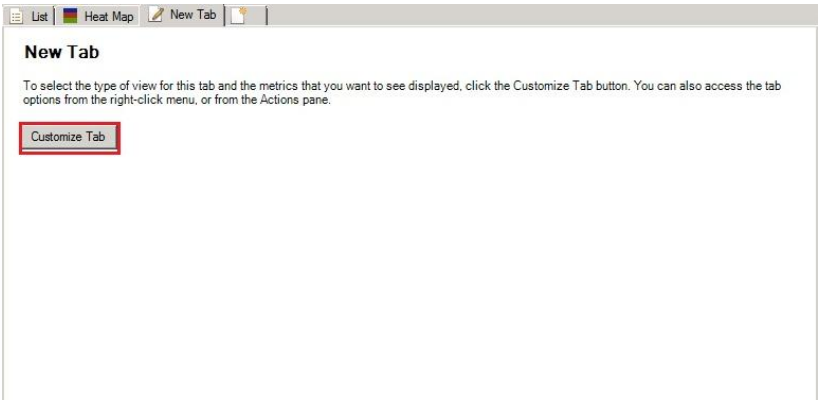



*Figure 6-60* ***Customize Tab*** *button*

5. In the **Customize Tab** dialog box, click **Heat Map View**, as shown in Figure 6-61. **Customize Tab** changes to the heat map view.

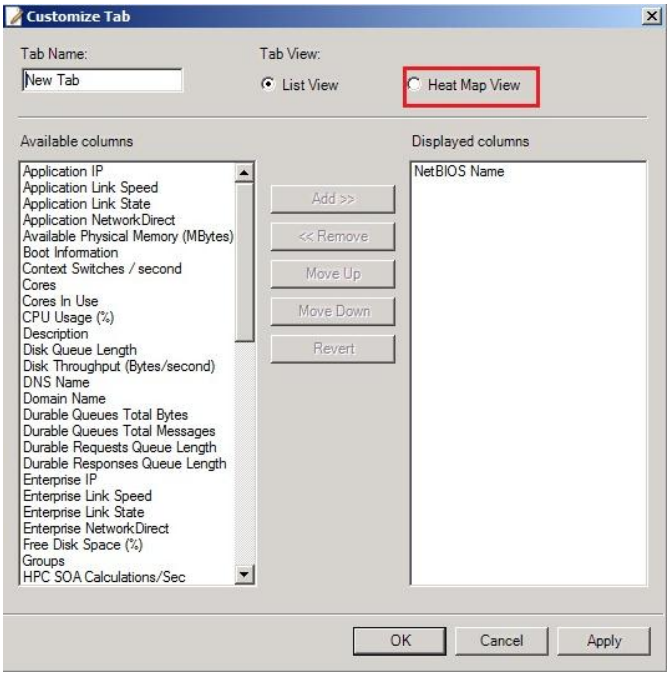



*Figure 6-61 Switching to heat map view*

6. In the **Customize Tab** dialog box, in heat map view, click **Overlaying** to view multiple metrics.

> **Two ways to view multiple metrics:**
>
> **Stacking:** Show multiple metrics at the same time, like a stack. If there are too many nodes, it is hard to identify the high load of some metrics in so many colors.
>
> **Overlaying:** Always show the metric with the highest load. If high load is identified on a node—for example, available memory is low—the node will be more obvious in the heat map view.

7. Select **CPU Usage (%)** and **Available Physical Memory (MBytes)** as the monitoring metrics. **CPU Usage (%)** is the primary metric, and **Available Physical Memory (MBytes)** is secondary. Click **Flip Scale** to the right of **Available Physical Memory (MBytes)**, as shown in Figure 6-62.

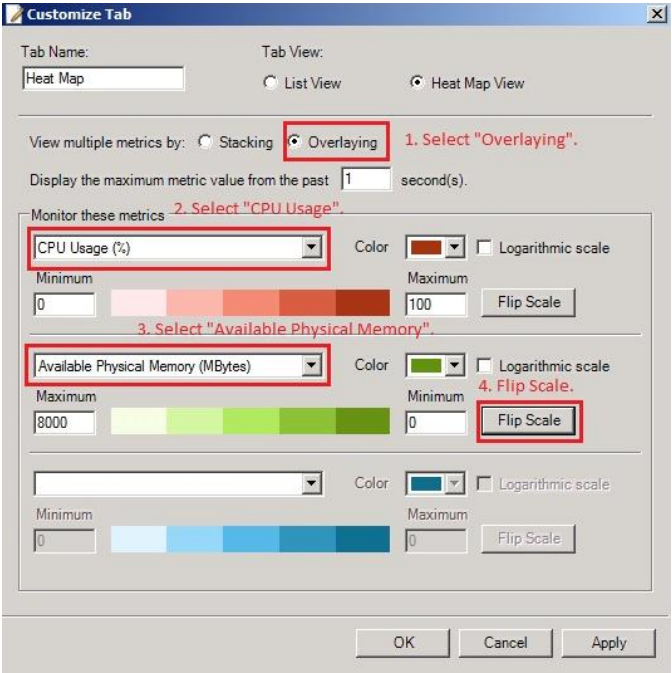



*Figure 6-62 Overlaying heat map*

> **What is flip scale?**
>
> By default, the higher the metric value, the darker the color. People naturally link the higher load with the darker color. For most metrics, like CPU utilization, it is the ideal way to present. But for some other metrics, like available physical memory, the smaller the value, the higher the load. In using the default method to display the available physical memory, the smaller value will make the color lighter, which won't get the attention of administrators. Flip scale is designed for metrics like available physical memory. Clicking `Flip Scale` changes the maximum value for the metric to 0. Then, when the available physical memory decreases, the node heat map that shows the available physical memory metric will become darker.

8. Let's make the available physical memory smaller on one node and see what happens on the stacking heat map. Use RDP to get to a compute node, open a command prompt, and run *headnode*\tools\Consume.exe memory 1000M (replace *headnode* with the name of your head node).

   The command will consume 1 gigabyte (GB) of memory on your computer.

Coming back to the heat map, the node's available physical memory metric is shown in Figure 6-63.

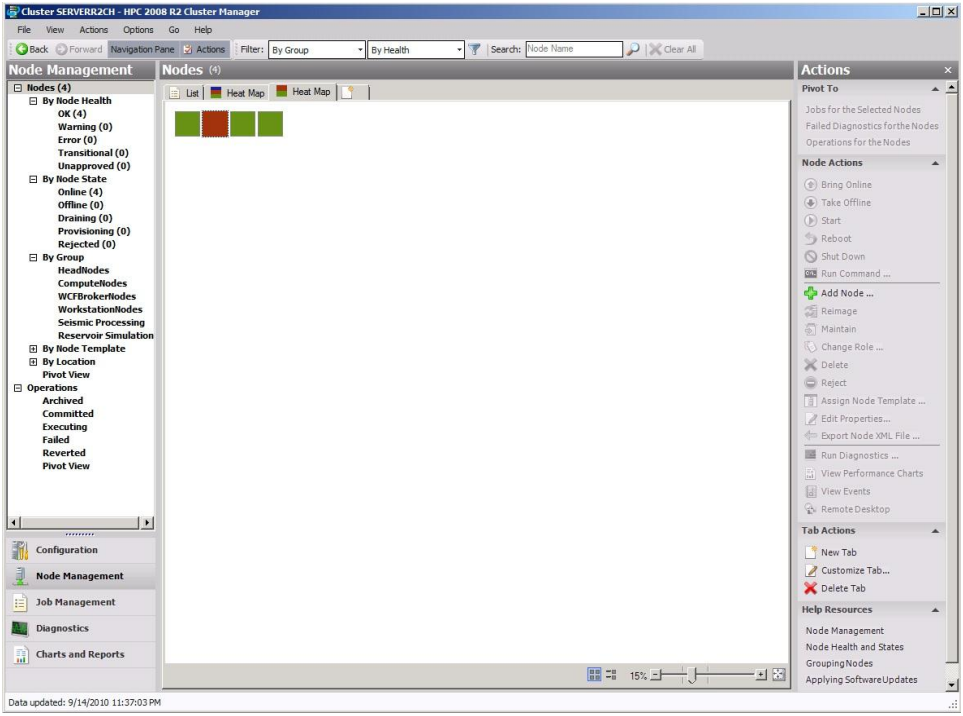



*Figure 6-63 Stacking heat map—show the node with higher memory utilization*

## 6.4.3 clusrun Commands

Cluster administrators often need to run commands on all the nodes to understand the node configurations, copy files, or do the same settings. The following operations need administrator permissions. The **clusrun** commands run concurrently on the cluster and send the standard output back to administrators' computers.

> Why not use Windows Management Instrumentation (WMI) commands?
>
> Administrations can run WMI scripts to do remote executions. But, using WMI scripts has two limitations: The command can only be executed linearly, and a security concern exists.
>
> The **clusrun** commands are run based on job scheduler. As a result, the commands' executions on multiple nodes are concurrent, and there is consistent credential cache and management.

### 6.4.3.1 View the Network Configurations on All the Nodes

1. Open the command prompt (cmd.exe).
2. Type **clusrun ipconfig /all**. The output is as shown in Figure 6-64.

```
Select Administrator: Command Prompt

   Media State . . . . . . . . . . . : Media disconnected
   Connection-specific DNS Suffix  . :
   Description . . . . . . . . . . . : Microsoft ISATAP Adapter #2
   Physical Address. . . . . . . . . : 00-00-00-00-00-00-00-E0
   DHCP Enabled. . . . . . . . . . . : No
   Autoconfiguration Enabled . . . . : Yes
------------------------- OLYMPUS-12-11 returns 0 -------------------------

Windows IP Configuration

   Host Name . . . . . . . . . . . . : OLYMPUS-12-11
   Primary Dns Suffix  . . . . . . . : hpclab.local
   Node Type . . . . . . . . . . . . : Hybrid
   IP Routing Enabled. . . . . . . . : No
   WINS Proxy Enabled. . . . . . . . : No
   DNS Suffix Search List. . . . . . : hpclab.local
                                       ntdev.corp.microsoft.com
                                       redmond.corp.microsoft.com
                                       dns.corp.microsoft.com

Ethernet adapter Private:

   Connection-specific DNS Suffix  . :
   Description . . . . . . . . . . . : Broadcom BCM5708S NetXtreme II GigE (NDIS
 VBD Client) #2
   Physical Address. . . . . . . . . : 00-1E-0B-DB-04-8A
   DHCP Enabled. . . . . . . . . . . : Yes
   Autoconfiguration Enabled . . . . : Yes
   Link-local IPv6 Address . . . . . : fe80::90ab:ef4:6996:38de%13(Preferred)
   IPv4 Address. . . . . . . . . . . : 192.168.1.216(Preferred)
   Subnet Mask . . . . . . . . . . . : 255.255.0.0
   Lease Obtained. . . . . . . . . . : Friday, April 30, 2010 4:47:11 PM
   Lease Expires . . . . . . . . . . : Tuesday, May 11, 2010 4:47:11 AM
   Default Gateway . . . . . . . . . :
   DHCP Server . . . . . . . . . . . : 192.168.0.1
   DHCPv6 IAID . . . . . . . . . . . : 301997579
   DHCPv6 Client DUID. . . . . . . . : 00-01-00-01-13-6D-21-C5-00-1E-0B-DB-04-8C

   DNS Servers . . . . . . . . . . . : fec0:0:0:ffff::1%1
                                       fec0:0:0:ffff::2%1
                                       fec0:0:0:ffff::3%1
   NetBIOS over Tcpip. . . . . . . . : Enabled

Ethernet adapter Enterprise:
```

*Figure 6-64 Output of the **clusrun** command*

### 6.4.3.2 Copy a File to All the Nodes

Copying one file to all the nodes entails copying the file to a shared folder first, and then running the **clusrun** copy command.

1. Create a foo.txt file, and type **foo** as the content.
2. Copy foo.txt to head node *headnode*\ccpspooldir (replace *headnode* with the name of your head node).
3. Open a command prompt, and then type **clusrun md c:\cluscopytest**. Figure 6-65 shows the result.

```
C:\Users\mingqxu>clusrun /scheduler:shpc-0042 md c:\cluscopytest
------------------------- SHPC-0042 returns 0 -------------------------
------------------------- SHPC-00421001 returns 0 -------------------------
------------------------- SHPC-00421000 returns 0 -------------------------

------------------------- Summary -------------------------
3 Nodes succeeded
0 Nodes failed
```

*Figure 6-65 Output of the **clusrun** make-directory command*

4. Type **clusrun /scheduler:** *<headnode>* **copy** *<headnode>*\ccpspooldir\foo.txt **c:\cluscopytest** (replace *<headnode>* with the real name of the head node). Figure 6-66 shows the result.

```
C:\Users\mingqxu>clusrun /scheduler:shpc-0042 copy \\shpc-0042\ccpspooldir\foo.t
xt c:\cluscopytest
------------------------- SHPC-0042 returns 0 -------------------------
        1 file(s) copied.
------------------------- SHPC-00421001 returns 0 -------------------------
        1 file(s) copied.
------------------------- SHPC-00421000 returns 0 -------------------------
        1 file(s) copied.

------------------------- Summary -------------------------
3 Nodes succeeded
0 Nodes failed
```

*Figure 6-66 Output of the **clusrun** copy command*

The last line shows the execution results on the nodes. Make sure that all nodes succeeded.

> **Why can't I see the output, and why am I encountering errors?**
>
> Cause: You are running the command from your workstation, and your workstation doesn't open File Sharing. Open File Sharing and try again.

5. Let's verify whether the file is copied to compute nodes. Type **clusrun dir c:\cluscopytest\foo.txt**. Figure 6-67 shows the result.

```
 Directory of c:\cluscopytest

07/01/2010  01:14 AM    <DIR>          .
07/01/2010  01:14 AM    <DIR>          ..
07/01/2010  01:12 AM                 3 foo.txt
               1 File(s)              3 bytes
               2 Dir(s)  144,315,420,672 bytes free
------------------------- SHPC-00421000 returns 0 -------------------------
 Volume in drive C is Node
 Volume Serial Number is 9C01-84A1

 Directory of c:\cluscopytest

07/01/2010  01:14 AM    <DIR>          .
07/01/2010  01:14 AM    <DIR>          ..
07/01/2010  01:12 AM                 3 foo.txt
               1 File(s)              3 bytes
               2 Dir(s)  120,250,265,600 bytes free

------------------------- Summary -------------------------
3 Nodes succeeded
0 Nodes failed
```

*Figure 6-67 Output of the **clusrun dir** command*

6. Delete the file, and then type **clusrun del c:\cluscopytest\foo.txt**. Figure 6-68 shows the result.

```
------------------------- SHPC-0042 returns 0 -------------------------
------------------------- SHPC-00421001 returns 0 -------------------------
------------------------- SHPC-00421000 returns 0 -------------------------

------------------------- Summary -------------------------
3 Nodes succeeded
0 Nodes failed
```

*Figure 6-68 Output of the **clusrun** command*

7. Delete the folder, and then type **clusrun rd c:\cluscopytest**. Figure 6-69 shows the result.

```
------------------------- SHPC-0042 returns 0 -------------------------
------------------------- SHPC-00421001 returns 0 -------------------------
------------------------- SHPC-00421000 returns 0 -------------------------

------------------------- Summary -------------------------
3 Nodes succeeded
0 Nodes failed
```

*Figure 6-69 Output of the **clusrun** command*

> **Attention**
>
> The **clusrun** commands won't be queued in the job scheduler, but run directly on the nodes. Be careful not to run long-running, high-resource-consumption (CPU, memory, I/O, and network throughput) **clusrun** commands on your production cluster to avoid interfering with your normal jobs, even causing job failure because of lack of resources.

### 6.4.3.3 Set Fusion Records

If, at run time, your application encounters the error "Cannot load DLL, please open Fusion Log," you need to get your job ID and find the nodes (HostA, HostB) that your job runs on in the Admin Console. Then, you open the Fusion Log by completing the following steps to find out in which directories the system is loading required dynamic-link libraries (DLLs).

1. Open the book's CD Chapter 6\ directory, and then copy foo.exe to one of the nodes—for example, the C:\HPCBook\ directory on HostA.
2. Log on to HostA, and then run c:\HPCBook\foo.exe. The output is shown in Figure 6-70. The error information asks us to open Fusion Log to get more information to troubleshoot.

```
Unhandled Exception: System.IO.FileNotFoundException: Could not load file or ass
embly 'bar, Version=1.0.0.0, Culture=neutral, PublicKeyToken=null' or one of its
 dependencies. The system cannot find the file specified.
File name: 'bar, Version=1.0.0.0, Culture=neutral, PublicKeyToken=null'
   at foo.Program.Main(String[] args)

WRN: Assembly binding logging is turned OFF.
To enable assembly bind failure logging, set the registry value [HKLM\Software\M
icrosoft\Fusion!EnableLog] (DWORD) to 1.
Note: There is some performance penalty associated with assembly bind failure lo
gging.
To turn this feature off, remove the registry value [HKLM\Software\Microsoft\Fus
ion!EnableLog].
```

*Figure 6-70 Command output*

3. Open a command prompt, and then type the **clusrun /nodes:hostA reg add HKLM\Software\Microsoft\Fusion /v EnableLog /t REG_DWORD /d 1** command to enable Fusion Log.

4. Run foo.exe again. The output is shown in Figure 6-71.

```
Unhandled Exception: System.IO.FileNotFoundException: Could not load file or ass
embly 'bar, Version=1.0.0.0, Culture=neutral, PublicKeyToken=null' or one of its
 dependencies. The system cannot find the file specified.
File name: 'bar, Version=1.0.0.0, Culture=neutral, PublicKeyToken=null'
   at foo.Program.Main(String[] args)

=== Pre-bind state information ===
LOG: User = REDMOND\mingqxu
LOG: DisplayName = bar, Version=1.0.0.0, Culture=neutral, PublicKeyToken=null
 (Fully-specified)
LOG: Appbase = file:///C:/HPCBook/
LOG: Initial PrivatePath = NULL
Calling assembly : foo, Version=1.0.0.0, Culture=neutral, PublicKeyToken=null.
===
LOG: This bind starts in default load context.
LOG: No application configuration file found.
LOG: Using machine configuration file from C:\Windows\Microsoft.NET\Framework\v2
.0.50727\config\machine.config.
LOG: Policy not being applied to reference at this time (private, custom, partia
l, or location-based assembly bind).
LOG: Attempting download of new URL file:///C:/HPCBook/bar.DLL.
LOG: Attempting download of new URL file:///C:/HPCBook/bar/bar.DLL.
LOG: Attempting download of new URL file:///C:/HPCBook/bar.EXE.
LOG: Attempting download of new URL file:///C:/HPCBook/bar/bar.EXE.
```

*Figure 6-71 Output with Fusion Log*

You can see that the application is trying to load bar.dll and bar.exe in the current directory and the bar\ directory.

5. Copy Chapter 6\bar.dll to the C:\HPCBook\ directory on HostA.
6. Run the application again. The result is shown in Figure 6-72.

```
Sucess to call bar.dll

C:\HPCBook>
```

*Figure 6-72 Success result*

7. Don't forget to disable Fusion Log: Type **clusrun /nodes:hostA reg delete HKLM\Software\Microsoft\Fusion /v EnableLog /f**

## 6.5 Diagnostics

Administrators must be able to quickly troubleshoot in order to help users maintain high efficiency. Windows HPC Server provides built-in diagnostic tests to help administrators. Below are the uses of the tests for some user scenarios.

### 6.5.1 Job Failed Because of a Failure in User Authentication

Scenario: Your user sent an e-mail message to report that her job failed. The error message reads "Failed to logon for the user: authentication with domain controller failed." You can use the following steps to diagnose the problem:

1. Start Cluster Manager.
2. Under **Diagnostics**, select **Deployment: Active DirectoryConnectivity Test**.
3. Double-click the test, and then click **All nodes**.
4. Click **Run**.

5. After the test finishes, open the result report, which looks like Figure 6-73.

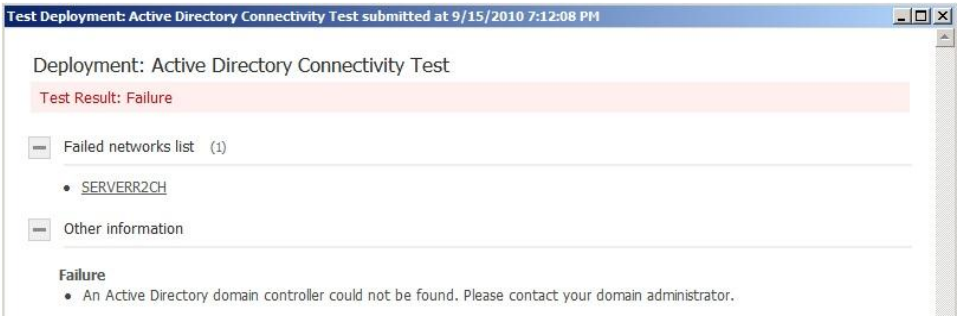



*Figure 6-73 Connectivity test failed*

## 6.5.1 MPI Job Failed

Scenario: Your user complains that an MPI job failed, with the error message of "Can't connect to rank zero process." You can use the following steps to diagnose the problem:

1. Start Cluster Manager.
2. Under **Diagnostics**, click **System**.
3. Under the system suite, select **Firewall Configuration Report**.
4. Find the nodes that the MPI job ran on, and enter them as nodes to test. Click **Run**.
5. Wait for the test to finish. Open the result report, and check whether the MPI port has been closed. The normal result should show running MPI service, as shown in Figure 6-74.

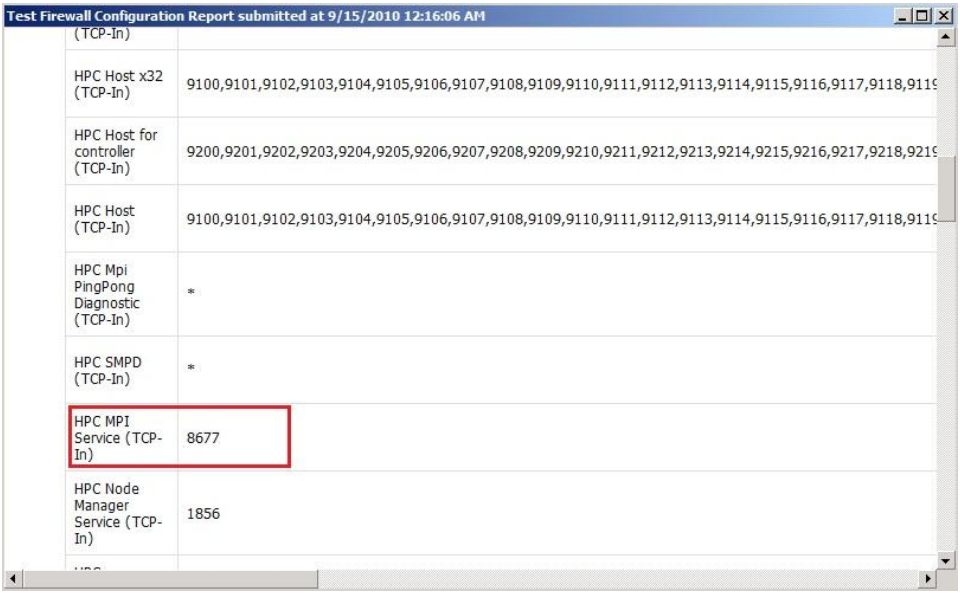
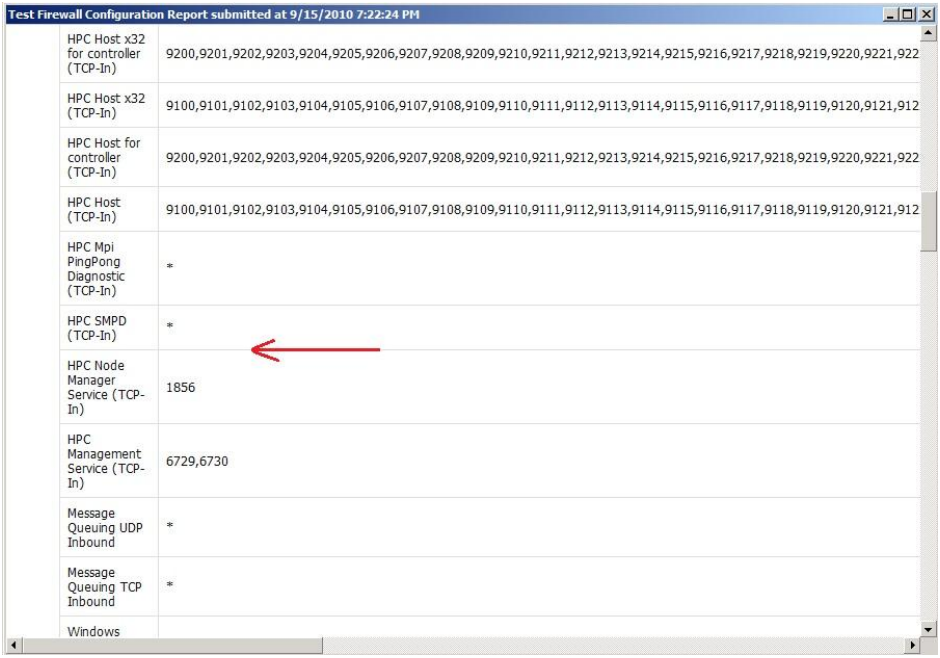



*Figure 6-74 Normal result of firewall port test*

6. However, when MPI service port is closed, the result is shown in Figure 6-75. We do not see MPI service between SMPD and Node Manager Service.



Test Firewall Configuration Report submitted at 9/15/2010 7:22:24 PM

| | |
|---|---|
| HPC Host x32 for controller (TCP-In) | 9200,9201,9202,9203,9204,9205,9206,9207,9208,9209,9210,9211,9212,9213,9214,9215,9216,9217,9218,9219,9220,9221,922 |
| HPC Host x32 (TCP-In) | 9100,9101,9102,9103,9104,9105,9106,9107,9108,9109,9110,9111,9112,9113,9114,9115,9116,9117,9118,9119,9120,9121,912 |
| HPC Host for controller (TCP-In) | 9200,9201,9202,9203,9204,9205,9206,9207,9208,9209,9210,9211,9212,9213,9214,9215,9216,9217,9218,9219,9220,9221,922 |
| HPC Host (TCP-In) | 9100,9101,9102,9103,9104,9105,9106,9107,9108,9109,9110,9111,9112,9113,9114,9115,9116,9117,9118,9119,9120,9121,912 |
| HPC Mpi PingPong Diagnostic (TCP-In) | * |
| HPC SMPD (TCP-In) | * |
| HPC Node Manager Service (TCP-In) | 1856 |
| HPC Management Service (TCP-In) | 6729,6730 |
| Message Queuing UDP Inbound | * |
| Message Queuing TCP Inbound | * |
| Windows | |

*Figure 6-75 MPI service port is closed*

**These ports should be open when the cluster is installed. Why are they closed?**

The system administrator might have created a Group Policy setting that checks and closes open ports. If the Group Policy setting does not identify MPI ports, it may close them. This occurred once to a customer. It took the customer one week to find the root cause. Thus, if any error about system communication occurs during a job run, we suggest running a Firewall Configuration Report test first.

## 6.6 Reporting-Based Decisions

### 6.6.1 Cluster Utilization and Resource Allocation

Scenario: A cluster administrator needs to report cluster utilization and resource allocation to the CIO monthly. The job template provides only a static distribution, whereas reporting can provide the actual usage.

1. Open Cluster Manager.
2. Navigate to **Charts and Reports**, which is shown in Figure 6-76.

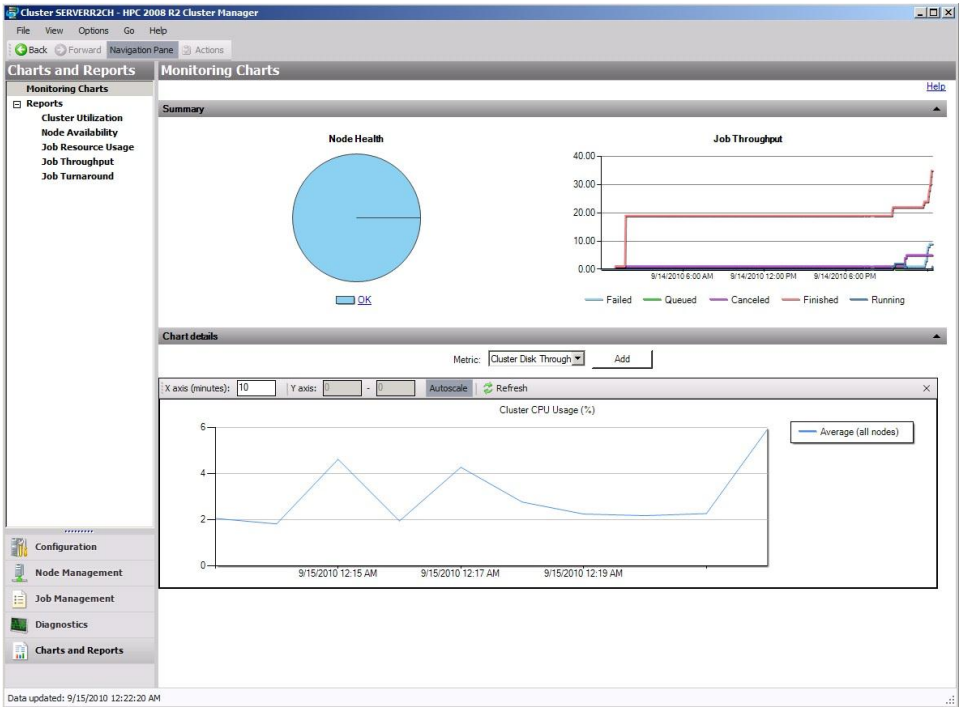



*Figure 6-76* ***Charts and Reports***

3. Click **Reports**, and then **click Job Resource Usage**.
4. In the right panel, click the **Group By** list, select Project, and click **View Report**. You can get the job resource usage report, with CPU time, Wall Clock Runtime, and memory usage.

> **What is CPU time for?**
>
> CPU time covers user-mode and kernel-mode CPU time. In a computational-intensive application, the CPU time/Wall Clock Runtime ratio should be high, and most of the CPU time should be in user mode.

## 6.6.2 Reporting Extensibility

The HPCReporting Microsoft SQL Server® database that Windows HPC Server 2008 R2 provides contains several views that you can query to obtain data for custom reports. Table 6-2 provides reference information about those SQL Server views.

*Table 6-2 Reporting Database Views*

| View name | Description |
|---|---|
| AllocationHistoryView | Lists the allocations of tasks or subtasks in high-performance computing (HPC) jobs to cores on the nodes in the Windows HPC Server cluster. |
| DailyNodeStatView | Provides statistics about the availability and use of nodes to run jobs on a daily basis. Contains one row per combination of node and date. |
| JobHistoryView | Lists each queued instance of a job on a Windows HPC Server cluster that finished, failed, or was canceled. Includes statistics and information about the properties and results, and tasks for the jobs. |
| NodeEventHistoryView | Lists the events that occurred on the nodes of a Windows HPC Server cluster. Events include the addition or removal of nodes and changes to the node state. |
| NodeGroupMembershipView | Indicates the node groups to which the nodes in a Windows HPC Server cluster belong. Contains one row for each combination of node and node group for which the node is a member of the node group. |
| NodeGroupView | Lists the node groups for the Windows HPC Server cluster. |
| NodeView | Lists the nodes in a Windows HPC Server cluster. |

Scenario: A cluster administrator runs a fixed set of jobs every day and wants to have a convenient way to view the health of those jobs for the current day. The administrator wants to check not only whether the jobs succeeded or failed, but also whether the jobs took the expected amount of time. The administrator has estimates of how long each of the jobs should take to run and does not expect the duration of the jobs to vary from the estimates by more than 3 percent.

This example uses features of Microsoft Office Excel® 2007 to get the job history data from the HPCReporting database and to analyze the data and summarize the results. To indicate the health of the job, the example uses the values in the following table.

*Table 6-3Health Definition Table*

| **Health value** | **Description** |
|---|---|
| Red | The job failed or was canceled. |
| Yellow | The job finished, but the duration of the job differed from the expected duration by more than 3 percent. |
| Green | The job finished, and the duration of the job differed from the expected duration by less than 3 percent. |

This example makes the following assumptions:

- A set of jobs is created and submitted to the Windows HPC Server cluster on a daily basis through a mechanism such as a scheduled task that runs a script or program to create the jobs in an automated fashion.
- Each job in this set of jobs has a name, and that name remains the same every day. The name for a job in the set of daily jobs should not be the same as the name of any other job in the set or any job for the Windows HPC Server cluster that is not part of this daily set of jobs.
- The administrator already has estimates for the amount of time that the jobs should take to run.
- The HPCReporting database in this example is either the live HPCReporting database or a database with the same schema and with data that you transferred from the live HPCReporting database. The latter scenario is recommended.
- Excel is installed on a computer that has access to the HPCReporting database described in the previous point.

**Note**

You need to refresh the report each day to get the current data for the report. After the report is refreshed, the data in the report applies only to the most recent submission of the jobs that the report tracks, and the report does not track the state or performance of the jobs over multiple days. You need to save the report data daily or modify the report to track the history of the job state and performance.

### 6.6.2.1 Step 1: Create a Connection to the SQL Server Database to Get Job History Data

To create this report, you first need to get the relevant data from JobHistoryView in the HPCReporting database for the cluster, or a copy of that database.

1. In Excel, click the **Data** tab. In the **Get External Data** group, click **From Other Sources**, and then click **From SQL Server**, as shown in Figure 6-77.

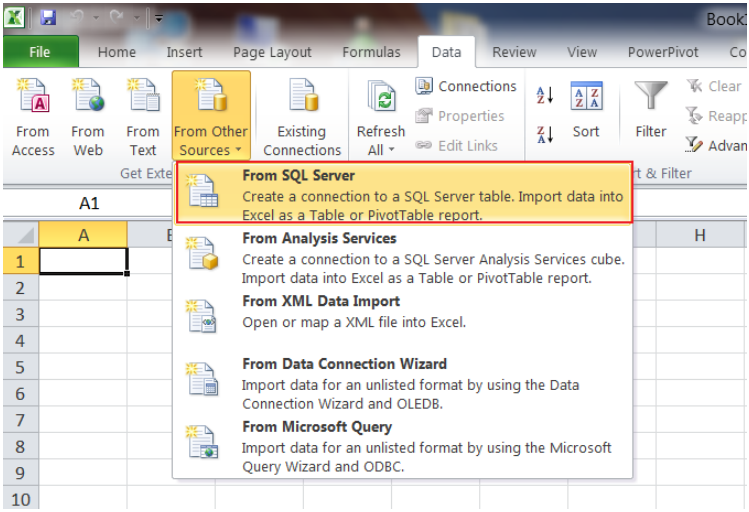



*Figure 6-77 Choose data source*

2. On the **Connect to Database Server** page of the Data Connection Wizard, In the **Server name** box, type the name of the computer and SQL Server instance that hosts the HPCReporting database for your Windows HPC Server cluster or a database that contains a copy of the data from that database.

   The name that you type should have a format of *ServerName*[*InstanceName*]. You can omit the instance name if you transferred the reporting data to the unnamed default SQL Server instance on another server.

3. Click **Use Windows Authentication**, as shown in Figure 6-78, and then click **Next**.

   Or, if you want to connect to the database with a different account or do not want to use Windows authentication to connect to the database, click **Use the following User Name and Password**, type a user name and password for an account that can access the database in the **User Name** and **Password** boxes, and then click **Next**.

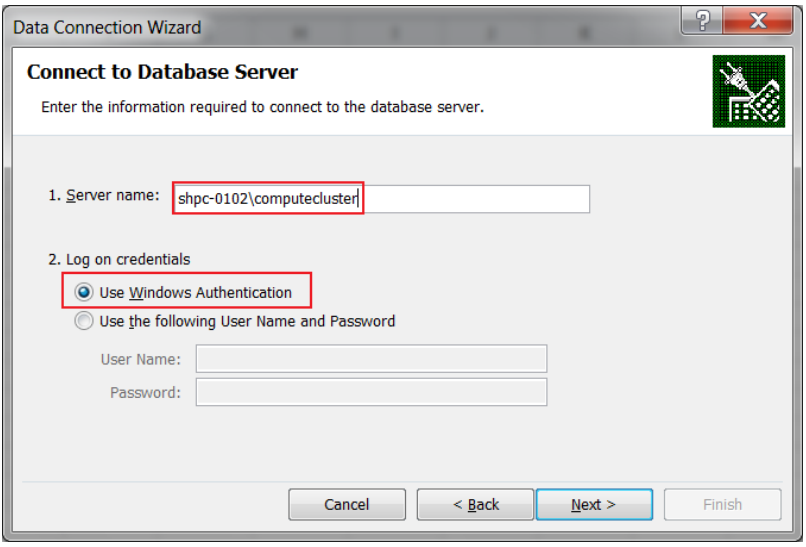



*Figure 6-78* ***Connect to Database Server*** *page*

4. On the **Select Database and Table** page, select **HPCReporting** from the **Select the database that contains the data you want** list, select the **Connect to a specific table** check box, click **JobHistoryView**, and then click **Next**, as shown in Figure 6-79.

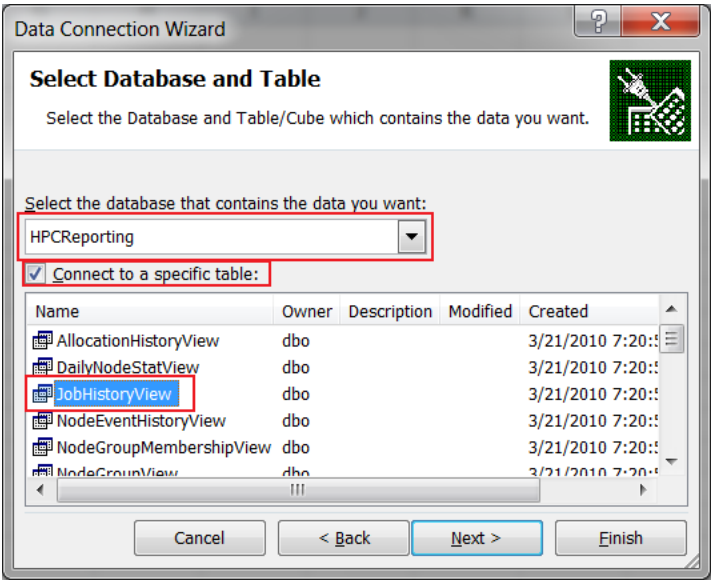



*Figure 6-79* ***Select Database and Table*** *page*

5. On the **Save Data Connection File and Finish** page, type a description for the connection in the **Description** box (as shown in Figure 6-80), and then click **Finish**.

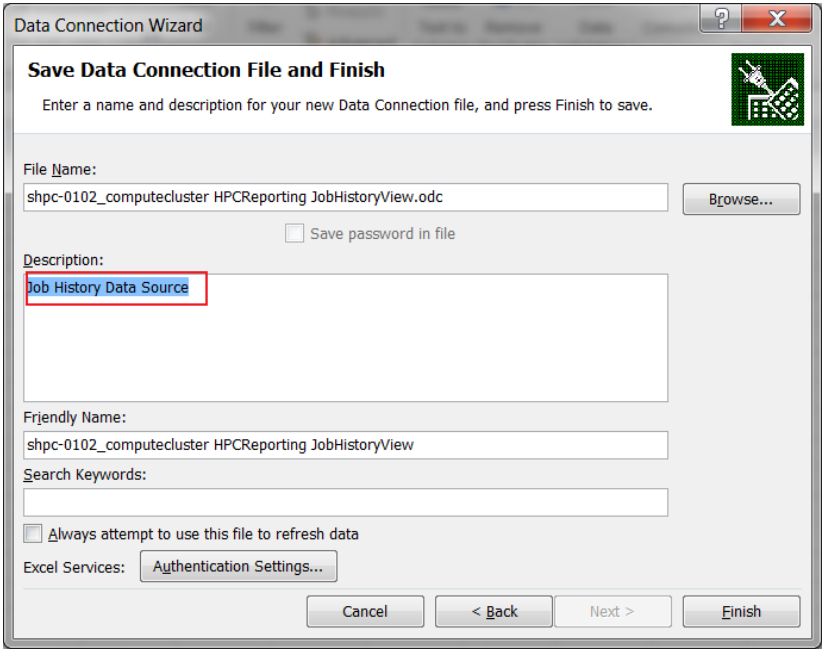



*Figure 6-80* ***Save Data Connection File and Finish*** *page*

6. In the **Import Data** dialog box, click **Properties**, as shown in Figure 6-81.

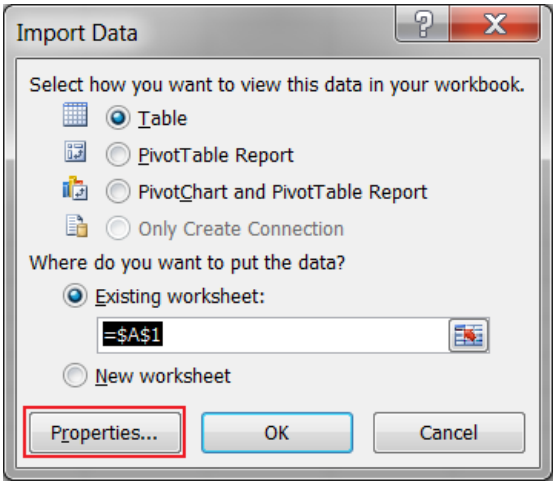



*Figure 6-81* ***Import Data*** *dialog box*

7. On the **Usage** tab of the **Connection Properties** dialog box, select the **Refresh every 60 minutes** and **Refresh data when opening the file** check boxes (as shown in Figure 6-82), and then click **OK**.

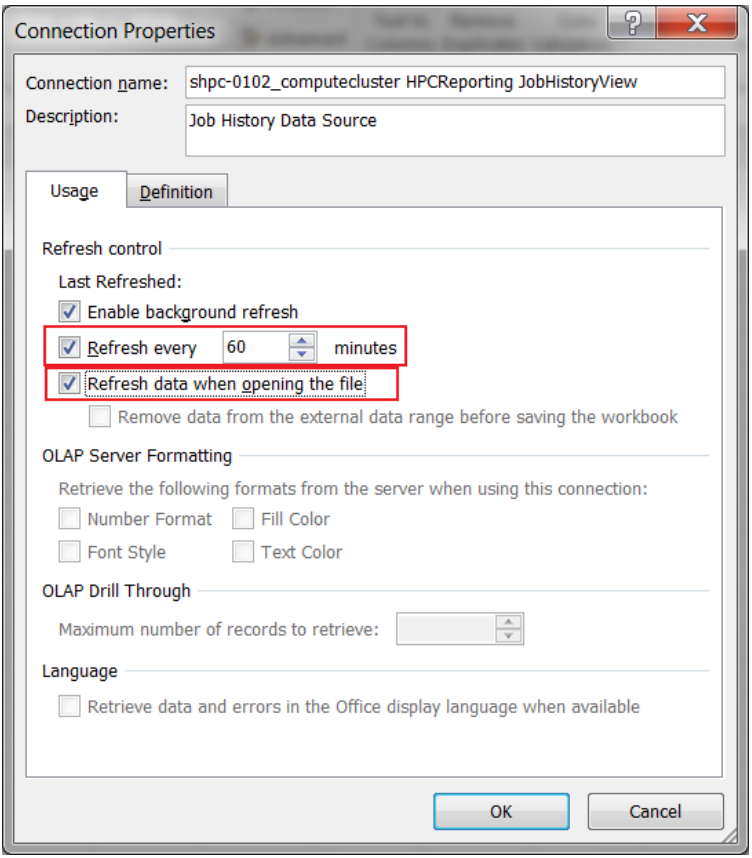



*Figure 6-82* ***Connection Properties*** *dialog box*

8. In the **Import Data** dialog box, click **OK**.
9. Double-click the tab on the worksheet that populates with the job history data, and change the name to **Job History Details**.

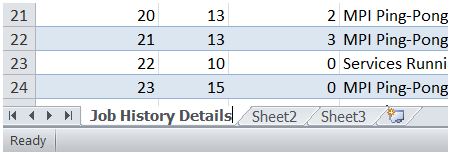



*Figure 6-83* ***Job History Details*** *tab*

10. Leave the Excel file open for the next section.

### 6.6.2.2 Step 2: Create the Report

After you have set up the Excel file so that it has the relevant job history data, you can set up a worksheet for the report.

To create the report, you need to add column headings, and then fill in the names of the jobs that you want to track and the amount of time you expect each job to take. Then, you add a **Refresh** button to the form and create Microsoft Visual Basic® Scripting Edition (VBScript) code for a macro that you assign to the button. When the user clicks **Refresh**, the script will calculate and fill in the information for the rest of the report.

1. In the Excel file that you created by following the previous procedure in this topic, double-click the tab for an empty worksheet and change the name of the worksheet to **Summary**, as shown in Figure 6-84.

*Figure 6-84 Naming worksheet*

2. In the first row of the **Summary** tab, add values to the first five cells to create headings for the report, as shown in Figure 6-85.

| Job Name | Expected Job Runtime (ms) | Job State | Actual Job Duration (ms) | Health |
|---|---|---|---|---|

| | A | B | C | D | E |
|---|---|---|---|---|---|
| 1 | Job Name | Expected Job Runtime (ms) | Job State | Actual Job Duration (ms) | Health |
| 2 | | | | | |
| 3 | | | | | |
| 4 | | | | | |

*Figure 6-85 Naming the columns*

3. In the **Job Name** column, add the names of the jobs for which you want to track the job health. Add each name only once.
4. In the **Expected Job Runtime (ms)** column, add the amount of time in milliseconds that you expect the corresponding jobs in the **Job Name** column to take, as shown in Figure 6-86.

| | Job Name | Expected Job Runtime (ms) |
|---|---|---|
| 2 | job1 | 350000 |
| 3 | job2 | 350000 |
| 4 | job3 | 350000 |
| 5 | job4 | 350000 |
| 6 | job5 | 350000 |

*Figure 6-86 Adding job name and expected job run times*

5. If the **Developer** tab is not visible, perform the following actions to make it visible:

   - Click the Office button, and then click **Excel Options**.
   - In the **Excel Options** dialog box, click **Popular**, select the **Show Developer tab in the Ribbon** check box, and then click **OK**.

6. On the **Developer** tab, click **Insert**, and then click the button control in the upper left under **Form Controls**, as shown in Figure 6-87.

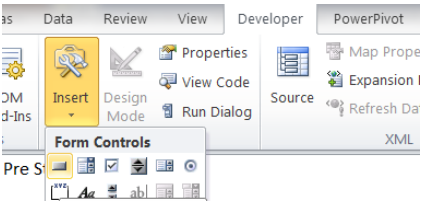



*Figure 6-87 Inserting button control*

7. Near the top of the sixth column of the sheet, drag the cursor to create a button of the size that you want, as shown in Figure 6-88.

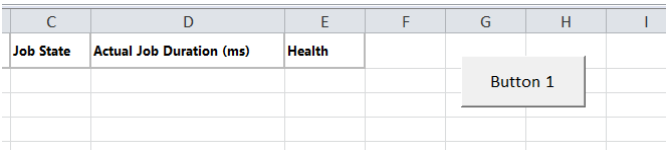



*Figure 6-88 Creating a button*

8. In the **Assign Macro** dialog box, click **New**, as shown in Figure 6-89.

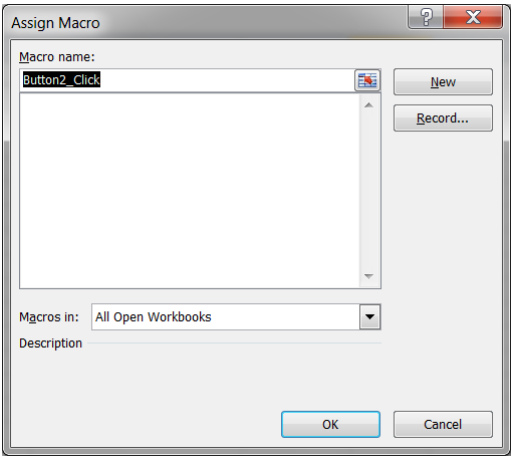



*Figure 6-89 Assigning a macro*

9. In the Visual Basic editor, in the **Book1 - Module1 (Code)** window (shown in Figure 6-90), replace the existing code with that shown in Code 6-1. The result should look like Figure 6-91.

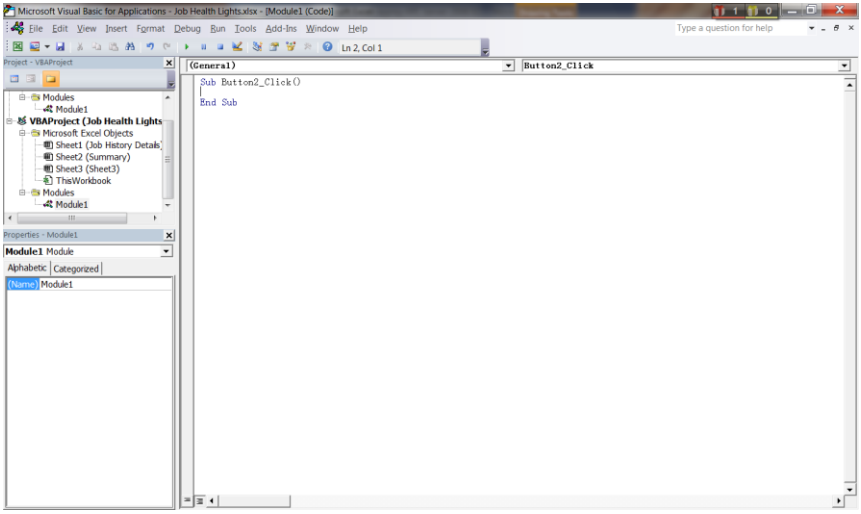



*Figure 6-90 Visual Basic editor*

```
Option Explicit
Option Compare Text

Function GetColorFromString(colorString As String) As Long

    If colorString = "Red" Then
        GetColorFromString = RGB(255, 0, 0)
    ElseIf colorString = "Green" Then
        GetColorFromString = RGB(0, 255, 0)
    ElseIf colorString = "Yellow" Then
        GetColorFromString = RGB(255, 200, 0)
    Else
        GetColorFromString = RGB(0, 0, 255)
    End If

End Function

Sub Button1_Click()

    Dim jobList As New Collection
    Dim expectTimeList As New Collection
    Dim jobTimeList As New Collection
    Dim populateStartIndex As Long
    Dim i As Long
    Dim j As Long
```

```
    populateStartIndex = 4
    With Worksheets("Summary")
        For i = 2 To 5000
            If (IsEmpty(.Cells(i, 1))) Then
                populateStartIndex = i
                i = 5000
                ' Break.
            Else
                jobList.Add (.Cells(i, 1))
                expectTimeList.Add (.Cells(i, 2))
                Dim minDate As Date
                minDate = "1900-01-01"
                jobTimeList.Add (minDate)
                ' Clear the last three columns.
                Worksheets("Summary").Cells(i, 3).ClearContents
                Worksheets("Summary").Cells(i, 4).ClearContents
                Worksheets("Summary").Cells(i, 5).ClearContents
            End If
        Next i
        ' Clear the rows after the first row without a job name.
        For i = populateStartIndex + 1 To 2000000000
            If (IsEmpty(.Cells(i, 1))) Then
                i = 2000000000
                ' Break.
            Else
                .Cells(i, 1).EntireRow.ClearContents
            End If
        Next i
    End With

    With Worksheets("Job History Details")
        For i = 2 To 2000000000
            If (IsEmpty(.Cells(i, 1))) Then
                i = 2000000000
                ' Break.
            Else
                ' Get the actual run time and event for the job.
                Dim evt As String
                Dim light As String
                Dim jobName As String
```

```
                Dim runTime As Long
                Dim expectedTime As Long
                Dim evtTime As Date

                jobName = .Cells(i, 4)
                evt = .Cells(i, 5)
                evtTime = .Cells(i, 6)
                runTime = .Cells(i, 19)
                ' In case the job is not found.
                light = "Black"
                expectedTime = -1

                ' Test if the job is in the list of jobs to track.
                For j = 1 To jobList.Count
                    If jobList.Item(j) = jobName Then
                        If (Not evt = "Finished") Then
                            light = "Red"
                        Else
                            expectedTime = expectTimeList.Item(j)
                            If (runTime) >= expectedTime * 1.03 Or
runTime <= expectedTime * 0.97 Then
                                light = "Yellow"
                            Else
                                light = "Green"
                            End If
                        End If

                       ' Update the job later.
                        If jobTimeList.Item(j) < evtTime Then
                            jobTimeList.Remove (j)
                            If jobTimeList.Count = 0 Then
                                jobTimeList.Add (evtTime)
                            ElseIf j > 1 Then
                                jobTimeList.Add evtTime, , , j - 1
                            Else
                                jobTimeList.Add evtTime, , j
                            End If

                        Worksheets("Summary").Cells(j + 1, 3) = evt
                        Worksheets("Summary").Cells(j + 1, 4) = runTime
```

```
                        Worksheets("Summary").Cells(j + 1,
5).Hyperlinks.Delete

                        Worksheets("Summary").Cells(j + 1,
5).Hyperlinks.Add Anchor:=Worksheets("Summary").Cells(j + 1, 5),
Address:="", _

                              SubAddress:=("'Job History Details'!A" +
Trim(Str(i)) + ""), ScreenTip:="click to go to detailed row",
TextToDisplay:=light

                        Worksheets("Summary").Cells(j + 1,
5).Font.Color = GetColorFromString(light)

                        End If


                    End If

                Next j

            End If

        Next i

    End With

End Sub
```

*Code 6-1 Visual Basic macro*

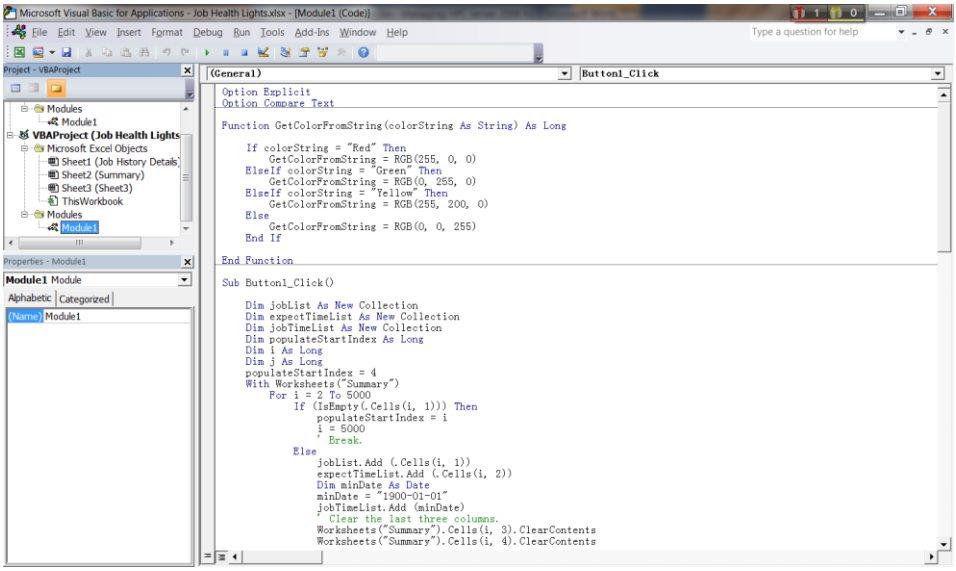



*Figure 6-91 Macro code*

10. On the **File** menu, click **Close and Return to Microsoft Excel**, as shown in Figure 6-92.

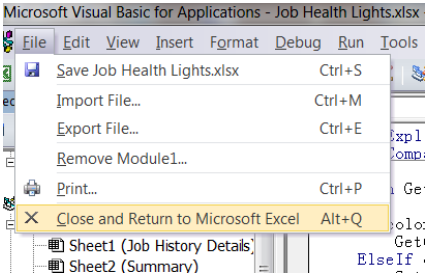



*Figure 6-92 Closing Visual Basic editor*

11. Right-click the button that you created, click **Edit Text**, and then change the text to **Refresh**, as shown in Figure 6-93.

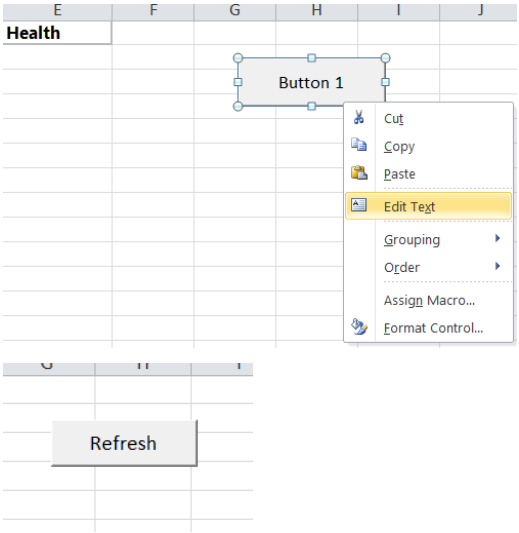



*Figure 6-93 Refreshing button text*

12. Click the **Refresh** button. The macro fills in the third, fourth, and fifth columns to complete the report, as shown in Figure 6-94.

| | Job Name | Expected Job Runtime (ms) | Job State | Actual Job Duration (ms) | Health |
|---|---|---|---|---|---|
| 1 | | | | | |
| 2 | job1 | 350000 | Finished | 368090 | Yellow |
| 3 | job2 | 350000 | Finished | 340123 | Green |
| 4 | job3 | 350000 | Finished | 373564 | Yellow |
| 5 | job4 | 350000 | Finished | 299369 | Yellow |
| 6 | job5 | 350000 | Failed | 344535 | Red |
| 7 | | | | | |
| 8 | | | | | |

Refresh

*Figure 6-94 Report result*

13. Save the file.

You can now open this Excel file on future days, click **Refresh** on the **Summary** tab, and get the current status of the jobs for that day. If some jobs failed, you can investigate the failures and rerun the failed jobs. If the actual duration of some jobs differed too much from the expected duration, you can investigate the causes.

# Chapter 7 Building VBA Applications and Workbooks for a Windows HPC Server 2008 R2 Cluster

Would you spend one hour to make your Microsoft® Excel® workbook run four times faster? How about eight times faster, or even more? If you have long-running or slow Excel workbooks, you can use HPC Services for Excel with a Windows® HPC Server 2008 R2 cluster to dramatically improve calculation performance.

HPC Services for Excel supports a variety of new programming models so that you can run Excel calculations on a Windows HPC Server cluster. But you don't need to understand code development, or know C#, to use HPC Services for Excel. In this chapter, we'll discuss how to build workbooks that utilize a Windows HPC Server cluster for calculation, by using only Excel-based Microsoft Visual Basic® for Applications (VBA).

In reality, it will probably take more than an hour to convert one of your workbooks to run on the cluster, depending on how complicated the workbook is. But if you spend an hour working through the examples in this chapter, you'll understand the concepts involved and the requirements for converting just about any workbook to run on a Windows HPC Server cluster. You'll also see the performance benefits that you can get with some simple changes to your VBA code.

Chapter highlights:

- The VBA programming model of HPC Services for Excel
- How to write a new high-performance computing (HPC)–enabled workbook
- How to convert an existing workbook to the one that can be run on a cluster

## 7.1 Structure and Formatting

The screen shots in this chapter all come from Microsoft Excel 2010 (Beta) and Windows HPC Server 2008 R2 (Beta 2) Cluster Manager. In the hands-on examples included in this chapter, code examples are formatted with a VBA editor style, as shown in Code 7-1.

```
'==========================================================
'
' Section 1: Variables and constants
'
'==========================================================


Dim SentRecords As Integer
Dim RcvRecords As Integer
```

*Code 7-1 Formatting example*

When lines of code are added or modified, they'll be highlighted in green. Much of the code included here is either imported from existing files or copied between different VBA files. In that case, some long lines of code may be truncated to fit within the margins of this document. Try to copy and paste from existing documents rather than from this chapter, to avoid accidentally including the truncated code examples.

## 7.2 Overview

HPC Services for Excel supports a number of programming models for developing and running Excel calculations on a Windows HPC Server 2008 R2 cluster. The solution that you select should depend on your particular needs and your development skills, as shown in Table 7-1.

*Table 7-1 Overview of HPC Services for Excel*

| **Intended use** | **Skills required** | **Solution** |
|---|---|---|
| Calculate workbooks in parallel, for better performance (faster calculation times) or the ability to process more data in the same amount of time | Excel VBA (macro) development | Executing workbooks on an Windows HPC Server cluster with Excel VBA |
| Build custom applications and services that execute Excel workbooks on a Windows HPC Server cluster and integrate with stand-alone applications or batch scripts | Microsoft .NET programming using C#, VB.NET, or any language supported by the .NET Common Language Runtime | Developing custom HPC services and client applications for Excel |
| Speed up calculation of Excel user-defined functions (UDFs) contained in Excel extension libraries (XLLs) | C or C++ programming, XLL development | Cluster-enabled Excel UDFs |

To support Excel developers, HPC Services for Excel includes a set of libraries that you can use directly from Excel—on your desktop—to build and run Excel workbooks on a Windows HPC Server 2008 R2 cluster. With these libraries, you don't need to know .NET programming or C#. You can do all the work from Excel by writing VBA macros. With this framework, you can dramatically speed up the calculation of Excel workbooks simply by writing VBA code and executing the workbooks on the cluster.

HPC Services for Excel includes a client/service framework specifically designed to be used from Excel on the desktop. There are two important parts to this framework: a client library, which is installed on the desktop; and a service library, which is installed on the cluster compute nodes. These libraries are installed by default when you install Windows HPC Server 2008 R2 Beta, so you won't need to do any custom configuration to use them.

After the client and service library are installed, you can build and run Excel workbooks for the cluster simply by adding some VBA macros to your workbook. We'll describe these new VBA macros as the *macro framework*. When we refer to the macro framework, we're specifically describing the new macros that you add to your workbooks to support calculation on a Windows HPC Server cluster.

Before we discuss the specific macros, it's worth highlighting a few key concepts that affect how the macros are constructed and how they work.

### 7.2.1 Event-Based Asynchronous Calculation

The most important concept to understand in developing workbooks for HPC Services for Excel, with the macro framework, is *asynchronous calculation*.

A typical Excel application runs *synchronously*. That is, you click a button to run a macro, and the macro executes some VBA code. Within this code, you might make changes to the spreadsheet, call other VBA functions, access databases, or do anything else. But you expect that the macros will run from beginning to end, and then finish.

The *asynchronous* model is a little different. In this model, you write VBA functions, but you don't call them directly. Instead, a client library—code that's installed with HPC Services for Excel—calls your VBA functions.

If you're familiar with Excel events in VBA, you already know this model. For example, if you write a VBA function that runs on the **Workbook.Open** event, you know that the code will run whenever the workbook is opened. But you don't call that code directly; rather, Excel automatically calls the function that you write when the workbook is opened.

The asynchronous programming model works the same way. You write functions that will be called during a calculation, but instead of calling them yourself, you wait for Excel—or in this case, the HPC Services for Excel library—to call the functions.

### 7.2.2 Macro Framework

To support HPC Services for Excel, your workbook needs to include a set of macros that implement the asynchronous functions. This is the macro framework. Seven macros are used in the framework; you can add them to any Excel workbook. The seven macros are shown in Table 7-2.

*Table 7-2 Descriptions of the Seven Macros in the Macro Framework*

| | |
|---|---|
| HPC_Initialize | Perform any pre-calculation or initialization steps |
| HPC_Partition | Collect required parameters for a single calculation step |
| HPC_Execute | Perform one calculation step, as part of a larger overall calculation |
| HPC_Merge | Process the results of a single calculation step |
| HPC_Finalize | Perform any post-calculation processing |
| HPC_ExecutionError | Handle errors arising from the calculation |
| HPC_GetVersion | Return the version of the macro framework used in the workbook |

(Don't worry if those descriptions aren't immediately clear. We'll describe each of these functions in more detail later.)

The same macros have to be added to any spreadsheet to support calculation on the cluster, but what's in the macros—the specific calculation functions—can be different. To simplify the process, the download files included with this chapter include a "skeleton" macro file that you can import into your workbook. After you have the basic macros in place, you can fill them in as necessary to support your calculation.

Each macro has a specific purpose in the macro framework, and each will be called by the HPC Services for Excel client library during a cluster calculation. When a calculation runs, each macro will be called at a specific point. A typical calculation runs as the flow shown in Figure 7-1.

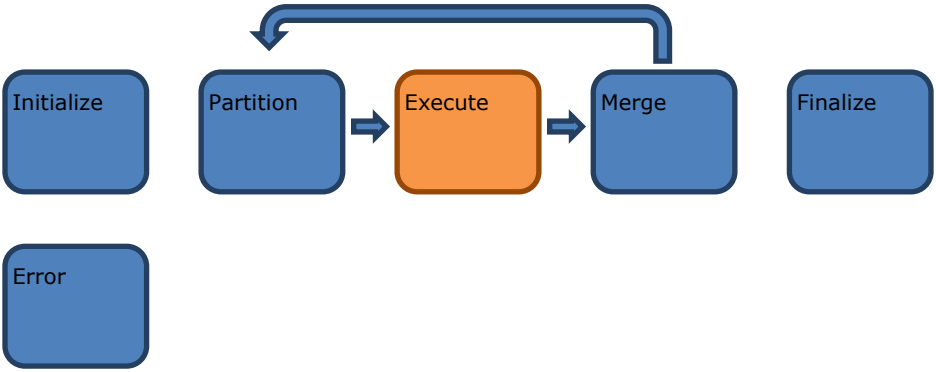



*Figure 7-1 Cluster execution workflow*

When a cluster calculation runs, first the client library calls the **HPC_Initialize** macro. That's used to handle any required initialization; for example, you might want to clear out old results from your spreadsheet.

Next, the calculation runs through three macros: **HPC_Partition**, **HPC_Execute**, and **HPC_Merge**. In the preceding diagram, these are shown as a loop. That's not really what happens during a calculation, but logically you can think of this as a loop. First, the client library calls **HPC_Partition**. **HPC_Partition** is designed to collect any data required for a single calculation step. For example, if you want to calculate a set of rows one by one, **HPC_Partition** might return the row number for a single step: first row 1, then row 2, and so on.

Next, the **HPC_Execute** macro is called. This macro runs the actual calculation. If you're calculating row by row, this will calculate a single row. **HPC_Execute** will return the result of the calculation: For example, it might return the last cell in the row, which is the end result of a longer calculation.

After **HPC_Execute**, the client library calls **HPC_Merge**. Whatever **HPC_Execute** has calculated is sent to **HPC_Merge**. The **HPC_Merge** macro is designed to take those results and return them to the spreadsheet running on your desktop. In the **HPC_Merge** macro, you might insert the calculation results back into your spreadsheet, or you might write them to a log file.

These three macros—**HPC_Partition**, **HPC_Execute**, and **HPC_Merge**—are called multiple times, once for each step in your calculation, until the entire calculation is complete. These macros implement the *iterative* calculation model discussed previously. The macros might be called many times during a single workbook calculation, but each time they are called represents a single calculation step, or *iteration*.

After the last calculation step has finished, the client library calls **HPC_Finalize**. You can use this macro to perform any post-calculation processing: for example, you might want to find the average of all the previous calculation steps by using another VBA macro.

The sixth macro, **HPC_ExecutionError**, is used only when the calculation encounters an error. If so, this macro is called and you can add some error handling code—for example, you might show a pop-up dialog box, warning the user of the error.

The seventh and final macro—**HPC_GetVersion**—has a special function. It tells the HPC libraries which version of the macro framework the workbook contains. The macro framework described in this document is version 1.0, so the **HPC_GetVersion** macro must return the string "1.0".

The **HPC_GetVersion** macro is important because it will allow the workbook to run with any future version of Windows HPC Server. Even if the default macro structure changes in future releases, as long as the **HPC_GetVersion** macro returns "1.0", the HPC libraries will be able to execute this workbook on a Windows HPC Server cluster.

In Figure 7-1, the **HPC_Execute** macro is highlighted in orange. That's to indicate that this macro is somewhat unusual. All the other macros run on the desktop, in Excel on your workstation. When you run a calculation on a Windows HPC Server cluster, however, the **HPC_Execute** macro actually runs on the cluster compute nodes.

That's how HPC Services for Excel, and the macro framework, support cluster calculation. While you write all the macros in the same workbook—the workbook on your desktop—when the calculation runs, the workbook will be copied to the cluster compute nodes and the **HPC_Execute** macro will be run on one of the compute nodes.

It's important to understand this, because it has some implications for how workbooks use and manage data during the calculation. The three main calculation macros—**HPC_Partition**, **HPC_Execute**, and **HPC_Merge**—pass data back and forth during the calculation process. In a typical VBA application, there are many ways to share data between different macros. For example, you can use global variables in your VBA code, or write values into spreadsheet cells. But when you're running the calculation on a Windows HPC Server cluster, you need to send data between the macros running on the desktop and the **HPC_Execute** macro running on the compute nodes.

For that reason, any data that you need to run the calculation must be sent from the **HPC_Partition** macro to the **HPC_Execute** macro directly. You do this by using the *return value* from the **HPC_Partition** macro. That return value becomes the *argument* (or *parameter*) to the **HPC_Execute** function. So data will pass directly between these two macros, and you can pass any value or set of values (as an array) between these macros.

But again, you can't use global variables or the spreadsheet cells to pass information between the two macros because when it runs on the cluster, there are actually multiple copies of the workbook running at the same time—one on the desktop, and one (or more) on the cluster compute nodes.

In the same way, when the calculation in **HPC_Execute** is complete, it returns a result as the return value of the macro function. That result becomes the argument (or parameter) to the next macro, **HPC_Merge**. Once again, any results that you want to return—the results of the calculation—must be sent in this way, as the return value of the function.

In the sections that follow, you'll walk through building two workbooks that support calculation on a Windows HPC Server cluster. In the first section, you'll build a workbook from scratch. This should help you understand the concepts involved and the requirements for using HPC Services for Excel. In the second section, you'll take an existing workbook—one that was designed to run on the desktop—and convert it so it can run on the cluster.

## 7.3 Prerequisites and Requirements

HPC Services for Excel is a set of tools included with the Enterprise edition of Microsoft HPC Pack 2008 R2 and later versions. You'll need a Windows HPC Server 2008 R2 cluster installed and configured.

Installing a Windows HPC Server cluster is outside of the scope of this chapter; we'll address only the specific configuration that you will need for HPC Services for Excel. For more information about installing and configuring the cluster, see the documentation included with HPC Pack 2008 R2. On your desktop, you'll need the HPC Pack 2008 R2 client components.

Note that you must have the Enterprise edition of Windows HPC Server 2008 R2 installed on your cluster scheduler and cluster compute nodes. The Enterprise edition includes support for executing Excel workbooks on the cluster.

**To verify that HPC Services for Excel is configured correctly**

1. In HPC Cluster Manager, click Diagnostics.
2. In Tests, expand Microsoft, and then select Excel, as shown in Figure 7-2.

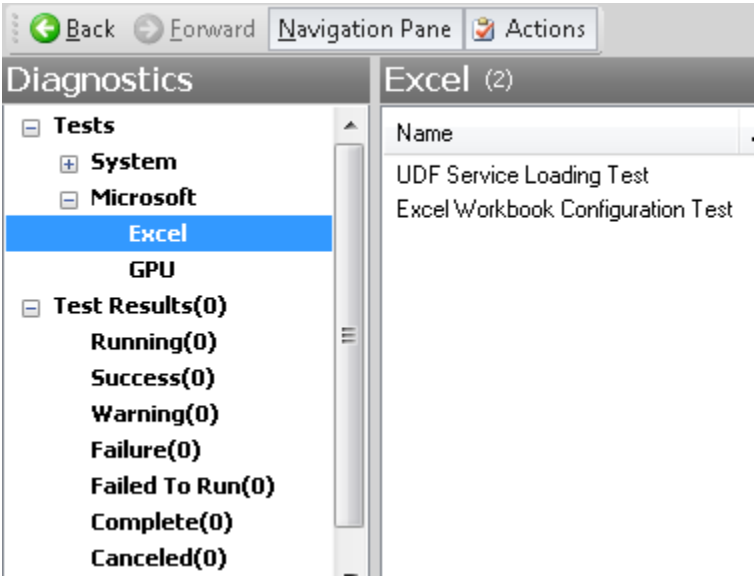



*Figure 7-2 HPC services for Excel diagnostic tests*

3. In the view pane, double-click **Excel Workbook Configuration Test**.
4. In the configuration dialog box, select **All Nodes**, and then click **Run**, as shown in Figure 7-3.

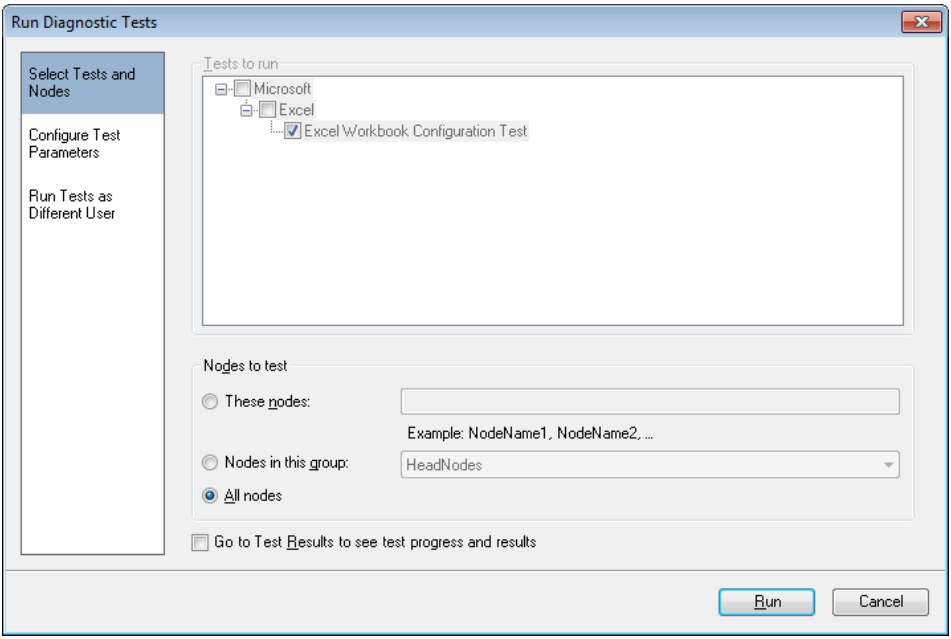



*Figure 7-3 Run diagnostic tests*

5. In the **Navigation Pane**, click **Test Results** to see the progress of the test. The test that you are running appears in the view pane.
6. After the test is complete, you can click the test in the main window to see the results.

If the test shows **Failure**, double-click the test and try to correct any errors in the pop-up window. Common errors you might see are that Excel is not installed on a compute node, or that Excel has not been activated. Make any changes you need, and then rerun the diagnostic test by using the preceding instructions.

When the test shows **Success**, you're ready to continue.

The last thing you'll need is a share directory. When you calculate a workbook on a Windows HPC Server cluster, each of the cluster compute nodes actually loads and runs a copy of the workbook. To support that, you need to create a share directory that is visible both to the cluster compute nodes and to your desktop.

If you have access to the cluster head node, create a share directory on the head node. This is convenient because you know that the cluster compute nodes can access the head node. If you can't create a share directory on the head node, create the share directory anywhere within the domain that both your desktop and the cluster compute nodes can access.

## 7.4 Part I: Building a Workbook from Scratch

This part includes the following activities:

1. Starting a new workbook
2. Preparing the workbook
3. Defining the calculations in the macro framework
4. Running the workbook locally and exploring the macros
5. Running the workbook on the cluster
6. Reviewing what happens when a workbook runs on the cluster

### 7.4.1 Starting a New Workbook

Start Excel 2010 with a new workbook. You'll be using VBA, so make sure that you have the **Developer** tab available on the Excel ribbon.

**To view the Developer tab**

1. On the Excel ribbon, click the **File** tab.
2. Click **Options**.
3. In the **Excel Options** dialog box, click **Customize Ribbon** on the left.
4. Select the box next to **Developer**, and then click **OK**, as shown in Figure 7-4.

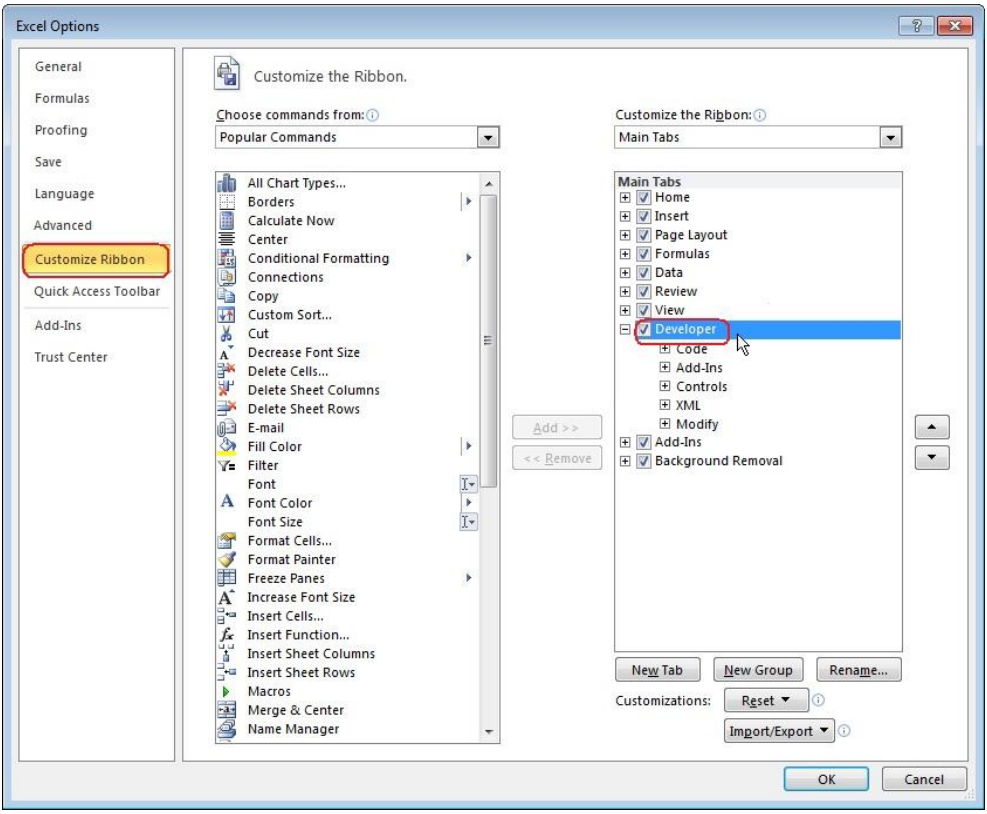



*Figure 7-4 Open Developer tab*

## 7.4.2 Preparing the Workbook

To prepare your workbook, you need to import the macro files and add the HPC references.

To build the basic workbook, you'll need two sets of macros. The first set of macros implements the framework functions (the functions described earlier in the "Overview" section). The HPC Services for Excel client library uses these macros during the calculation. The second set of macros contains the control functions—these macros run the actual calculation.

The sample files include skeleton macro files for each of these that you can use to get started. You don't have to use these files—you can also write your own, if you prefer—but the macros themselves are generic, and you may find that using these files is all you need. You can also modify these files to make any changes that you need for your particular workbook.

You'll need to add a reference to this project to use HPC Services for Excel. **Microsoft_Hpc_Excel** is the client library, which provides an object that you can use to control a cluster calculation.

**Note**

In the earlier "Overview" section, we described the tools as a client/server framework. The client library was installed when you installed the HPC Pack 2008 R2 client components, and it will now be available in VBA.

**To import the macro files and add references**

1. On the **Developer** tab of the ribbon, click **Visual Basic** to open the VBA editor.
2. In the VBA editor, right-click **VBA Project** at the top of the tree view, and then click **Import File**, as shown in Figure 7-5.

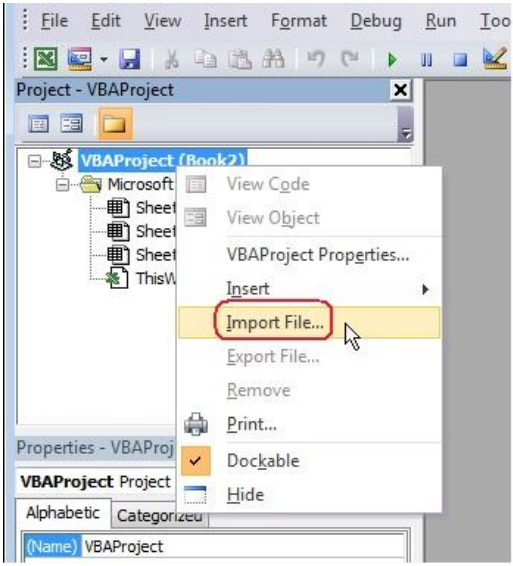



*Figure 7-5 Import the first sample workbook*

3. In the dialog box, browse to the Chapter 7\First Workbook folder, click the file **HPC Excel Macros.bas**, and then click **OK**. That adds the first macro file to your project, and it will be available in the Modules folder of the VBA project window, as shown in Figure 7-6.
4. Repeat steps 2 and 3 to import HPC Control Macros.bas, and then click **OK**. That adds the second macro file to your project.

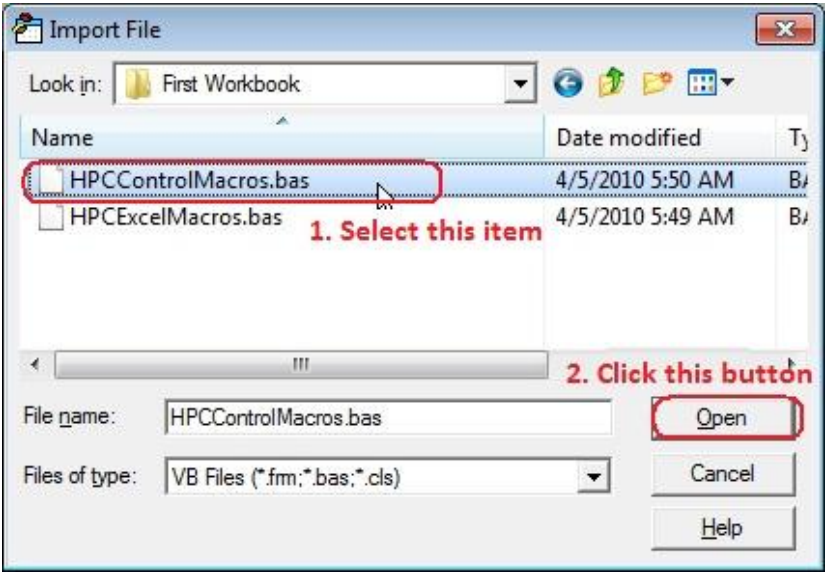



*Figure 7-6 Add a new macro file*

5. Verify that these two macro files appear in the VBA project window, as shown in Figure 7-7.

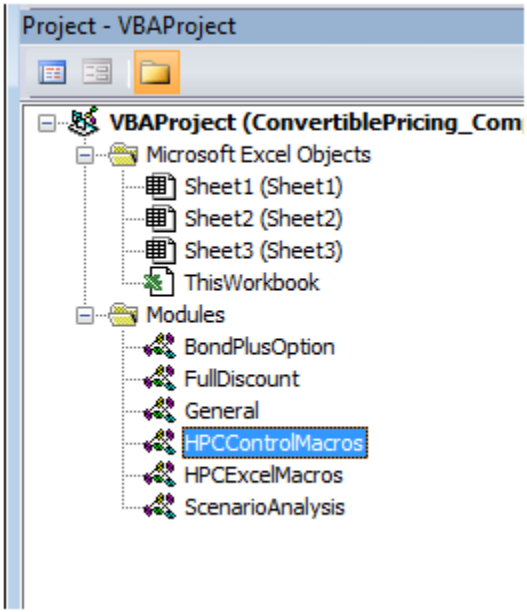



*Figure 7-7 VBA Project files*

6. In the VBA editor window, click **Tools**, and then click **References** to open the project references dialog box.
7. In the list of available references, scroll down until you find **Microsoft_Hpc_Excel** and select the check box next to it. Note that the reference name includes the underscore character (_), so it will come at the end of all the other Microsoft entries. When you've selected the check box, the dialog box should look like that shown in Figure 7-8.

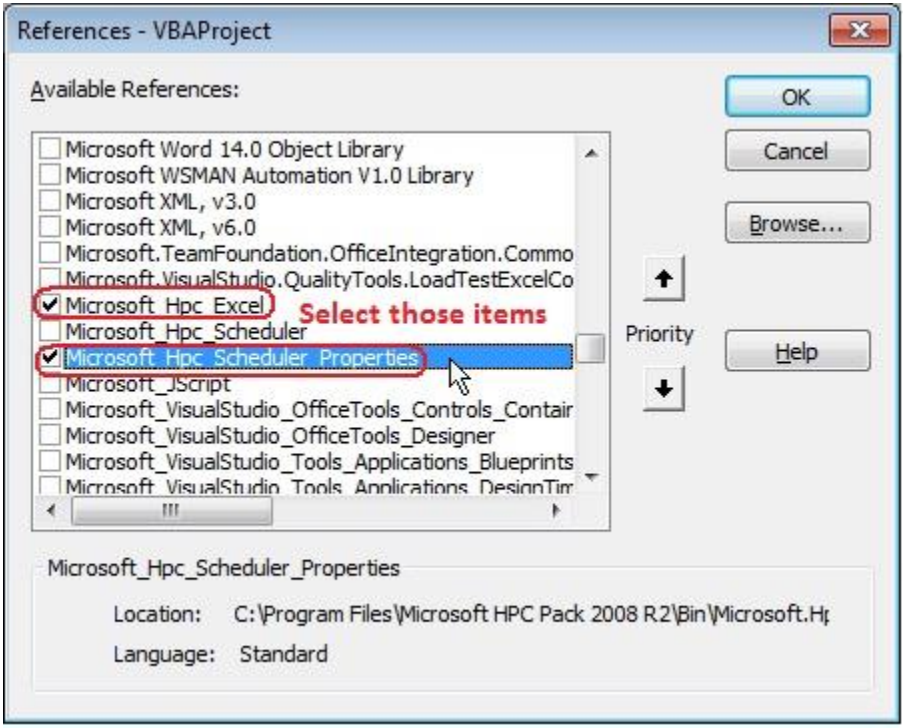



*Figure 7-8 Add references*

8. Click **OK** to close the project references dialog box.

> **Note**
>
> If you don't find the entry in the list, double-check that you're at the end of the list of Microsoft entries. Again, the name has an underscore in it, so even though the list is alphabetical, it will come after some other Microsoft entries. If you still can't find it, verify that you have installed the HPC Pack 2008 R2 client components. If the files are installed, the **Add/Remove programs** or **Programs and Features** item in Control Panel displays an entry named **Microsoft HPC Pack 2008 R2 Client Components**. If you don't see that entry, try reinstalling the client components.

### 7.4.3 Defining the Calculations in the Macro Framework

Our aim in building this workbook is to illustrate how a cluster calculation works, so you don't need to build a very complicated spreadsheet. You'll construct a calculation that runs 100 individual steps. To do that, you'll add a counter to the HPC/Excel macros file, and you'll use that counter in the macros when the calculation runs.

You'll use the counter to keep track of how many calculation steps have been sent, so you can limit the calculation to 100 steps. **HPC_Initialize** is called whenever you start a new calculation, so you can reset the counter in that function. You'll want to increment the counter for each calculation step. You can do that in the **HPC_Partition** macro. Remember that the **HPC_Partition** macro is used to collect any data necessary for an individual calculation step; and after the calculation is complete, the **HPC_Partition** macro should return Null. So to run a calculation for 100 steps, you'll increment the counter every time **HPC_Partition** is called; and after the counter reaches 100, the macro will return Null.

**To construct a calculation that runs 100 steps**

1. In the VBA project window, in the **Modules** folder, double-click **HPCExcelMacros**.
2. At the top of the file, in "Section 1: Variables and constants," add a counter variable named **SentRecords**, as shown in Code 7-2.

```
'==========================================================
'
' Section 1: Variables and constants
'
'==========================================================
Dim SentRecords As Integer
```

*Code 7-2 Add a counter*

3. Scroll down until you find the **HPC_Initialize** function, and add a line to reset the counter variable, as shown in Code 7-3.

```
'----------------------------------------------------------
'
' HPC_Initialize will be called when the client starts
' a calculation.  Put any pre-calculation steps in this
' function.
'
'----------------------------------------------------------
Public Function HPC_Initialize()


    SentRecords = 0


End Function
```

*Code 7-3 Reset counter*

4. Scroll down until you find the macro **HPC_Partition**, and then modify it so that it contains the statements shown in Code 7-4.

```
Public Function HPC_Partition() As Variant
    If SentRecords = 100 Then
        HPC_Partition = Null
    Else
        SentRecords = SentRecords + 1
        HPC_Partition = SentRecords
    End If

End Function
```

*Code 7-4 Use counter to terminate computation*

5. Save the workbook.

> **Note**
>
> Because the new workbook includes macros, you'll need to save it as either a macro-enabled workbook (XLSM file) or a binary workbook (XLSB file). Either one is fine. Usually, we recommend saving workbooks as XLSB files, because they're smaller and a bit more efficient.

Now you have a calculation that will run 100 steps and then be complete. The modifications that you just made are all you have to do to create an Excel cluster calculation.

## 7.4.4 Running the Workbook Locally and Exploring the Macros

Now that you have a calculation, you can add a button to run the workbook, and then run the workbook on the desktop. To see how the macros work, you will add a few new pieces of code to your macro skeleton, and then rerun the workbook to see what the changes did. You will test the workbook locally (on the desktop) before you run the workbook on the cluster.

### 7.4.4.1 Adding a Button That Runs the Workbook Locally

1. With your Excel workbook open, click the **Developer** tab on the ribbon.
2. On the **Developer** tab, click **Insert**, and then select the button control—the first control in the list—as shown in Figure 7-9.

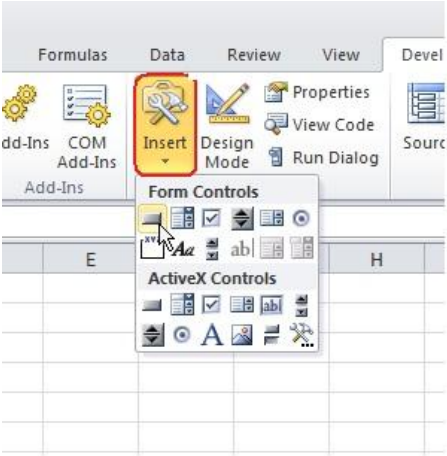



*Figure 7-9 Insert a button*

3. After you've clicked the button, draw a rectangle somewhere on the spreadsheet to insert the button in that location. After you position the button, the **Assign Macro** dialog box appears, as shown in Figure 7-10.

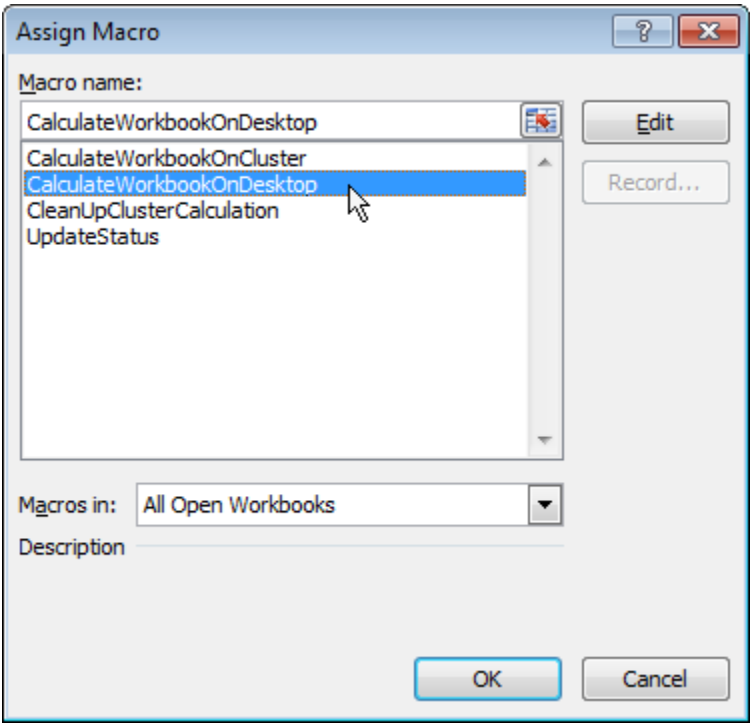



*Figure 7-10 Assign Macro dialog box*

4. In the dialog box, select the macro **CalculateWorkbookOnDesktop** from the list, and then click **OK**. Be sure to select the desktop macro—you'll want to test the workbook first and find any errors before you run it on the cluster.
5. Right-click the new button, and then click **Edit Text** to change the label, as shown in Figure 7-11.

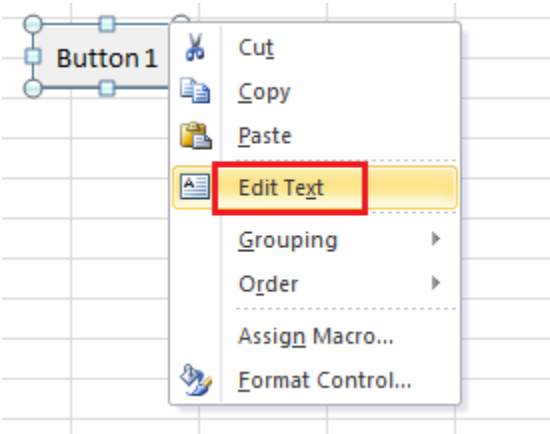



*Figure 7-11 Edit the button text*

6. Type **Desktop** as the name of the label.
7. Optionally, you can verify that you have assigned the correct macro by right-clicking the button and then clicking **Assign Macro**. In the dialog box, verify that **CalculateWorkbookOnDesktop** is selected.
8. Save the workbook.
9. Click the button to run the workbook on the desktop.

If there are any errors, you'll see an error dialog box, and the portion of VBA code with the error will be highlighted. Go back over the preceding sections and check that everything looks correct.

If everything is working, nothing will happen because your workbook doesn't do anything. This is a little unsatisfying, so you can make a few more changes to see the calculation.

#### 7.4.4.2 Modifying the HPC_Execute and the HPC_Merge Macros

Remember that the flow of information goes from **HPC_Partition** (which collects the parameters needed for the calculation), to **HPC_Execute** (which runs the calculation), and then to **HPC_Merge** (which processes the result). When a calculation runs, whatever is returned from the **HPC_Partition** macro is used as the input to the **HPC_Execute** macro. The **HPC_Execute** macro is designed to use that input data, perform a calculation step, and then return a result. The result then is passed to **HPC_Merge**, which can insert the results in the spreadsheet.

In an earlier step, you added code to the **HPC_Partition** macro to update a counter variable. In this example, **HPC_Partition** returns the value of the counter variable and passes that value to **HPC_Execute**. To illustrate the flow of information and have your workbook return some results, you will simply pass this counter value along through the macros. You will add a line of code to the **HPC_Execute** macro so that it takes the input and passes it along as the return value. The value of your counter variable then passes to the **HPC_Merge** macro, which you will modify so that it inserts the value of the counter into your spreadsheet.

**To modify HPC_Execute and HPC_Merge**

1. On the **Developer** tab of the ribbon, click **Visual Basic** to open the VBA editor.
2. In the VBA editor, in the project tree, double-click the **HPCExcelMacros** module to open the macro file.
3. Scroll to the **HPC_Execute** macro, and add a line of code that passes the input value along as the return value, as shown in Code 7-5.

```
Public Function HPC_Execute(data As Variant) As Variant

    HPC_Execute = data

End Function
```

*Code 7-5 Passing values into the HPC_Execute macro*

4. Scroll to the **HPC_Merge** macro, and add a line of code to insert the input value into the spreadsheet. The macro should look like that shown in Code 7-6.

```
Public Function HPC_Merge(data As Variant)

    Cells(data, 1).Value = data

End Function
```

*Code 7-6 HPC_Merge macro*

> **Note**
>
> The "data" value is the counter, which was returned from **HPC_Execute**. So here in the **HPC_Merge** macro, you're filling in a cell in the spreadsheet—by using the counter as the row number—and setting the value to be the counter. The VBA expression "Cells" creates a reference in the spreadsheet with the specified row and column numbers. So every time the **HPC_Merge** macro is called, it will insert a value into column 1 (column A in the spreadsheet), with a different row number.

5. Now go back to the spreadsheet and click your **Desktop** button to run the calculation.

You'll see the first column fill up with numbers from 1 to 100, which are the counter values. Every time you click the button, it will fill in the same numbers, so it's hard to tell that it's running. You can make one more small changes to see the numbers filling in.

#### 7.4.4.3 Modifying the HPC_Initialize Macro

The **HPC_Initialize** macro is called when you first run the workbook. In an earlier step, you added a line of code to reset the counter variable. You can also use this macro to erase the values that were inserted in the spreadsheet during the previous run.

**To modify the HPC_Initialize macro**

1. On the **Developer** tab of the ribbon, click **Visual Basic** to open the VBA editor.
2. In the VBA editor, in the project tree, double-click the **HPCExcelMacros** module to open the macro file.
3. Scroll to the **HPC_Initialize** macro. Add a line of code that empties the first column. The macro should look like that in Code 7-7.

```
Public Function HPC_Initialize()

    Range("A:A").Value = ""
    SentRecords = 0

End Function
```

*Code 7-7 HPC_Initialize macro*

4. Now go back to the spreadsheet and click your **Desktop** button to run the calculation.

It should now be obvious that the macro is filling in the column with numbers every time you click it. If there are any errors, you'll see an error message, and the VBA code with the error will be highlighted. Double-check that it matches the code shown earlier and try again.

### 7.4.5 Running the Workbook on the Cluster

Now you have a very simple Excel calculation that uses the HPC macro framework, and you can run it on the cluster. To do that, you'll first need to set a few values to tell Excel how to contact the cluster. These values are defined in the **HPCControlMacros** module. At the top of this file, there are two values that you'll need to fill in: the cluster scheduler and the share directory. Remember that you (the desktop user) must have write access to this share directory, and the cluster compute nodes must have read access to the directory. In most cases, your cluster calculations will run under your user account, but it's possible to use a different user account when you connect to the cluster session (more about that in a moment).

**To specify the head node and the share directory**

1. On the **Developer** tab of the ribbon, click **Visual Basic** to open the VBA editor.
2. In the VBA editor, in the project tree, double-click the **HPCControlMacros** module to open the macro file.
3. For the cluster scheduler, use the name of your cluster head node—this is the computer name that you would use on the network. You can use a fully qualified name (for example, headnode.contoso.com), but that's not necessary if you're in the same domain.
4. For the share directory, use the full path to the share directory that you created earlier (in the "Before You Start" section).

For example, if, in your cluster, the scheduler name is HN01, and you created a share directory on the head node called HPCTemp, your settings will look like those in Code 7-8.

```
'-----------------------------------------------------------
'
' This is the cluster scheduler, or head node.  Fill in
' the hostname of your cluster scheduler.
'
'-----------------------------------------------------------
Private Const HPC_ClusterScheduler = "HN01"
```

```
'-------------------------------------------------------
'
' This is a network share used to store a temporary copy
' of the workbook.  Make sure that the directory exists,
' that you have write access to the directory, and that
' the compute nodes in the cluster have read access.
'
'-------------------------------------------------------
Private Const HPC_NetworkShare = "\\HN01\HPCTemp"
```

*Code 7-8 Settings*

Now go back to the Excel spreadsheet. You'll add another button, this time to run the calculation on the cluster.

**To add a button that runs the workbook on the cluster**

1. On the **Developer** tab, click **Insert**, and then select the button control—the first control in the list—as shown in Figure 7-12.

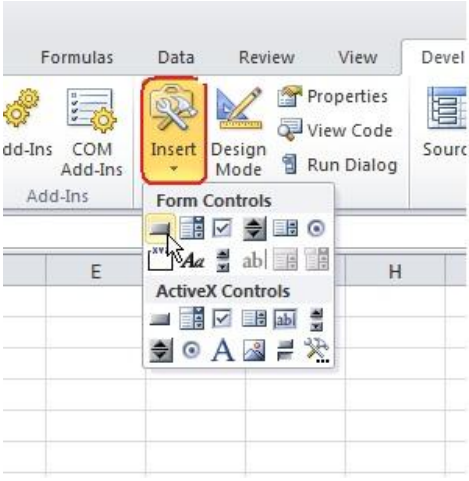



*Figure 7-12 Insert a button*

2. After you've clicked the button, draw a rectangle somewhere on the spreadsheet to insert the button in that location. After you position the button, the **Assign Macro** dialog box appears.
3. In the dialog box, select the macro **CalculateWorkbookOnCluster** from the list, and then click **OK**, as shown in Figure 7-13.

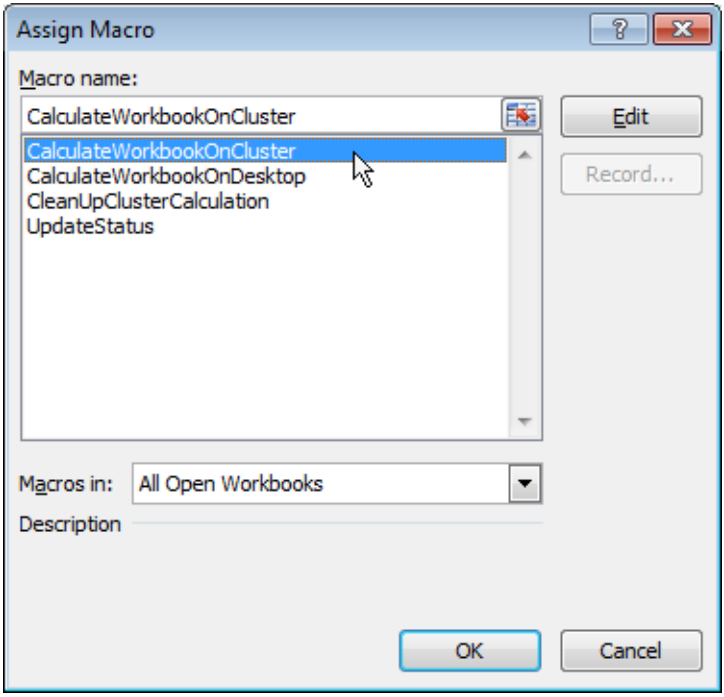



*Figure 7-13 Assign a macro*

4. Right-click the new button, and then click **Edit Text** to change the label, as shown in Figure 7-14.

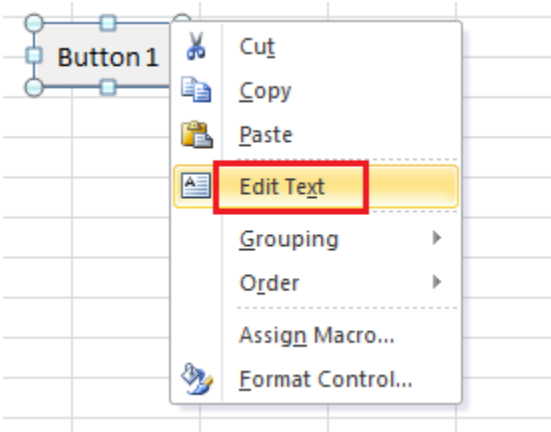



*Figure 7-14 Edit the button text*

5. Type **Cluster** as the name of the label.
6. Save the workbook.
7. Click the button to run the workbook on the cluster.
8. If this is the first time you've run any cluster jobs, either by using Excel or by using any other application, you'll see a user authentication dialog box. Type your user name and password, and (if you want) select the box to save your credentials. If you want to run calculations on the cluster as a different user, you can enter different user account settings.

**Note**

Make sure that your user account—either your standard user account or whatever you want to use—is designated as a cluster user in HPC Cluster Manager.

If there are any errors, you'll see a message box that describes the error. If the calculation worked on the desktop, the most likely errors you'll receive now have to do with the cluster settings—the head node or the share directory. If the message box describes either of those errors, double-check the settings that you changed in the last step. Make sure that you have the correct name for the cluster head node, and that your user account can write to the share directory.

### 7.4.6 Reviewing What Happens When a Workbook Runs on the Cluster

Let's review what happens when you run the cluster calculation. When you click the button, it calls the macro **CalculateWorkbookOnCluster**. That macro uses the client library to start a cluster calculation with the settings that you provided for the cluster head node and share directory.

When the calculation starts, the HPC Services for Excel client code takes over. Remember, you added a reference to the HPC library in the VBA editor at the beginning of this part—that's the client library. First, the client library creates a cluster session, used to manage the calculation. That takes a few seconds after you click the button.

Next, the client library calls the **HPC_Initialize** macro. That happens on the desktop. In this workbook, that macro first resets the internal counter, and then empties column A in the spreadsheet. So when you click the button, first you'll see the delay as the session is created, and then you'll see that column A is cleared. That's the **HPC_Initialize** macro.

Next, the client library starts calling the **HPC_Partition** macro. This also happens on the desktop. In your code, the **HPC_Partition** macro increments the counter and returns the counter value. Every time this macro returns a value—until it returns Null, on the last call—the client library sends a request to the HPC scheduler.

When the scheduler receives one of these requests, it forwards the request to one of the cluster compute nodes for processing. When the compute node receives the request, it starts Excel, loads the workbook, and then calls the **HPC_Execute** macro.

This is the **HPC_Execute** macro that you have in your workbook, but when the calculation runs on the cluster, the **HPC_Execute** macro is actually run on the compute nodes—unlike all the other macros. That's why you're passing data from one macro to another. Information flows from one macro to the next, but as data is passed from the **HPC_Partition** macro to the **HPC_Execute** macro, it is sent from the desktop, through the HPC scheduler, and to one of the compute nodes.

After one of the compute nodes runs the **HPC_Execute** macro, when the macro is completed, the result of the calculation—the value returned from the **HPC_Execute** macro—is sent back to the scheduler, and then returned to the client library on the desktop. When the client library receives the calculation result, it calls the **HPC_Merge** macro on the desktop. So once again, you can think of information flowing from one macro (**HPC_Execute**) to the next (**HPC_Merge**), but this is happening over the network: from the workbook running on the compute node to the workbook running on the desktop.

It's important to understand that these macro function calls—each of the calls to **HPC_Partition**, as you send data to the cluster, and each of the calls to **HPC_Merge**, as data is received—happen asynchronously. That is, when a calculation is sent to the cluster, the client library does not wait (or block) until that particular calculation is complete before it sends the next one. If the **HPC_Partition** macro is very fast, as it is in this case, it's possible that all 100 calculation requests will be sent before any one of those requests is completed.

It's also worth noting that because the results come back from the cluster—and the **HPC_Merge** macro is called in your workbook—they might not come back in the same order that you sent them. That's because as each request is sent to the cluster, the scheduler forwards it to one particular compute node. The compute node calculates the request, by using the **HPC_Execute** macro in the workbook, and then sends back the result. But for a number of reasons, one compute node might run more slowly or more quickly than another compute node; and if that happens, results might be returned in a different order.

You probably won't notice that in this workbook, because the macros are so simple. But it's something that you'll have to prepare for in more complicated workbooks, because you'll have longer calculation functions. You'll address that later when you build a "real" workbook.

#### 7.4.6.1 Communicating Calculation Progress

At this point, you've built a complete workbook that runs on the Windows HPC Server cluster. You should now understand what the various macros do, and why they're designed the way they are. You should have a sense of how data moves between the macros, and what parts are executed on the desktop and on the cluster.

Before you finish up with the simple workbook, you can make a few final changes to add some visual feedback—basically a simple user interface. When you ran the workbook on the cluster, the only feedback you had that it was working was the numbers filling in on the spreadsheet. That's a start, but you can make some changes that will make what the workbook is doing more obvious. What you're going to do is add a few more variables to keep track of the calculation, and then display them in the Excel status bar.

Your code will include the following counters:

- You already have a counter variable that keeps track of the number of times the **HPC_Partition** macro was called. That represents the number of times data was sent from the desktop to the cluster.
- You can add another counter that's incremented every time the **HPC_Merge** macro is called. That represents how many times data has come back from the cluster to the desktop.
- You can also add some variables to keep track of time—how long the calculation takes. This can be really useful to check if the cluster calculation is in fact faster than the desktop (and if so, how much faster).

By using these first two counters, you can see the number of outstanding requests and get a sense of the overall progress of the calculation. You'll add a new VBA function that can report the calculation status. This function has three parts:

- It creates a string that shows the number of calculations sent (the number of times **HPC_Partition** was called) and the number of results received back (the number of times **HPC_Merge** was called).
- If the **CalculationComplete** variable is true, it updates the string to show the total calculation time.
- It uses **Application.StatusBar** to set the Excel status bar message with the information we've just created.

You can use the HPC macros that are already in place to update these values and update the status bar message with the new function.

The following procedure describes how to incorporate these new elements into your sample.

**To add progress variables and update the Excel status bar**

1. Go back to the VBA editor, and double-click the module **HPCExcelMacros** to open the macro code. At the top of the VBA code, after the original counter, add new variables as shown in Code 7-9.

```
'==========================================================
'
' Section 1: Variables and constants
'
'==========================================================

Dim SentRecords As Integer
Dim RcvRecords As Integer
Dim CalculationComplete As Boolean
Dim StartTime As Double
Dim FinishTime As Double
```

*Code 7-9 New status variables*

2. Add a new VBA function called **UpdateStatus** that can report the calculation status. Scroll down to the bottom of the file, and add code as shown in Code 7-10.

```
Sub UpdateStatus()

    Dim statusMessage As String
    statusMessage = "Calculated " & RcvRecords & "/" & SentRecords

    If CalculationComplete Then
        statusMessage = statusMessage & "; Completed in " & _
           FormatNumber(FinishTime - StartTime) & "s"
    End If

    Application.StatusBar = statusMessage

End Sub
```

*Code 7-10 Update function*

3. You need to initialize the variables. As with the original counter, you're going to do that in the **HPC_Initialize** macro. Scroll to that macro, and add a few lines as shown in Code 7-11.

```
Public Function HPC_Initialize()


    Range("A:A").Value = ""
    SentRecords = 0
    RcvRecords = 0
    StartTime = Timer
    CalculationComplete = False
    UpdateStatus


End Function
```

*Code 7-11 Initializing the status variables*

> **Note**
>
> What you're doing here is (1) clearing the two counters, setting them to zero; (2) setting the StartTime value to the current time (the Timer call in VBA returns the current time, in seconds); and (3) setting the CalculationComplete flag to false, meaning that it hasn't finished yet. After these values are set, you call the new UpdateStatus macro to write the message to the status bar.

4. Scroll down to the **HPC_Partition** macro. In this macro, you're already setting the first counter, so you need to make only one change: After updating the counter, you add a line to call the new macro and update the status bar, as shown in Code 7-12.

```
Public Function HPC_Partition() As Variant


    If SentRecords = 100 Then
        HPC_Partition = Null
    Else
        SentRecords = SentRecords + 1
        UpdateStatus
        HPC_Partition = SentRecords
    End If


End Function
```

*Code 7-12 Calling the status update function*

5. Scroll down to the **HPC_Merge** macro. Here, you'll want to update the new counter—the one that represents messages returned back to the desktop—and again update the status bar. So modify the **HPC_Merge** macro as shown in Code 7-13.

```
Public Function HPC_Merge(data As Variant)

    Cells(data, 1).Value = data
    RcvRecords = RcvRecords + 1
    UpdateStatus

End Function
```

*Code 7-13 Calling the status update function*

6. Scroll down to the **HPC_Finalize** macro. Modify the **HPC_Finalize** macro as shown in Code 7-14.

```
Public Function HPC_Finalize()

    CalculationComplete = True
    FinishTime = Timer
    UpdateStatus

    ' Clean up the calculation.  It's a good idea to
    ' leave this here, even if you make changes to
    ' this function.  The function we call here is in
    ' the "HPCControlMacros" module.

    CleanUpClusterCalculation

End Function
```

*Code 7-14 Calling the status update function*

> **Note**
>
> When you set the **FinishTime** field to Timer, you're setting it with the current time, in seconds. In the **UpdateStatus** macro, you're using **StartTime** and **FinishTime** to figure out the total calculation time.

7. Save the workbook.
8. Go back to the Excel spreadsheet and click the **Cluster** button.

You'll see the status bar update with numbers as the calculation runs. First, you'll see the number of calculations sent (the second number) jump up to 100. Then, after a brief delay, you'll see the number of results received (the first number) climb up to 100, as shown in Figure 7-15.

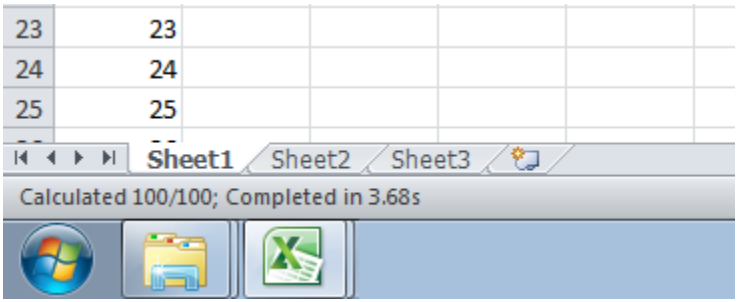



*Figure 7-15 Progress information shown in the Excel status bar*

What you're seeing here is the asynchronous messaging we discussed earlier. Because the **HPC_Partition** macro is so fast, the client library can send all 100 requests before any of the calculations are complete. There's a brief delay—that delay is the compute nodes that are starting Excel and loading the workbook. After each compute node has started Excel, calculating the results is pretty fast, and the first number gets up to 100 quickly.

If you click the **Desktop** button, you'll see that the numbers move in a different way. On the desktop, the client library is calling the same macros, but it is doing it synchronously (that is, not asynchronously). To run the desktop calculation, the client library calls each macro in turn—first **HPC_Partition**, then **HPC_Execute**, and then **HPC_Merge**—but in this case, it waits for each macro to finish before calling the next one. That's why when you run the calculation on the desktop, you'll see both numbers moving together.

## 7.5 Part II: Converting a Desktop Workbook to Run on the Windows HPC Server Cluster

In the last section, you constructed a workbook from scratch to calculate on a Windows HPC Server cluster. In this section, you'll take an existing workbook and modify it to run on the cluster. If you followed the last part, you should have an understanding of what's required—although there were a lot of individual steps, the actual work involved was very simple. In summary, you:

- Added two macro files.
- Added two references to the VBA project.
- Made a few modifications to the macros.
- Added buttons to run the spreadsheet.

Converting an existing workbook is about the same amount of work. There's not a lot of new VBA code to add, and most if it can reuse the same skeleton macro files. For the most part, converting an existing workbook involves moving code around—copying code from existing macros into the HPC macros.

The difficult part of converting an existing workbook is figuring out what code to move, and where it should go. Excel VBA is a comprehensive development environment, and there are many ways to build Excel applications. Different developers might write the same application in many different ways.

In this section, you're going to take an existing workbook and convert it to run on the cluster. You should read this section as a guide, or as an example, and then try to apply the concepts to your own workbooks. You have a fairly simple example in this case (although it's based on a real workbook that was used to solve a real business problem). For just about any workbook, the process will be the same, but the specific details will be different.

### 7.5.1 Analyzing the Workbook

In Excel, open the workbook ConvertiblePricing.xlsb from the Second Workbook directory. You might get a warning about macros; if so, click **Enable Macros** to continue.

This workbook calculates convertible bond prices, based on the values in the "Model Parameters" section on the left. The table on the right is used to understand what effect small changes in the parameters—changes in the conversion price and in the coupon rate—will have on the final price.

Click the **Calculate Table** button in the spreadsheet. You'll see the table fill in, slowly, as each price is calculated. On a typical workstation, filling in the complete table will take one to two minutes.

To figure out what this workbook does when you click the button, you can walk through the VBA code.

**To identify the macro and see the code**

1. Right-click the **Calculate Table** button, and then click **Assign Macro**, as shown in Figure 7-16.

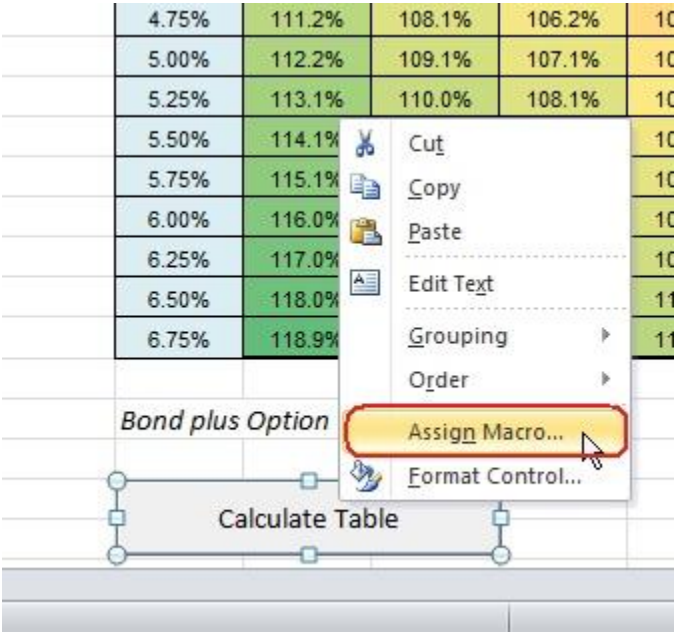



*Figure 7-16 Clicking Assign Macro on the shortcut menu*

2. The **Assign Macro** dialog box shows which macro the button currently uses, as shown in Figure 7-17. In the dialog box, click **Edit** to jump into the code. You should see the code for the **CalculateTable** macro. This is the code that's executed when you click the button in the spreadsheet.

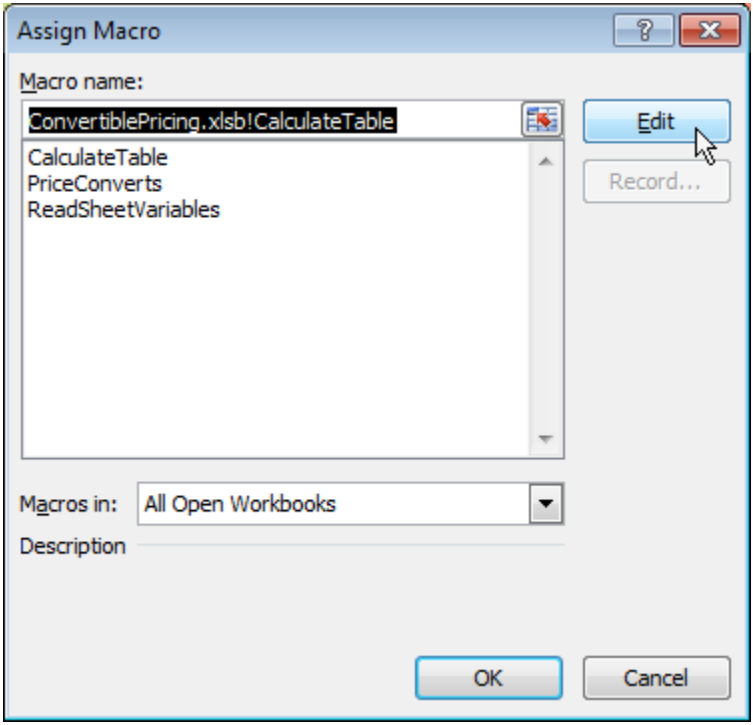



*Figure 7-17 Assign Macro dialog box*

Looking at this macro, you can see a few distinct sections. First, there are some variable declarations. Next, there's some initialization code. Finally, there's a section that calculates the table—you can identify where it loops over the rows and columns in the table and fills in values.

This macro is fairly short, but if you look at the loop, you'll see that it calls some other functions—the functions **FullDiscountModel** and **BondPlusOptionModel**. These are the actual calculation routines, the business logic that power this workbook.

In the VBA editor, you can jump to the source of any function by right-clicking the name and then clicking **Definition**. If you right-click **FullDiscountModel** and then click **Definition**, you'll see that the actual calculation is rather complex. But to convert the workbook to run on the cluster, you don't have to modify or even understand that calculation. You only need to worry about the control code—that is, the function that contains the loop.

This workbook is very cleanly designed, in that the business logic is contained in separate functions. In other cases, all this code might be in a single macro, which would make the workbook harder to understand. But in either case, what you have to do to convert this workbook is take the control code—the startup routines and the loop functions—and move them to the HPC macros.

You also know that in this workbook, the calculations are independent. That is, each cell of the table is calculated without reference to any other cell in the table. That means the workbook is iterative, and it will support parallelization (as discussed earlier in the "Overview" section). It might not always be obvious that calculations are independent. One way to figure this out is to write some test functions—for example, you can rewrite the **CalculateTable** macro (the macro that's called when you click the button in the spreadsheet)—to run only a single calculation, or one cell in the table. In that way, you can identify whether the results were consistent and, if so, you can make a determination whether the calculations were in fact independent.

To summarize, the features that are most important for any workbook to support execution on a Windows HPC Server cluster are as follows:

- The workbook is iterative: It runs the same calculation many times, with different sets of input data; or it runs many individual calculations, such as rows or cells in a table, one at a time.
- The individual calculations are independent: That is, the results of a single calculation don't affect any of the other calculations.

Both of these are true for this workbook, so it's a good candidate for running on the cluster.

### 7.5.2 Preparing the Workbook

As in the last workbook, the first changes you'll make have to do with setting up the VBA environment. You can use skeleton macro files to save some time—these include the HPC macros that you'll need for the calculation.

In the last workbook, you made some changes to the skeleton macro files to add some user feedback (displaying the calculation progress in the Excel status bar). In this second workbook, the skeleton macro files already include that status bar code.

**To import the macro files and add references**

1. On the **Developer** tab of the ribbon, click **Visual Basic** to open the VBA editor.
2. In the VBA editor, right-click **VBA Project** at the top of the tree view, and then click **Import File**.
3. In the dialog box, locate the project files that you downloaded. Go into the directory Second Workbook, click the file **HPCExcelMacros.bas**, and then click **OK**, as shown in Figure 7-18.

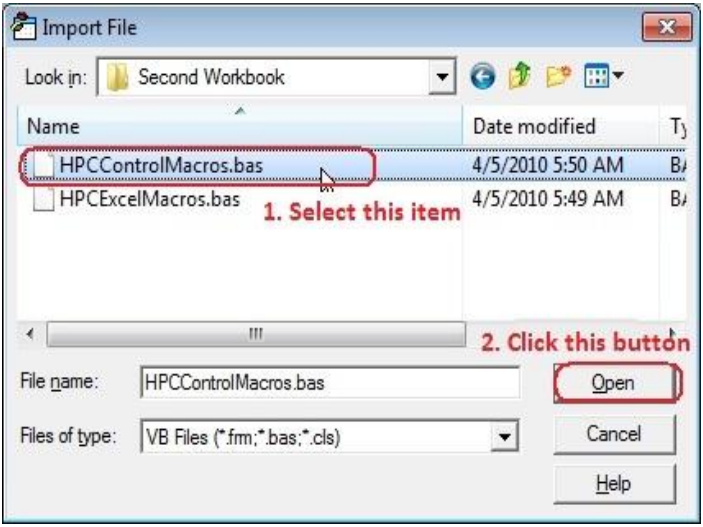



*Figure 7-18 Opening the second workbook*

4. Repeat steps 1–3 to import HPCControlMacros.bas, and then click **OK**. That adds the second macro file to your project.
5. Verify that these two macro files appear in the VBA project window, as shown in Figure 7-19.

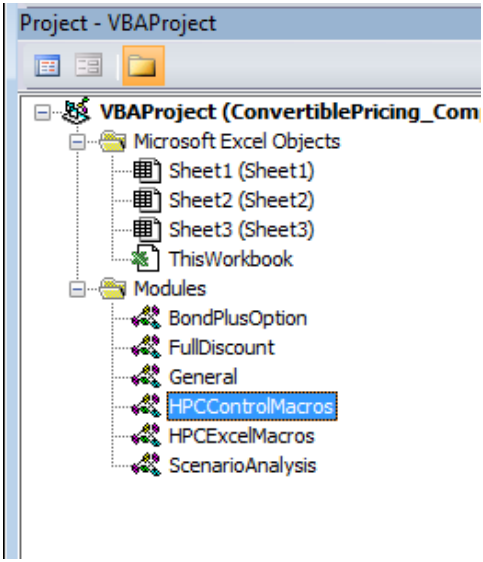



*Figure 7-19 VBA project for the second workbook*

6. In the VBA editor window, click **Tools**, and then click **References** to open the project references dialog box.
7. In the list of available references, scroll down until you find **Microsoft_Hpc_Excel** and select the check box next to it. When you've selected the box, the dialog box should look like the one shown in Figure 7-20.

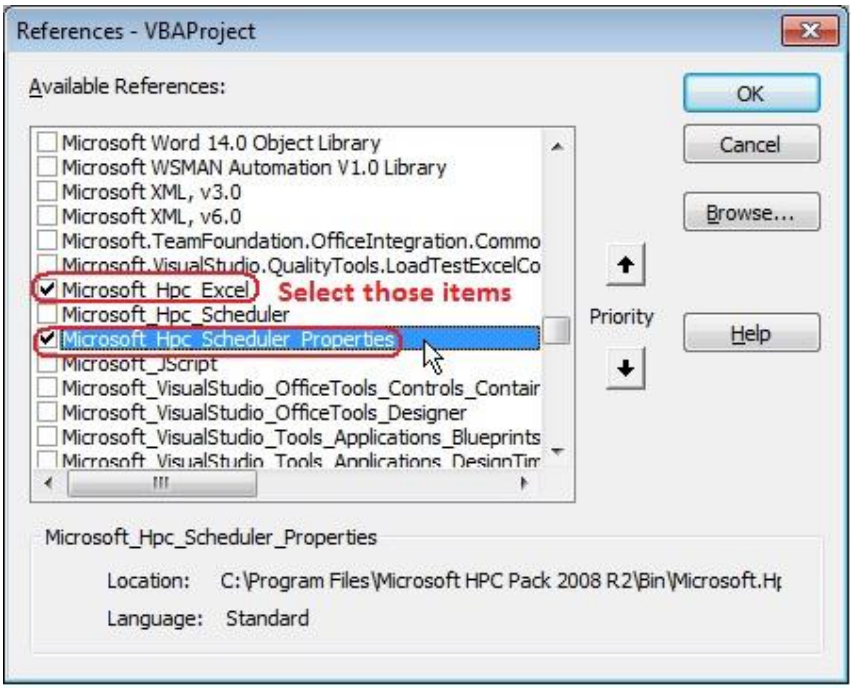



*Figure 7-20 Adding HPC SDK references*

8. Click **OK** to close the project references dialog box.

## 7.5.3 Defining the Calculations in the Macro Framework

With the VBA environment set up, you can move to the next step—modifying the macros to run the calculation. The **HPCExcelMacros** module includes the same macros as in the first workbook, only this time they already include the counters and the code for updating the status bar.

### 7.5.3.1 Defining Variables and Copying Initialization Code From the Original Macro

The first things you'll need are counters for the row and column. In this workbook, you're dividing the calculation so that each cell in the table is calculated separately. The original macro used loops to run through each cell in the table. In the HPC macro functions, you're using the asynchronous framework, so you need to keep track of the row and column numbers as you send out each individual calculation. You'll use row and column variables in the macros to step through each cell in the table. They reflect the row and column you're calculating at each step.

You'll go back to the original macro to see what it did when it started, and then move that to the **HPC_Initialize** macro.

**To set variables and modify HPC_Initialize**

1. On the **Developer** tab of the ribbon, click **Visual Basic** to open the VBA editor.
2. In the VBA editor, in the project tree, double-click the **HPCExcelMacros** module to open the macro file.
3. At the top of the file, in "Section 1: Variables and constants," add row and column variables as shown in Code 7-15.

```
'==========================================================
'
' Section 1: Variables and constants
'
'==========================================================

Dim CurrentRow As Integer
Dim CurrentCol As Integer

Dim SentRecords As Integer
Dim RcvRecords As Integer
Dim CalculationComplete As Boolean
Dim StartTime As Double
Dim FinishTime As Double
```

*Code 7-15 Declaring variables*

4. Scroll to the **HPC_Initialize** macro and initialize the row and column counters that you just set up, as shown in Code 7-16.

```
Public Function HPC_Initialize()

    CurrentRow = 1
    CurrentCol = 1

    ' clear counters, capture starting time and update status bar

    SentRecords = 0
    RcvRecords = 0
    StartTime = Timer
    CalculationComplete = False
    UpdateStatus

End Function
```

*Code 7-16 Initializing variables*

5. In the VBA editor, double-click the **ScenarioAnalysis** module in the VBA project window. That was the file that contained the original macro code. The initialization part is the first section, which determines the size of the table and sets variables (**NumRows** and **NumCols**), as shown in Code 7-17.

```
    ' clear values, then figure out the size of the table

    Range("cashtable").ClearContents

    NumCols = 1
    While NumCols < MaxSensTblSize And CDbl(Range("table1").Offset(0,
   NumCols …
        NumCols = NumCols + 1
    Wend
    NumRows = 1
    While NumRows < MaxSensTblSize And _
       CDbl(Range("table1").Offset(NumRows + 1, 0).Value) <> 0 And _
       CDbl(Range("table1").Offset(NumRows + 1, 0).Value) <> …
        NumRows = NumRows + 1
    Wend
```

*Code 7-17 ScenarioAnalysis module*

6. Copy the code from the original macro, and then go back to the **HPCExcelMacros** module (double-click **HPCExcelMacros** in the VBA project window) and paste it into the **HPC_Initialize** macro.

   The final code in **HPC_Initialize** now looks like that shown in Code 7-18.

```
Public Function HPC_Initialize()

    ' clear values, then figure out the size of the table

    Range("cashtable").ClearContents

    NumCols = 1
    While NumCols < MaxSensTblSize And CDbl(Range("table1").Offset(0,
   NumCols + …
        NumCols = NumCols + 1
    Wend
    NumRows = 1
    While NumRows < MaxSensTblSize And _
       CDbl(Range("table1").Offset(NumRows + 1, 0).Value) <> 0 And _
       CDbl(Range("table1").Offset(NumRows + 1, 0).Value) <> …
        NumRows = NumRows + 1
    Wend

    CurrentRow = 1
    CurrentCol = 1

    ' clear counters, capture starting time and update status bar

    SentRecords = 0
    RcvRecords = 0
    StartTime = Timer
    CalculationComplete = False
    UpdateStatus

End Function
```

*Code 7-18 Final code of HPC_Initialize*

### 7.5.3.2 Modifying the HPC_Partition Macro

The next macro to modify is the **HPC_Partition** macro. Remember that this macro collects whatever data is necessary to run a single calculation. In this case, that's going to be the row and column number. For this workbook, you'll want to calculate each individual table cell on the cluster separately. To do that, you'll use the row and column number.

All the macros in the HPC framework use the **Variant** data type for inputs and outputs. The **Variant** type can be any primitive type (integer, long, double, string) or an array of these types. You need to include both the row and column, so you can use an array.

The other thing you need to do in the **HPC_Partition** macro is to figure out when the calculation is complete. In this case, the calculation will be completed when you've reached the end of the table. You can do that by incrementing the row and column numbers as you move through the table. Whenever you reach the end of a column, you move to the next column. When you've passed the last column, the table is finished and you'll want the function to return Null.

**To modify the HPC_Partition macro**

1. In the **HPCExcelMacros** code, scroll to the **HPC_Partition** section and add code to declare an array for the row and column, as shown in Code 7-19.

```
Public Function HPC_Partition() As Variant

    Dim data(3) As Variant

    ' update the status bar with the counters

    SentRecords = SentRecords + 1
    UpdateStatus

End Function
```

*Code 7-19 Declaring arrays used for passing data among the macros*

2. Add code to check for the end of the table, as shown in Code 7-20. This will end the calculation when you've finished the table.

```
Public Function HPC_Partition() As Variant

    Dim data(3) As Variant

    ' first check the row; if we're past the bottom
    ' of the table, increment the column and set the
    ' row back to the top

    If CurrentRow > NumRows Then
        CurrentRow = 1
```

```
        CurrentCol = CurrentCol + 1
    End If

    ' next check the column; if we're past the last
    ' column, then we're done

    If CurrentCol > NumCols Then

        ' return null to indicate the end of the calculation
        HPC_Partition = Null

        ' and exit the function now, we can skip the rest
        Exit Function
    End If

    ' update the status bar with the counters

    SentRecords = SentRecords + 1
    UpdateStatus

End Function
```

*Code 7-20 Adding termination determination logic*

3. If you haven't yet finished the table, the last part of the function will store the row and column values and use them as the return value for the function, and then move to the next row, as shown in Code 7-21.

```
Public Function HPC_Partition() As Variant

    Dim data(3) As Variant

    ' first check the row; if we're past the bottom
    ' of the table, increment the column and set the
    ' row back to the top

    If CurrentRow > NumRows Then
        CurrentRow = 1
        CurrentCol = CurrentCol + 1
    End If
```

```
    ' next check the column; if we're past the last
    ' column, then we're done

    If CurrentCol > NumCols Then

        ' return null to indicate the end of the calculation
        HPC_Partition = Null

        ' and exit the function now, we can skip the rest
        Exit Function
    End If

    data(0) = CurrentRow
    data(1) = CurrentCol
    HPC_Partition = data

    ' move to the next row

    CurrentRow = CurrentRow + 1

    ' update the status bar with the counters

    SentRecords = SentRecords + 1
    UpdateStatus

End Function
```

*Code 7-21 Adding column traversal code*

To review, the **HPC_Partition** macro has two purposes: It collects parameters necessary for a single calculation (here that's the row and column number); and when the calculation is complete, it signals that by returning Null. In the function you've created, it does that by stepping through the table, first by row and then by column. When it has passed the last column, it returns Null to indicate that the table is complete.

#### 7.5.3.3 Modifying the HPC_Execute Macro

The next macro is **HPC_Execute**. Remember that this macro is designed to run the actual calculation, or the business logic. The inputs to this function will be whatever you returned from the **HPC_Partition** macro in the last step. Because you just wrote the **HPC_Partition** macro, you know that the inputs to the **HPC_Execute** macro will be the row and column number of the table cell that you want to calculate.

The body of the **HPC_Execute** macro is going to be the code copied from the original calculation macro. You'll need two parts of this code: the variable declarations at the top, and the code that runs the loop. You already used the section that deals with the size of the table in the **HPC_Initialize** macro, so you don't need that again, but it will be easier to copy the whole code block and then remove the parts you don't need. Then, you'll make a few changes to support the asynchronous framework.

**To modify the HPC_Execute macro**

1. Go back to the original macro by double-clicking the **ScenarioAnalysis** module in the VBA project window.
2. Copy the code from the first line:

   ```
   Dim Prem as double
   ```

   Down to the last line:

   ```
   Next cols
   ```

3. Go back to the HPCExcelMacros module (by double-clicking in the VBA project window) and paste the code into the **HPC_Execute** macro.
4. In the **HPC_Execute** macro, delete the lines that you've already used, starting from:

   ```
   ' clear values, then figure out the table size
   ```

   Down to the line just before :

   ```
   ' set up and run through the table
   ```

5. Verify that the **HPC_Execute** macro looks like the one shown in Code 7-22.

```
Public Function HPC_Execute(data As Variant) As Variant

    Dim Prem As Double
    Dim TempPIKdate As Date
    Dim TempPIKfreq As Double
    Dim rws As Integer
    Dim cols As Integer
    Dim Model As Integer

    ' set up and run through the table

    Call ReadSheetVariables
    TempPIKdate = PIKdate 'Save this value!
    TempPIKfreq = PIKfreq 'Save this value!

    Model = 1
    If Range("PricingModel").Value2 = "Bond plus Option" Then
        Model = 2
    End If
```

```
    For cols = 1 To NumCols
        For rws = 1 To NumRows

            Call ReadSheetVariables
            PIKdate = TempPIKdate
            PIKfreq = TempPIKfreq
            PIKrate = CDbl(Range("table1").Offset(rws, 0).Value)
            Prem = CDbl(Range("table1").Offset(0, cols).Value)
            If Prem = 0 Then End
            Ratio = CDbl(Face / Underlying / (1 + Prem))

            Select Case Model
            Case 1
                Range("cashtable").Cells(rws, cols).Value =
   FullDiscountModel(360) / Face
            Case Else
                Range("cashtable").Cells(rws, cols).Value =
   BondPlusOptionModel(360) / Face            End Select

        Next rws
    Next cols

End Function
```

*Code 7-22 HPC_Execute macro*

> **Note**
>
> In the macro, you now have the calculation part of the function, including the loop that runs through each cell of the table. For the asynchronous framework, you don't want to use a loop here; you just want to calculate a single cell. Remember that the **HPC_Execute** macro is code that will run on the compute nodes, and you're giving it the row and column number that you want to calculate.

6. Remove the lines that make up the loop (the highlighted lines shown in Code 7-23).

```
Public Function HPC_Execute(data As Variant) As Variant

    Dim Prem As Double
    Dim TempPIKdate As Date
    Dim TempPIKfreq As Double
```

```
    Dim rws As Integer
    Dim cols As Integer
    Dim Model As Integer

    ' set up and run through the table

    Call ReadSheetVariables
    TempPIKdate = PIKdate 'Save this value!
    TempPIKfreq = PIKfreq 'Save this value!

    Model = 1
    If Range("PricingModel").Value2 = "Bond plus Option" Then
        Model = 2
    End If

    For cols = 1 To NumCols
        For rws = 1 To NumRows

            Call ReadSheetVariables
            PIKdate = TempPIKdate
            PIKfreq = TempPIKfreq
            PIKrate = CDbl(Range("table1").Offset(rws, 0).Value)
            Prem = CDbl(Range("table1").Offset(0, cols).Value)
            If Prem = 0 Then End
            Ratio = CDbl(Face / Underlying / (1 + Prem))

            Select Case Model
            Case 1
                Range("cashtable").Cells(rws, cols).Value =
   FullDiscountModel(360) / Face
            Case Else
                Range("cashtable").Cells(rws, cols).Value =
   BondPlusOptionModel(360) / Face
            End Select

        Next rws
    Next cols

End Function
```

*Code 7-23 Deleting loop constructs*

7. Replace the loop with the row and column number that are passed in to the function. The modified code should look like that shown in Code 7-24.

```
    rws = data(0)
    cols = data(1)

    Call ReadSheetVariables
    PIKdate = TempPIKdate
    PIKfreq = TempPIKfreq
    PIKrate = CDbl(Range("table1").Offset(rws, 0).Value)
    Prem = CDbl(Range("table1").Offset(0, cols).Value)
    If Prem = 0 Then End
    Ratio = CDbl(Face / Underlying / (1 + Prem))

    Select Case Model
    Case 1
        Range("cashtable").Cells(rws, cols).Value = FullDiscountModel(360)
   / Face
    Case Else
        Range("cashtable").Cells(rws, cols).Value =
   BondPlusOptionModel(360) / Face
    End Select
```

*Code 7-24 Input parameters*

Instead of the loop, now you're using the values from your data array—the array that you returned from the **HPC_Partition** macro—which contains the row and column of the cell that you want to calculate. To review that last step, you removed the two loop lines:

```
For cols = 1 To NumCols
    For rws = 1 To NumRows
```

As well as the **Next** lines at the end of the loop:

```
    Next rws
Next cols
```

And added lines just above the loop to use the data array variable:

```
rws = data(0)
cols = data(1)
```

The last change you'll want to make to the **HPC_Execute** function has to do with returning the data. Remember that data moves through the macros; each macro passes results to the next one. The original VBA code for this workbook was designed to run on the desktop, so as each cell was calculated, the result was pasted into the spreadsheet. Because this workbook is going to run on the cluster, instead of pasting results into the spreadsheet, you need to collect the calculation result and return it from the function.

And there's one other thing you need to consider: When the **HPC_Execute** function runs on a compute node, it calculates the result for a particular cell in the table. This result is then going to be returned to the desktop and sent to the **HPC_Merge** macro. But remember that in the asynchronous framework, results can be sent back in any order—not necessarily the same order that you sent them. So you need to tell the **HPC_Merge** macro which cell you've calculated.

You can do that by including the row and column number in the results data. As with the **HPC_Partition** macro, you can return an array from the **HPC_Excecute** macro. Doing that will allow the function to return not only the result, but also which cell was calculated.

In fact, you can simply reuse the same array that was passed in to the function to return the result, because that array already contains the row and column number.

To store the result, change the code so that it reads as shown in Code 7-25.

```
    Select Case Model
    Case 1
        data(2) = FullDiscountModel(360) / Face
    Case Else
        data(2) = BondPlusOptionModel(360) / Face
    End Select

    HPC_Execute = data
```

*Code 7-25 Result and the cell ID being the return result*

You've changed the code that inserted values into the spreadsheet to instead store the result values in the data array. And the last line uses the data array as the return value from the function, so the results—and the row and column numbers—will be sent to the next macro.

So the complete, final code for the **HPC_Execute** macro should look like that shown in Code 7-26.

```
Public Function HPC_Execute(data As Variant) As Variant

    Dim Prem As Double
    Dim TempPIKdate As Date
    Dim TempPIKfreq As Double
    Dim rws As Integer
    Dim cols As Integer
    Dim Model As Integer
```

```
    ' set up and run through the table

    Call ReadSheetVariables
    TempPIKdate = PIKdate 'Save this value!
    TempPIKfreq = PIKfreq 'Save this value!

    Model = 1
    If Range("PricingModel").Value2 = "Bond plus Option" Then
        Model = 2
    End If

    rws = data(0)
    cols = data(1)

    Call ReadSheetVariables
    PIKdate = TempPIKdate
    PIKfreq = TempPIKfreq
    PIKrate = CDbl(Range("table1").Offset(rws, 0).Value)
    Prem = CDbl(Range("table1").Offset(0, cols).Value)
    If Prem = 0 Then End
    Ratio = CDbl(Face / Underlying / (1 + Prem))

    Select Case Model
    Case 1
        data(2) = FullDiscountModel(360) / Face
    Case Else
        data(2) = BondPlusOptionModel(360) / Face
    End Select

    HPC_Execute = data

End Function
```

*Code 7-26 HPC_Execute complete code*

#### 7.5.3.4 Modifying the HPC_Merge Macro

The last macro that you need to modify is the **HPC_Merge** macro. Remember that this macro receives a single calculation result from the cluster. You'll want to use it to insert the result into the table.

The input to **HPC_Merge** is going to be whatever you return from **HPC_Execute**. Because you just wrote the **HPC_Execute** macro, you know that the return value will be an array that contains the row and column number and the result of the calculation. To update the table, you can use code from the original macro (this is code that you just changed in the **HPC_Execute** function).

The original line of code, in the **ScenarioAnalysis** module, looked like that shown in Code 7-27.

```
        Range("cashtable").Cells(rws, cols).Value = FullDiscountModel(360) /
   Face
```

*Code 7-27 Original code: insert result to the workbook*

You'll rewrite that line to use the values passed in to the macro in the data parameter. The completed **HPC_Merge** macro should look like the one shown in Code 7-28.

```
Public Function HPC_Merge(data As Variant)

    Range("cashtable").Cells(data(0), data(1)).Value = data(2)

    ' update the status bar with the counters

    RcvRecords = RcvRecords + 1
    UpdateStatus

End Function
```

*Code 7-28 Updated code: insert result to the workbook*

This is the same line of code that was just removed from the **HPC_Execute** macro. You've replaced the row and column values (`rws` and `cols` in the original line) with the row and column from the array. You've also replaced the function call with the result, which is stored in the same array.

Those are all the modifications that you have to make to run this workbook on the cluster. Although it might seem like a lot of steps, particularly if this is the first time you've used HPC Services for Excel, look back at the macros to see the total code that was changed. It's really not a whole lot. For the most part, this involved copying and pasting from the original macro into the HPC macros. To support asynchronous calculation, you made a number of changes to the code—but even that was relatively simple. You made no changes to the actual business logic in this workbook, and you don't even have to understand the business logic functions in order to convert the workbook. More importantly, if users make changes to the business logic in the future, it won't require changing the cluster calculation.

### 7.5.4 Running the Workbook

To test changes to the code, you'll first run it on the desktop. As in the first workbook, you'll add a new button to the spreadsheet and use that to run the calculation. Then, you'll specify the head node and the file share, add a button that runs the workbook on the cluster, and then run the workbook on the cluster.

### 7.5.4.1 Adding a Button That Runs the Workbook Locally

1. With your Excel workbook open, click the **Developer** tab on the ribbon.
2. On the **Developer** tab, click **Insert**, and then select the button control—the first control in the list—as shown in Figure 7-21.

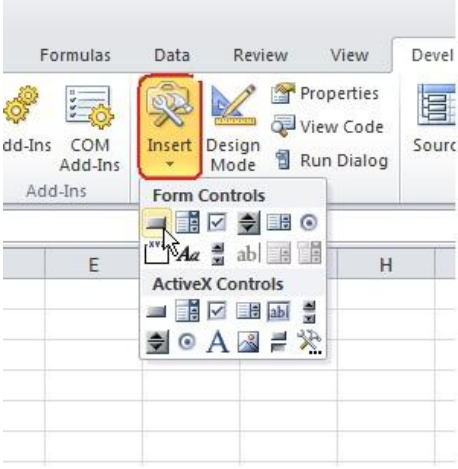



*Figure 7-21 Adding a button*

3. After you've clicked the button, draw a rectangle somewhere on the spreadsheet to insert the button in that location. After you position the button, the **Assign Macro** dialog box appears.
4. In the dialog box, select the macro **CalculateWorkbookOnDesktop** from the list, and then click **OK**, as shown in Figure 7-22. Be sure to select the desktop macro—you'll want to test the workbook first and find any errors before you run it on the cluster.

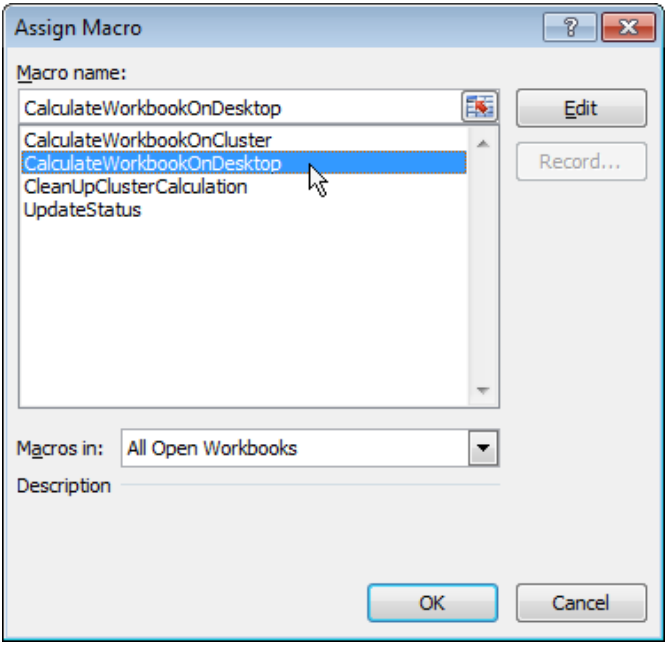



*Figure 7-22 Assign macro*

5. Right-click the new button, and then click **Edit Text** to change the label.
6. Type **Desktop** as the name of the label.
7. Save the workbook.
8. Click the button to run the workbook on the desktop.

If there are any errors, you'll see an error dialog box, and the portion of VBA code with the error will be highlighted. Double-check against the code shown earlier to resolve any errors.

If there are no errors, you'll see the table being filled in. The values should be the same as when you clicked the original button on the spreadsheet. Again, this is running on the desktop, so it should run at about the same speed as the original button—one to two minutes on most workstations.

Because you're running with the macro framework code, you'll also see the calculation progress displayed in the Excel status bar. As with the first workbook running on the desktop, you'll see the numbers move together, because on the desktop, the workbook runs synchronously.

#### 7.5.4.2 Running the Workbook on the Cluster

Next, you'll try running the workbook on the cluster. To do that, you'll first need to set a few values to tell Excel how to contact the cluster. These values are defined in the **HPCControlMacros** module. At the top of this file, there are two values that you'll need to fill in: the cluster scheduler and the share directory. Remember that you (the desktop user) must have write access to this share directory, and the cluster compute nodes must have read access to the directory. In most cases, your cluster calculations will run under your user account, but it's possible to use a different user account when you connect to the cluster session (more about that in a moment).

**To specify the head node and the share directory**

1. On the **Developer** tab of the ribbon, click **Visual Basic** to open the VBA editor.
2. In the VBA editor, in the project tree, double-click the **HPCControlMacros** module to open the macro file.
3. For the cluster scheduler, use the name of your cluster head node—this is the computer name that you would use on the network. You can use a fully qualified name (for example, headnode.contoso.com), but that's not necessary if you're in the same domain.
4. For the share directory, use the full path to the share directory that you created earlier (in the "Before You Start" section).

**To add a button that runs the workbook on the cluster**

1. On the **Developer** tab, click **Insert**, and then select the button control—the first control in the list.
2. After you've clicked the button, draw a rectangle somewhere on the spreadsheet to insert the button in that location. After you position the button, the **Assign Macro** dialog box appears.
3. In the dialog box, select the macro **CalculateWorkbookOnCluster** from the list, and then click **OK**.
4. Right-click the new button, and then click **Edit Text** to change the label.
5. Name the label **Cluster** or something similar.
6. Save the workbook.
7. Click the button to run the workbook on the cluster.
8. If this is the first time you've run any cluster jobs, either by using Excel or by using any other application, you'll see a user authentication dialog box. Type your user name and password, and (if you want) select the box to save your credentials. If you want to run calculations on the cluster as a different user, you can enter different user account settings.

**Note**

Make sure that your user account—either your standard user account or whatever you want to use—is designated as a cluster user in HPC Cluster Manager.

If there are any errors, you'll see a pop-up dialog box. If the desktop calculation worked, the most likely errors at this point will have to do with the settings that you created for the cluster head node and the share directory. Double-check those settings and try again.

If there are no errors, you'll see the cluster calculation start. First, there will be a brief pause as the client library creates an HPC session for calculation. Next, the client library will call the **HPC_Initialize** macro, and you'll see that the table is erased. If you look at the Excel status bar, you'll see that first, all the records are sent to the cluster; the second number will jump up to 105 (there are 105 cells in the table). After that, you'll see another brief pause before any results are returned. This pause occurs as each compute node starts Excel and loads the workbook.

After this pause, you'll see results start to populate the table, and the status bar will start showing the number of results returned. Because this is a long calculation, it's very likely that you'll see results returned out of order. There will be blank spaces in the table, which then are filled in. Once again, that's because the calculation is asynchronous, and some compute nodes calculate faster than others. Eventually, the table will be filled in completely and the status bar will show the calculation time.

Even if you have only two compute nodes in your Windows HPC Server cluster, the calculation should be a lot faster than running on the desktop. If you have four nodes, eight nodes, or more, the calculation should be extremely fast. You can always click the **Desktop** button again to compare the performance of the desktop calculation to the cluster calculation.

## 7.6 Best Practices for Cluster-Enabled Excel Workbooks

The following sections describe some important considerations to keep in mind when you are designing Excel workbooks for use with HPC Services for Excel. This information applies to workbooks that use the HPC/Excel macro framework running from VBA or with the .NET ExcelClient library.

### 7.6.1 Cluster Application Performance

When you design an application for calculation on a Windows HPC Server cluster, typically you divide the work required into individual tasks, and you send those tasks to the cluster for calculation. The performance benefits of using HPC come from parallelization—having many compute nodes working at the same time.

It's the responsibility of the HPC scheduler (the head node) to distribute these tasks to the compute nodes for processing. However, to ensure that the compute nodes are never waiting for work, you should try to ensure that the HPC scheduler always has a list of tasks to process. If the scheduler has to wait for the next processing task, one or more compute nodes will sit idle—doing nothing—and you won't get the best utilization of your Windows HPC Server cluster.

You can ensure that the scheduler and the compute nodes always have enough work to do by sending computation requests as quickly as possible. In the HPC/Excel macro framework, requests are sent with the **HPC_Partition** macro. Therefore, it's important to ensure that the **HPC_Partition** macro is as fast as possible. If you can design the **HPC_Partition** macro to send tasks quickly, you'll ensure that the cluster is fully utilized, and you'll get the best possible performance from HPC.

When individual calculations on the cluster compute nodes are complete, results are sent back to your desktop application. In the HPC/Excel macro framework, results are processed through the **HPC_Merge** macro. You'll want to make this macro as fast as possible as well, for two reasons. First, if the **HPC_Merge** macro is slow to process results, this may needlessly occupy cluster resources. Second, slow processing of results will slow down your overall calculation—reducing the benefits of using HPC. If you ensure that the **HPC_Merge** macro processes results quickly, you'll release any cluster resources for other use, and you'll make the overall user experience better.

## 7.6.2 HPC/Excel Macro Performance

In this section, we'll describe a few steps that you can take to help make sure that your HPC-enabled Excel workbooks are as efficient as possible, so you can get the best possible performance when you run calculations on a Windows HPC Server cluster.

This list isn't comprehensive, but you may find some tips that apply to your workbook. And remember that these are optimizations—the best approach when designing HPC-enabled Excel workbooks is to design the calculation first and keep it as simple as possible. Then, after you have the workbook running correctly on the cluster, you can make the kind of changes described here to improve overall performance.

### 7.6.2.1 Avoid Opening and Closing Resources in the HPC_Partition and HPC_Merge Macros

Opening and closing external resources—such as log files or database connections—can take a long time, and may slow down the **HPC_Partition** and **HPC_Merge** macros. Additionally, you should avoid reading large data sets in these macros if possible. For example, you don't want to open a database connection, search the records, and locate a single record for each call to the **HPC_Partition** macro.

A better approach is to open these resources once, in the **HPC_Initialize** macro, and perform any initial processing steps. For example, if you need to load a large data set from a database, create the database connection and perform the query in the **HPC_Initialize** macro. Then, in the **HPC_Partition** macro, you can just move to the next record. You can close the database connection in the **HPC_Finalize** macro.

Similarly, if you are writing to a log file in the **HPC_Merge** macro, consider opening the file in the **HPC_Initialize** macro. In the **HPC_Merge** macro, you can write results one at a time, and then you can close the log file in the **HPC_Finalize** macro.

### 7.6.2.2 Avoid Recalculating the Spreadsheet in the HPC_Merge Macro

If you are inserting results into the spreadsheet in the **HPC_Merge** macro, you might inadvertently recalculate the spreadsheet. If the spreadsheet is very complicated, this can slow down processing.

By default, Excel will recalculate the spreadsheet whenever you change the value of a cell, including if you update the cell from a VBA macro. Excel will recalculate only those parts of the spreadsheet that are affected by the change—so if the cell sits by itself, this won't be significant. But if the cell is used as part of a larger calculation, or as chart data, the recalculation may be slow.

If you are updating any spreadsheet cells in the Merge macro, consider disabling automatic recalculation. You can disable automatic recalculation in the **HPC_Initialize** macro. Then, in the **HPC_Finalize** macro, you can recalculate the entire spreadsheet and re-enable automatic calculation.

In the **HPC_Initialize** macro, you can turn off automatic calculation by using the following code.

```
Application.Calculation = xlCalculationManual
```

You can then recalculate the spreadsheet and re-enable automatic calculation in the **HPC_Finalize** macro by using the following code.

```
Application.Calculate
Application.Calculation = xlCalculationAutomatic
```

#### 7.6.2.3 Insert Blocks of Data Instead of Individual Values

When you update spreadsheet cells from VBA, this involves a number of internal steps—and it may be slower than you expect. It is almost always better to change multiple cells as a single block, instead of one at a time.

If the **HPC_Execute** macro returns a large number of values—an array of 100 numbers, for example—there are two ways to insert this data into the spreadsheet. You can insert them one at a time, in a loop, as follows.

```
For i = 1 To 100 Step 1
   Cells(1, i).Value2 = v(i)
Next i
```

Or, you can insert the full range as an array, as follows.

```
Range(Cells(2, 1), Cells(2, 100)).Value2 = v
```

Inserting one cell takes about the same amount of time as inserting a block of cells. So if you have an array of 100 values, the first way of doing this—inserting cells one at a time in a loop—can take up to 100 times as long as the second approach, inserting the block of cells as an array.

#### 7.6.2.4 Turn off Screen Updates

If you are making any changes to the spreadsheet in the **HPC_Merge** macro—if you are updating cells, or changing chart data—turning off screen updates can reduce the processing time. This works because Excel consumes some time updating the screen and the user interface.

You can disable screen updates in the **HPC_Initialize** macro and re-enable screen updates in the **HPC_Finalize** macro. In the **HPC_Initialize** macro, use the following code.

```
Application.ScreenUpdating = False
```

In the **HPC_Finalize** macro, use the following code.

```
Application.ScreenUpdating = True
```

That will disable screen updates during the calculation and re-enable updates after the calculation is completed. When you re-enable screen updates in the **HPC_Finalize** macro, Excel will automatically refresh the screen.

#### 7.6.2.5 Use Data Structures to Temporarily Hold Information in Memory

If collecting information for processing—what you do in the Partition macro—is very time-consuming, or if you need to do very complex post-processing of results in the **HPC_Merge** macro, consider using data structures to store information in memory during the calculation.

For example, if each of your calculation requests in the **HPC_Partition** macro requires a very complicated database query, this can slow down processing and result in poor performance. In this case, it may be a good idea to perform this complex processing in the **HPC_Initialize** macro before the calculation starts.

If you perform this complex processing in the **HPC_Initialize** macro, you can store each of the calculation requests in a data structure in VBA, such as an array of variants. Then, in the **HPC_Partition** macro, you can simply return the next entry in the array. That will help ensure that the **HPC_Partition** macro is as fast as possible.

Similarly, if you must perform very complex post-processing on results as they arrive, it may be advisable to store the results in a data structure in the **HPC_Merge** macro, and then process the results in the **HPC_Finalize** macro after the calculation is complete.

For example, if you are storing results in a database, but each database write requires a complex insert statement, it may be better to move this processing to the **HPC_Finalize** macro. You can create a data structure—such as an array of variants—and in the **HPC_Merge** macro, simply insert each result into the array. Then, in the **HPC_Finalize** macro, you can loop through the array and perform the necessary database write operations all at once.

## 7.7 Final Notes

Converting an existing workbook to use HPC Services for Excel is more of an art than a science. The specific details—what code to move, and where to move it—will always depend on the particular workbook.

However, you should now understand the concepts involved, and understand why we use the asynchronous macro framework. To convert any workbook to use HPC Services for Excel requires creating a parallel calculation in essentially the same way: add the macro framework functions to your workbook, and fill in the appropriate code.

You can use the skeleton macro files included with this chapter in your own workbooks. Follow the steps you used in the examples above to identify the iterative code in your workbook, and move it to the appropriate macros. And remember to test the calculation on your desktop to identify any errors before you run calculations on the cluster.

You should also have a sense of the performance benefits that you can get from running Excel workbooks on a Windows HPC Server cluster with HPC Services for Excel. In this chapter, we started with a workbook that was already reasonably fast—to calculate the entire workbook took only about one to two minutes on the desktop. But even that workbook might run twice as fast, four times as fast, or faster, on a Windows HPC Server cluster. The actual performance depends on the number of available compute nodes in your cluster, and you can always improve performance by adding more compute nodes to the cluster.

With a relatively small amount of work, it's possible to convert those long-running Excel workbooks to run on a Windows HPC Server 2008 R2 cluster with HPC Services for Excel. And for really long-running workbooks—workbooks that take hours or even days to calculate—using HPC Services for Excel can result in really dramatic performance improvement.

# Chapter 8 User Experience: SAIC (Shanghai Automotive Industry Corporation)

The adoption of high performance computing in the Chinese manufacturing industry is very low, especially when compared to the population in China. As stated by Professor Lang Xianping, the manufacturing industry in China is profiting mainly from the processing of materials. This type of business earns the lowest revenue in the supply chain. Without engaging in more design activities, manufacturing companies cannot increase profitability to ensure continuous growth. In this chapter, we will analyze SAIC, a Chinese corporation with the competency to design locally. HPC resources are deeply involved in the design activities of SAIC products.

In the past, the IT department and the engineers in SAIC encountered countless issues when managing and using Linux HPC clusters. It was difficult to popularize the use of HPC resources by departments in this corporation. Things changed in 2009, when the IT department in SAIC successfully deployed Windows HPC Server clusters and applications. HPC resources are now being used by hundreds of engineers from multiple departments. Increasing use of HPC resources has boosted the productivity of design jobs and reduced the cost of daily operations for the IT department.

In this chapter, we will discuss the business requirements for SAIC, their pain-points when working with Linux clusters, and the benefits brought by the Windows HPC Server cluster solutions. In previous chapters, the reader may have learned how Windows HPC Server can help to simplify the process of configuring, customizing, and managing cluster systems and jobs. Our focus for this chapter is to introduce the reader to a Web portal for simulated systems, which simplifies the process of running applications and monitoring results for engineers.

Sections 8.3 to 8.6 are excerpts of the white paper: "*SimCloud Solution – Over Windows HPC Architecture*" by VanCloud Info-Tech Co., Ltd. The content in those sections are the intellectual property of VanCloud Info-Tech Co.,Ltd. We extend our gratitude to VanCloud Info-Tech Co., Ltd. for authorizing us to include that content in this book.

Chapter Highlights:

- Introduction to SAIC and its business requirements
- Facts and challenges of supporting CAE (computer-aided engineering) by using HPC resources
- SimCloud: a Web portal over Windows HPC Server

## 8.1 Introduction to SAIC and its business requirements

As one of the top three automotive manufacturers in China, SAIC is focused on the development, manufacturing, and sales of cars, commercial vehicles, and accessories. In 2008, SAIC sold 1.826 million vehicles, with total sales at 24.88 billion USD. This positioned SAIC as the number one automotive manufacturer in China, as well as the 359$^{th}$-ranked corporation in the list of the top 500 global corporations.

In the early days, SAIC was focused on the manufacturing of vehicles for foreign automotive corporations, like Volkswagen and General Motors. In 2004, SAIC started its own brand of vehicles. In those days, SAIC had to rent equipment for the design and the testing of its vehicles. This changed in 2006, when they set up their own design center. SAIC is now designing, manufacturing, and selling its own brand of cars, including the Roewe 550, the Roewe 750, the MG 3-SW, the MG 7, and the MG TF. The HPC center in SAIC was set up to support designing and analyzing the safety of its own brand of cars.

In 2007, SAIC set up an HPC cluster running Red Hat Linux. That cluster was focused on executing CAE applications, including FLUENT, STAR-CD, STAR-CCM, LS-DYNA, MSC.Nastran, and ABAQUS.

For engineers and IT professionals who are familiar with Windows workstations and servers, it can be quite challenging to manage, customize, and use an HPC cluster that is based on Linux. Engineers in SAIC had to use the Linux command line to interact with the cluster. This user experience was both unfriendly and inefficient. It also made it difficult for the IT department in SAIC to popularize the use of HPC resources by more departments. Engineers generally preferred to continue to use applications that were running on individual Windows workstations.

## 8.2 The Solution of Windows HPC Server and its Advantages

In the summer of 2009, SAIC installed a cluster running Windows HPC Server 2008 on Hewlett-Packard blades. They quickly discovered that the performance of a Windows HPC Server cluster is comparable to that of Linux clusters. Furthermore, it can be much easier to use and manage a Windows HPC Server cluster than a Linux cluster. Most important, SAIC can fully customize the cluster to meet the different needs of its different departments.

SAIC, together with Microsoft Corporation and VanCloud, developed a Web portal. Ten different CAE applications were deployed to the Windows HPC Server cluster. It took two engineers two months to complete the Web portal. The Web portal seamlessly integrated the Windows HPC Server 2008 Job Scheduler, Windows HPC Server 2008 management tool, and the CAE applications required by SAIC. A graphical user interface (GUI) was provided for submitting and managing jobs. At the same time, the integration between the Web portal and Active Directory Domain Services helped engineers to use the cluster as conveniently as accessing any intranet resources.

By using the Job Scheduler in Windows HPC Server, SAIC achieved the goal of scheduling jobs efficiently, and according to policies. Key jobs get the priority level that they require, compared to less important ones. SAIC can customize the Web portal for load-balancing among different departments as needed. And that is very difficult to achieve on a Linux cluster. Nowadays, more than 200 engineers from five departments are benefiting from the Windows HPC Server cluster, and that number continues to grow. Engineers in remote sites can access the Windows HPC Server cluster through the Web portal. Engineers can work in a familiar environment, with a friendly user interface. As a result, they can focus their time on their core competency, which is to design top-class vehicles.

Transitioning to Windows HPC Server clusters helped SAIC to greatly simplify the way it uses HPC resources. The HPC resources have been well received by more departments, boosting the efficiency of engineers. By using the Windows HPC Server management console, IT professionals improved their efficiency by at least 20%.

SAIC is now planning to double the size of their HPC clusters. At the same time, the Linux clusters will be migrated to Windows HPC Server clusters.

## 8.3 Computer Aided Engineering (CAE) Application and Challenges in High-Performance Computing

### 8.3.1 Scenarios

Engineering simulation software like CAE/CFD represents one important type of HPC application. And HPC is the most important performance optimization solution for engineering simulation software applications. For the past decade, industry and HPC keep on communicating needs, try to gradually clarify, deepen, and enhance each other's value, and maintain, enhance, and extend partnerships. However, due to different positions, industry has failed to deliver an HPC solution that addresses the real needs of the customer and is easy to deploy.

VanCloud, as an independent HPC/CAE solution provider, has studied simulation application customers in different industries comprehensively and summarized the following typical scenarios:

Scenario 1: Engineer Andy has been using a PC for computation since he started learning simulation software. But when the CPU speed became a bottleneck, his productivity has been stagnating. Andy is distressed by not being able to solve engineering problems by large-scale grid simulation.

Scenario 2: After Mark in Corporate IT gets to know Andy's problem, he recommends that Andy set up a small compute cluster with multiple PCs and apply to management for introducing parallel simulation software licenses. The use of parallel computing mitigates the problem. However, soon Andy is stumped again: the PC network is extremely unstable, often makes calculation work fail, and cannot guarantee working efficiency. Also, the simulation team brought on a new member, Ryan, and Andy and Ryan have to coordinate to share the same license due to the high license price. Often, Ryan cannot get the license and can't carry out his work effectively.

Scenario 3: IT manager Vincent came across the situation, and decided to introduce HPC cluster with scheduling software. Andy, using the manual, types in some commands and successfully ordered the cluster to run calculation jobs. Ryan also learns to use the cluster when it's not busy with running other work. Now the simulation team is functioning again. But it doesn't last long before Vincent found that Andy and Ryan didn't come to the cluster anymore, and were running simulation software on their PC again. Vincent is angry, saying, "We've paid a lot of money to set up the computing device and scheduling software. Why weren't you using it?" Andy and Ryan responded, "It's too complicated to input the long list of commands. It's far easier to run it on a PC. And after submitting a computing task, I have to check the state now and then. It's just too much trouble! I'd rather run the problem on a PC if it's a small-scale problem."

Scenario 4: Ryan gradually gets familiar with the simulation software and the workload is increasing. The manager gives him an urgent task. Ryan went to the cluster immediately, only to find that Andy is already running a big computing task on the cluster. Ryan is anxious but cannot stop Andy's calculation. Ryan called the management team, and managers in turn talk with Andy to persuade him to stop the task. Andy reluctantly enters the command to save the intermediate data and abort the calculation so that Ryan can start this. Half a day has passed during this process, and there is no way for Ryan's calculation to finish on time.

Scenario 5: Andy finally gets his vacation and is sleeping on the beach to get tanned. Suddenly, management calls him with an urgent computation task which Ryan cannot take over. Andy wants to set up the task with his notebook but cannot connect to the cluster. He can only FTP it to Ryan so that he can trace it on the cluster. After Andy is back from vacation, Ryan tells him that the calculation didn't succeed because he didn't know how to adjust parameters. Only Andy knows how. Yet another computing task failed to meet the deadline.

Scenario 6: After years of working, Andy becomes famous in the industry and decides to join a high-profile enterprise. When he gives his two-week notice, management says, "Andy, you can't leave! If you leave, we have to train the new people. Plus, how can you give your years of experience to your successor?" Andy responds, "I'll give my hard disk to the company, but I am not sure about the exact usage of those files. To be honest, the experience is in my head, and I don't have time to clean them up!"

### 8.3.2 Dilemma: The delivery of HPC processing capability

In scenario 3, HPC processing capability doesn't meet the initial expectation. The root cause is that the current scheduling software is too complex. And all main features are hardware focused, and there is no tailor for different simulation software. When engineers want to do calculations, they still need to memorize those obscure commands and debug the complex compute cluster. Furthermore, when an engineer leaves the corporate network environment, they are isolated from enterprise computing resources, so they can't handle urgent tasks efficiently.

The ideal simulation cloud platform should shadow the cluster hardware and scheduling system. Engineers should be able to complete their work in a normal Web browser and use enterprise computing resources securely from anywhere. This is where the SimCloud simulation cloud platform shines.

### 8.3.3 Issue: The software license management chaos

Scenario 4 is a common problem in a lot of enterprises—software license management chaos. As a result of the high price of simulation software, an enterprise tends to have limited licenses. However, an urgent project may suffer and is unable to deliver on time due to lack of licenses. The management team struggles to cope with this problem with manual planning. The software license management chaos has a negative impact on the engineering team's productivity and morale.

SimCloud introduces accurate software license management, optimizing in real time the use of CPU core and online users, providing a solution to the problem of license management.

### 8.3.4 Gap: Simulation data and process need to be supervised

The brain drain mentioned in scenario 6 is painful but inevitable. The key is to establish a simulation data and process management system, effectively supervising the engineering team work and turning the simulation research result into an enterprise knowledge base.

SimCloud Enterprise Edition has a built-in workflow engine, providing quick deployment customized for customer simulation workflow and data management, and a tailored simulation knowledge collection system, helping enterprise to eliminate the "information island" phenomenon, and improving enterprise's core innovation knowledge sharing mechanism.

## 8.4 SimCloud: The portal solution, which is based on Windows HPC Server

SAIC uses a High Performance Portal System from VanCloud—SimCloud.

## 8.4.1 Platform Architecture

SimCloud is a simulation cloud computing platform. The platform is designed as an enterprise HPC solution to integrate the CAE/CFD simulation applications, HPC cluster management, SOA (Service-Oriented Architecture), and so on.

We can share from multiple perspectives to analyze this new cloud computing platform. The overall network infrastructure of platform is shown in Figure 8-1.

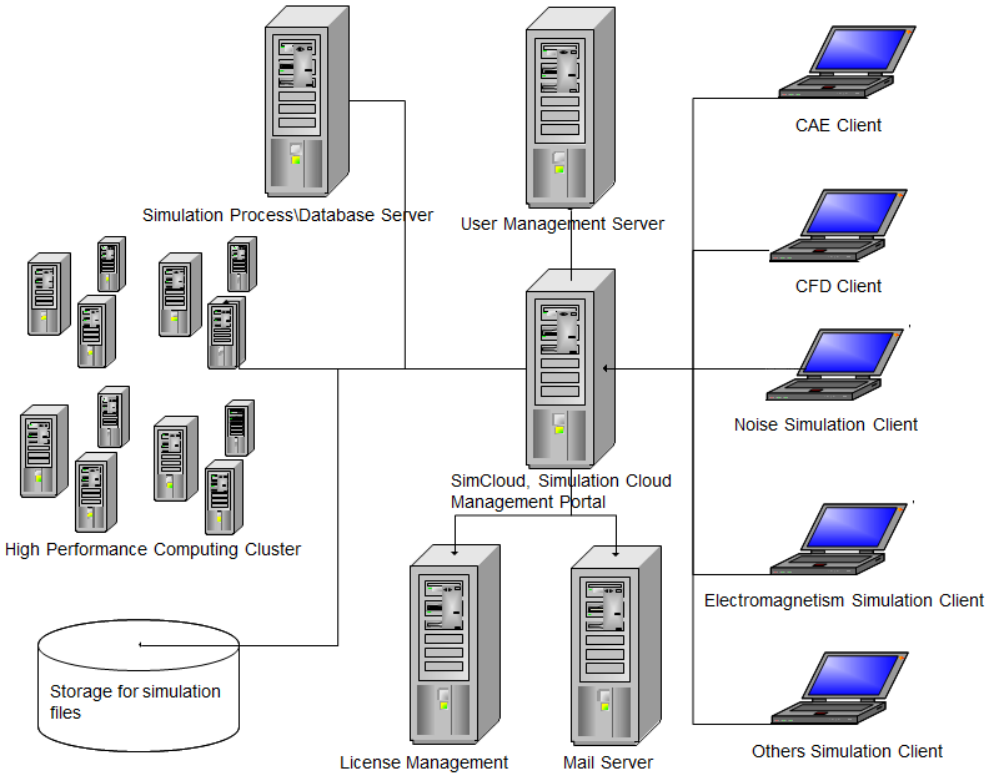



*Figure 8-1 Network infrastructure of an HPC platform*

The center of the platform architecture is a simulation cloud management portal. It is responsible for intercommunication between the clients and the HPC cluster, user management with Active Directory, mail service, simulation process/data management, simulation data storage, and other functions of the presentation layer services. The HPC cluster and related equipment are on the left of the simulation cloud management portal. On the right are the various types of simulation applications. (The figure is only an example; SimCloud can be extended to all simulation application clients.)

## 8.4.2 Feature Architecture

The feature architecture of the SimCloud platform can be divided into the resources layer, services layer, business layer, and presentation layer. These layers are shown in Figure 8-2.

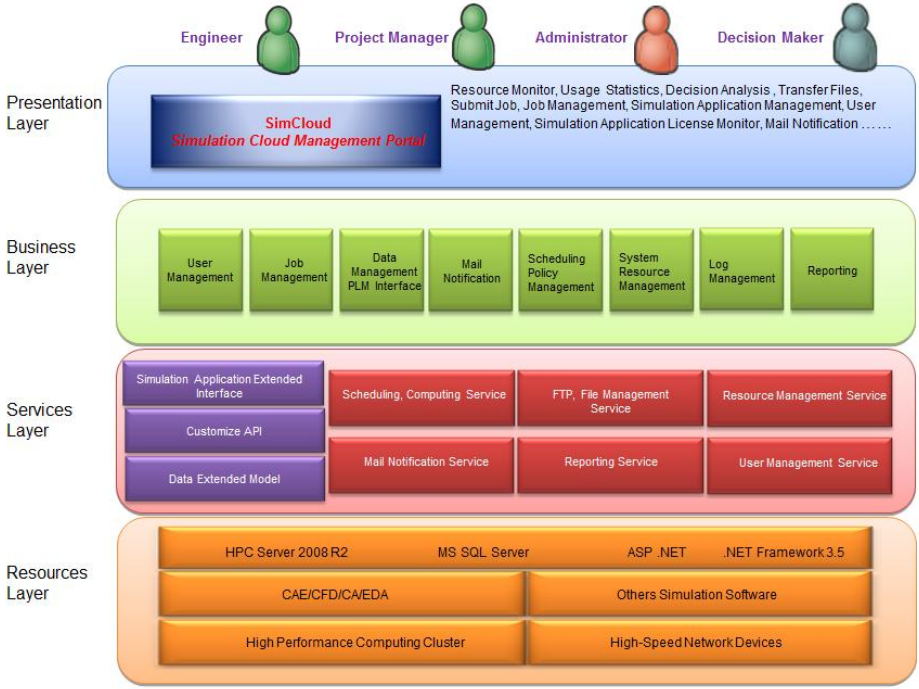



*Figure 8-2 SimCloud system stack*

Th resources layer is responsible for the integration of hardware resources, network resources, and software resources. The SimCloud platform is based on Windows HPC architecture. It uses the Windows HPC Server operating system for integration, and simulation software package into a Web service interface through the .NET service.

The services layer packages the features of the server role, such as Windows HPC Server R2, Email Server, FTP Server, AD Server, and so on, as well as provides a rich interface and extended function interfaces for the entire platform.

The business layer contains the business features, such as user management, job management, data management, mail notification, scheduling policy management, system resource management, log management, statistical reports, and so on, as well as provides a business process extension interface for PLM and other workflow systems.

The presentation layer integrates a series of application functionality as a Web portal by the SimCloud simulation cloud management portal, such as job submission, resource monitor, job management, file transfer, license management, usage statistics, user management, decision analysis, and so on.

## 8.4.3 Simulation engineer's scenarios

SimCloud provides ease of use and rich functionality.The application process is also closely bound to the different user roles of enterprise, and it aims for simple logic on the base of full functionality.

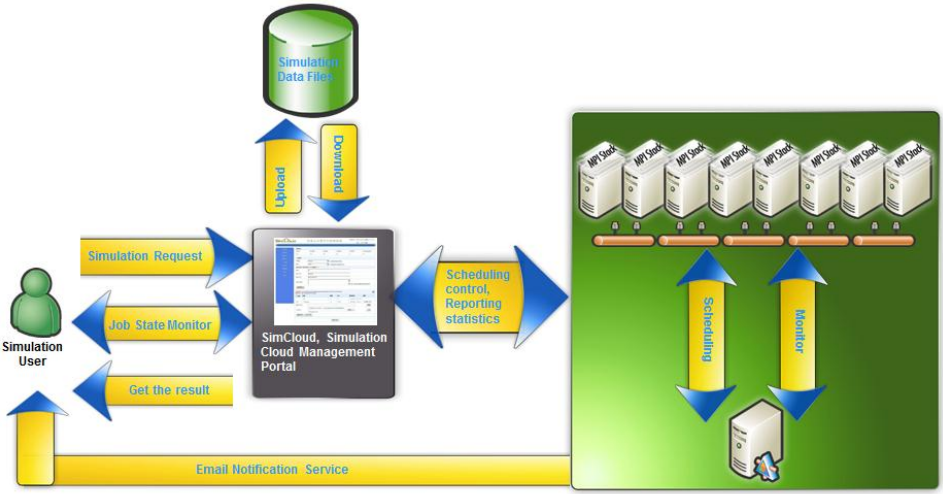



*Figure 8-3 Simulation engineer's scenario*

As shown in Figure 8-3, the simulation users can use a domain account to log in to SimCloud. It is easy to submit a simulation job to a head node of the HPC cluster. The head node followsthe established policy to schedule the jobs. It is based on the current utilization of hardware resources and the usage of simulation software licenses, submits the job to the queue, and automatically dispatches. After computing is completed, it will merge the simulation's data from the compute nodes, and then sends email to the simulation user. The user is able to obtain the simulation results from SimCloud.

# 8.5 Application Integration Case

SimCloud can integrate a variety of simulation applications, including computational structural mechanics, computational fluid mechanics, computational acoustics, multibody dynamics, computational electromagnetics, and other disciplines of common commercial software. It also provides APIs to facilitate integration of various applications. SimCloud is based on Windows HPC Server 2008 R2. It supports all the simulation software, which can be installed on Windows. The following lists the instructions and screenshots for only the common software.

## 8.5.1 ANSYS Integration

Figure 8-4 shows the UI of the ANSYS integration, which is computational structural mechanics software.

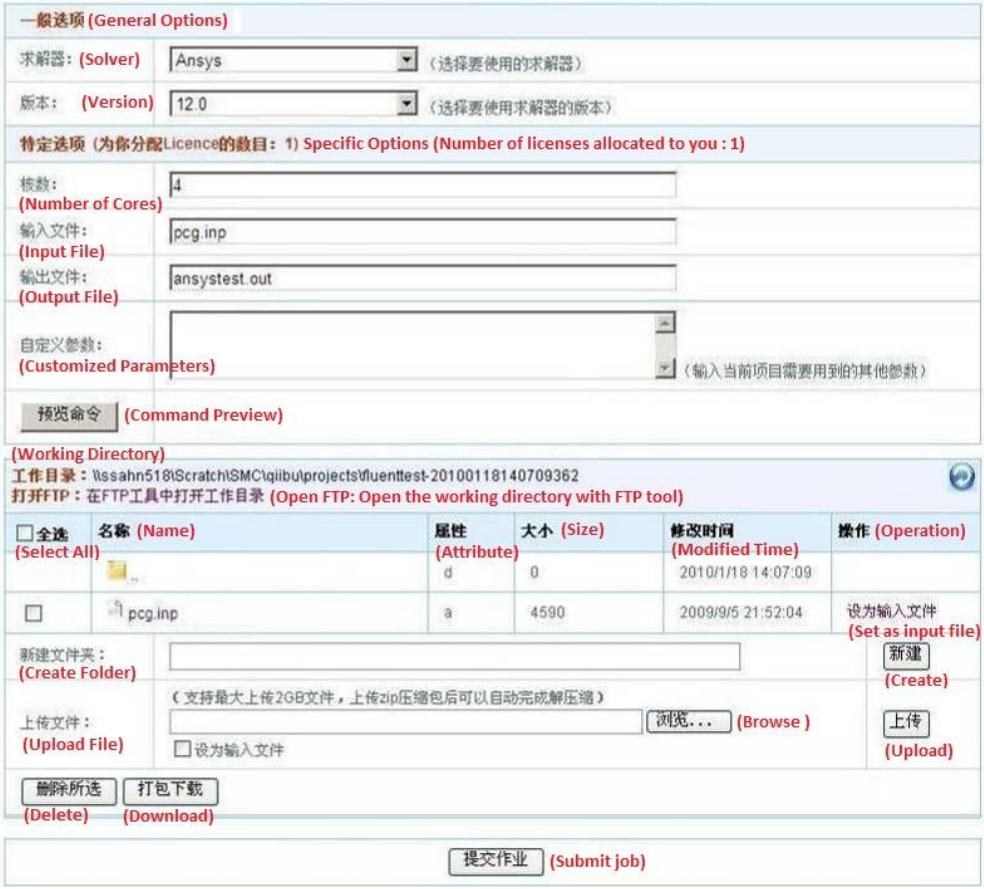



*Figure 8-4 Submit an ANSYS job*

The users select the corresponding version of the ANSYS solver, and then choose the parallel scale (that is, the number of CPU cores; the maximum is set by the administrator for the user), upload the input files to the specified working directory, and set the output file (which can be monitored the convergence in real-time), then submits the job to the simulation cloud system. An email notification will be sent to the engineer when the simulation job is started and when the job fails or succeeds.

## 8.5.2 Fluent Integration

Figure 8-5 shows the UI of Fluent integration, which is nonlinear mechanics software.

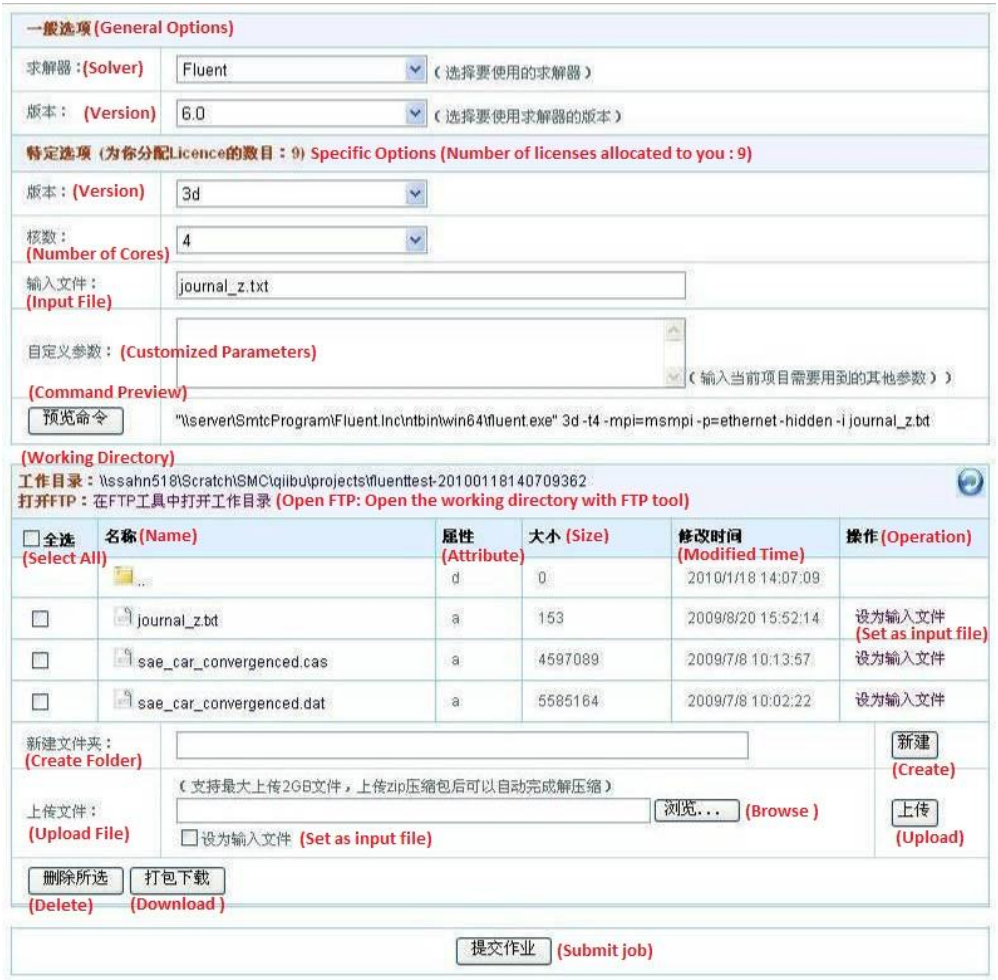



*Figure 8-5 Submit a Fluent job*

The users select the Fluent solver, and then choose the space's topological properties of the solving calculation instances (2D or 3D), the parallel scale (that is, the number of CPU cores; the maximum is set by the administrator for the user), and the standard script files. (The case (.cas) and data (.dat) files and the related computational iteration control are defined in script files.) An email notification will be sent to the engineer when the simulation job is started, and when the job fails or succeeds. The simulation job of Fluent can be carried out through a Web page monitoring convergence curve.

## 8.5.3 ABAQUS Integration

Figure 8-6 shows the UI of the ABAQUS integration, which is computational fluid mechanics software.

The users select the ABAQUS solver, and then choose the version of the solver and the parallel scale (that is the number of CPU cores; the maximum is set by the administrator for the user), uploads the input file (.inp) to the specified working directory, and submits the job to the simulation cloud system. An email notification will be sent to the engineer when the simulation job is started, and when the job fails or succeeds.

*Figure 8-6 Submit an ABAQUS job*

## 8.5.4 Nastran integration

Figure 8-7 shows the UI of the MSC.Nastran integration, which is computational structural mechanics software.

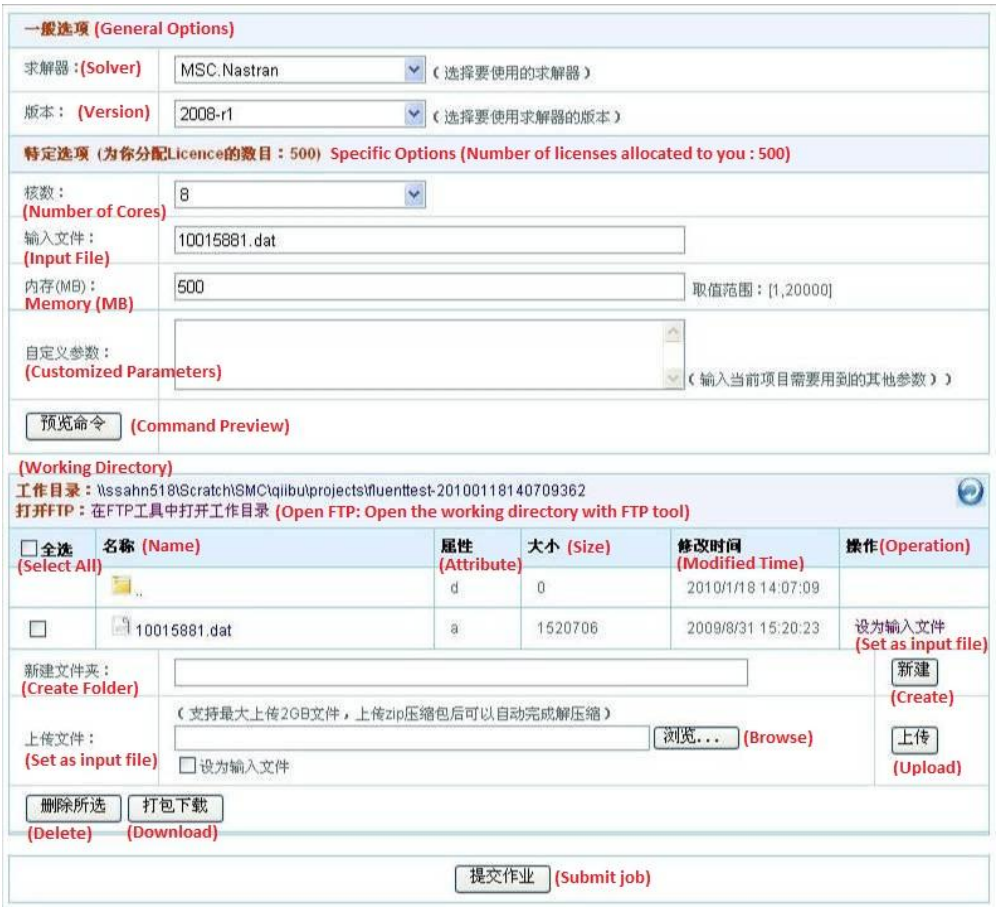



*Figure 8-7 Submit a Nastran Job*

The users select the MSC.Nastran solver, and then choose the version of the solver and the parallel scale (that is, the number of CPU cores; the maximum is set by the administrator for the user), sets the amount of memory required for the job, uploads the input file (.dat) to the specified working directory, and submits the job to the simulation cloud system. An email notification will be sent to the engineer when the simulation job is started, and when the job fails or succeeds.

## 8.5.5 LS-DYNA Integration

Figure 8-8 shows the UI of the LS-DYNA integration, which is explicit dynamical analysis software.

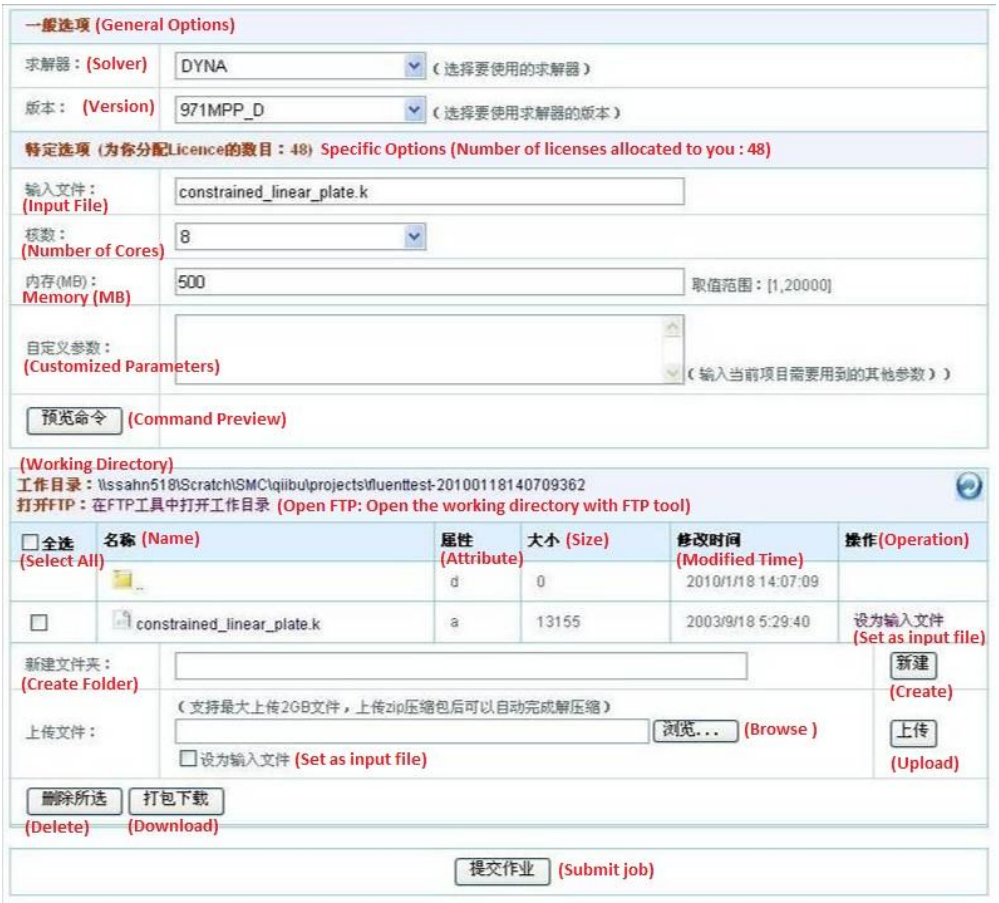



*Figure 8-8 Submit a DYNA job*

The simulation engineer selects the DYNA solver, and then chooses the version of the solver and the parallel scale (that is, the number of CPU cores; the maximum is set by the administrator for the user), sets the amount of memory required for the job, uploads the input file (.k) to the specified working directory, and submits the job to the simulation cloud system. An email notification will be sent to the engineer when the simulation job is started, and when the job fails or succeeds.

## 8.5.6 ADAMS Integration

Figure 8-9 shows the UI of MSC.Adams integration, which is virtual prototyping simulation analysis software.

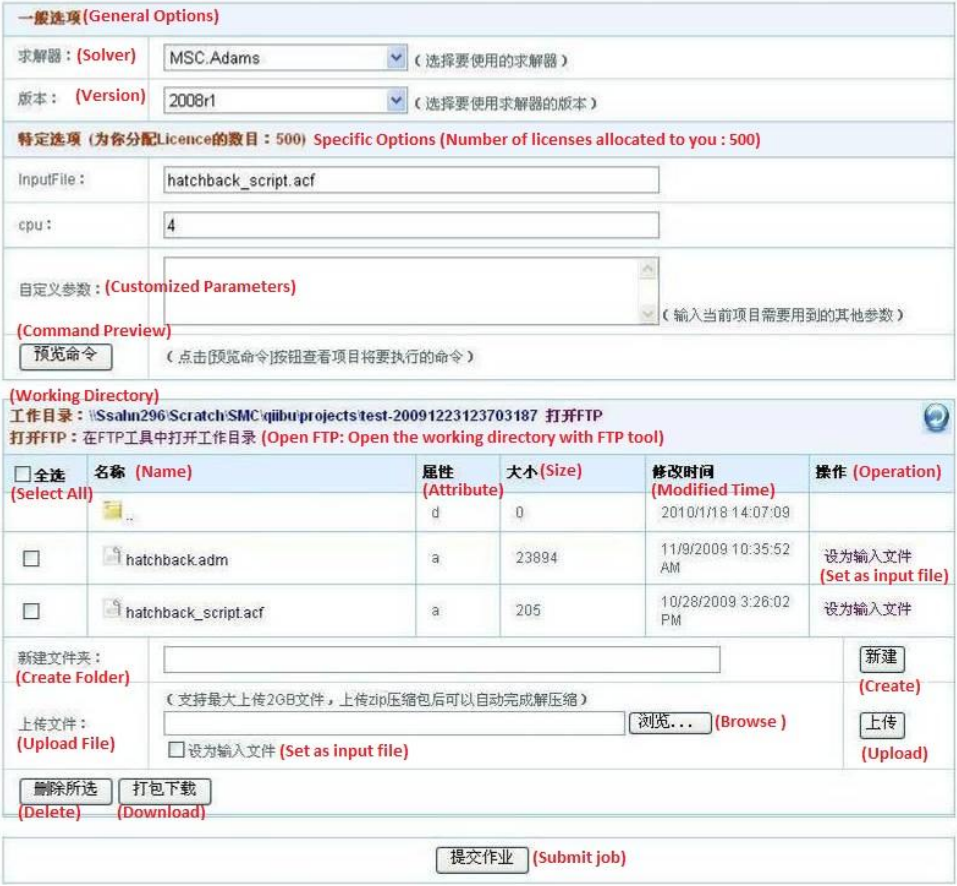



*Figure 8-9 Submit an MSC.Adams job*

The user selects the MSC.Adams solver, and then chooses the version of the solver and the parallel scale (that is the number of CPU cores; the maximum is set by the administrator for the user), uploads the input file (.acf) to the specified working directory, and then submits the job to the simulation cloud system. An email notification will be sent to the engineer when the simulation job is starts, and when the job fails or succeeds.

## 8.5.7 STAR-CCM+ Integration

Figure 8-10 shows the UI of the STAR-CCM+ integration, which is continuum mechanics simulation software.

The user selects the STAR-CCM+ solver, and then chooses the version of the solver and the parallel scale (that is the number of CPU cores; the maximum is set by the administrator for the user), uploads the input file (.sim) to the specified working directory, and submits the job to the simulation cloud system. An email notification will be sent to the engineer when the simulation job is started, and when the job fails or succeeds.

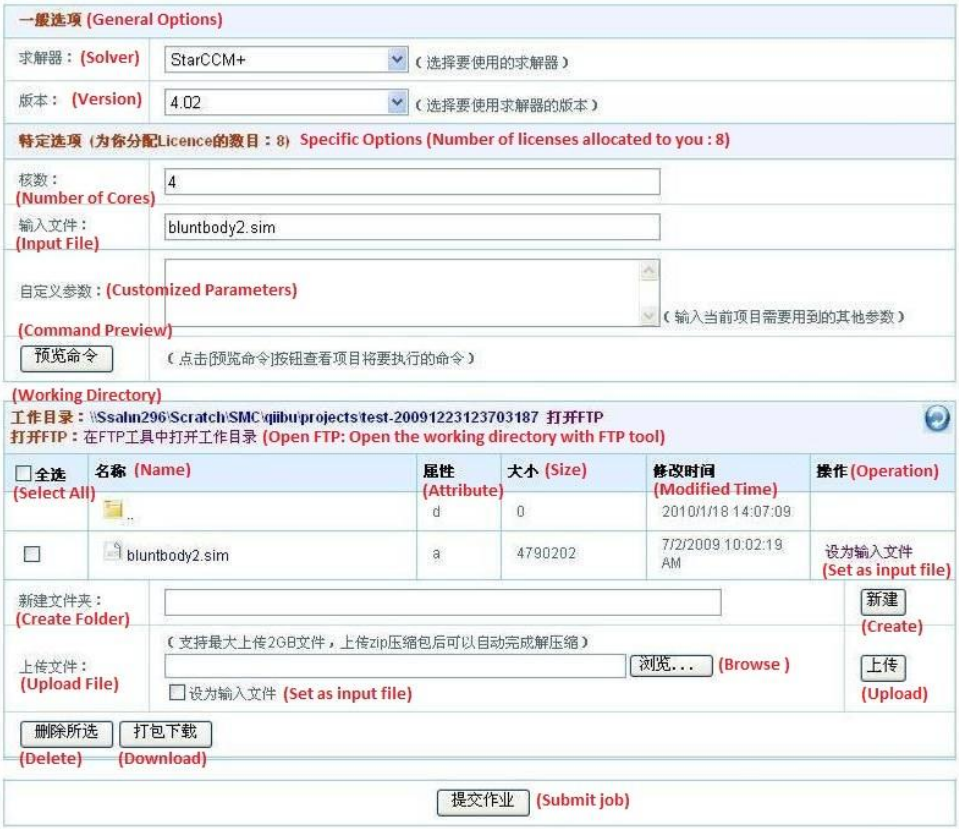



*Figure 8-10 Submit a StarCCM+ job*

## 8.5.8 RECURDYN Integration

Figure 8-11 shows the UI of the RECURDYN integration, which is multi-body dynamics software.

*Figure 8-11 Submit a RecurDyn job*

The user selects the RECURDYN solver, and then chooses the version of the solver and the parallel scale (that is the number of CPU cores; the maximum is set by the administrator for the user), uploads the input file (.rdyn, .rss) to the specified working directory, and submits the job to the simulation cloud system. An email notification will be sent to the engineer when the simulation job is started, and when the job fails or succeeds.

## 8.6 License Management

The license management module of SimCloud shows the licenses of simulation software for dynamic real-time asset management. It is able to monitor the usage of licenses that provides the data support for asset utilization statistics. Figure 8-12 shows the list of simulation software licenses. This list shows the number of licenses for the simulation solvers which are integrated into the SimCloud platform and their current usage in the system.

查看许可(View License) abaqus=0;

○ DYNA 971_S ○ RecurDyn 4.08 ○ MSC.Adams 2008r1 ○ Fluent 6.0 ○ MSC.Nastran 2008-r1 ○ StarCCM+ 4.02 ○ StarCD 4.08
◉ Abaqus 6.9-1 ○ NX.Nastran 6.1 ○ DYNA 971_D

| Features | Total | Used |
|---|---|---|
| abaqus | 20 | 8 |
| standard | 20 | 8 |
| parallel | 16384 | 4 |
| cae | 2 | 1 |
| cosim_direct | 1 | 0 |
| explicit | 20 | 0 |
| multiphysics | 1 | 0 |

*Figure 8-12 Software license management*