Using Microsoft Windows HPC Server 2008 Job Scheduler

Microsoft Corporation

Published: February 2008

Abstract

Windows HPC Server 2008, the successor to Windows Compute Cluster Server (WCCS), includes a new Job Scheduler that provides greater scalability and supports advanced policies. The new Job Scheduler supports a new Service Oriented Architecture mode that provides access to interactive applications through the Windows Communication Framework. The graphical interface for the Job Scheduler is now fully integrated into the Administration Console, and the command-line interface now uses Windows PowerShell for all Job Scheduler functions, while maintaining compatibility with the command-line interface commands in WCCS. This paper explains the new Job Scheduler functionality.

This document was developed prior to the product's release to manufacturing, and as such, we cannot guarantee that all details included herein will be exactly as what is found in the shipping product.

The information contained in this document represents the current view of Microsoft Corporation on the issues discussed as of the date of publication. Because Microsoft must respond to changing market conditions, it should not be interpreted to be a commitment on the part of Microsoft, and Microsoft cannot guarantee the accuracy of any information presented after the date of publication.

This White Paper is for informational purposes only. MICROSOFT MAKES NO WARRANTIES, EXPRESS, IMPLIED, OR STATUTORY, AS TO THE INFORMATION IN THIS DOCUMENT.

Complying with all applicable copyright laws is the responsibility of the user. Without limiting the rights under copyright, no part of this document may be reproduced, stored in or introduced into a retrieval system, or transmitted in any form or by any means (electronic, mechanical, photocopying, recording, or otherwise), or for any purpose, without the express written permission of Microsoft Corporation.

Microsoft may have patents, patent applications, trademarks, copyrights, or other intellectual property rights covering subject matter in this document. Except as expressly provided in any written license agreement from Microsoft, the furnishing of this document does not give you any license to these patents, trademarks, copyrights, or other intellectual property.

© 2008 Microsoft Corporation. All rights reserved.

Microsoft, Active Directory, SQL Server, Visual C++, Visual C#, Visual Studio, Windows, the Windows logo, Windows PowerShell, and Windows Server are trademarks of the Microsoft group of companies.

All other trademarks are property of their respective owners.

Contents

Windows HPC Server 2008 Operations Overview

Microsoft® Windows® HPC Server 2008 extends the mission of Windows Compute Cluster Server 2003 to bring high-performance computing (HPC) to industry standard, low-cost servers that support larger and heterogeneous clusters. Jobs—discrete activities scheduled to perform on the compute cluster—are the key to operating Windows HPC Server. Cluster jobs can be as simple as a single task or can include many tasks. Each task can be serial—running one after another, or parallel—running across multiple processors. Tasks can also run interactively as Service Oriented Architecture (SOA) applications. The structure of the tasks in a job is determined by the dependencies among tasks and the type of application being run. In addition, jobs and tasks can be targeted to specific nodes within the cluster. Nodes can be reserved exclusively for jobs or can be shared between jobs and tasks.

Note For general information about Windows HPC Server features and capabilities, see the white paper "Windows HPC Server 2008 Technical Overview." For overall management and deployment information, see the white paper "Windows HPC Server 2008 Management and Deployment."

The basic principle of job operation in Windows HPC Server 2008 relies on three important concepts:

- Job Submission
- Job Scheduling
- Job Execution

These three concepts form the underlying structure of the job life cycle in HPC and are the basis on which Microsoft engineered Windows HPC Server. Figure 1 illustrates the core relationship among each aspect of job operation. Each time a user prepares a job to run in the compute cluster, the job runs through the three stages.

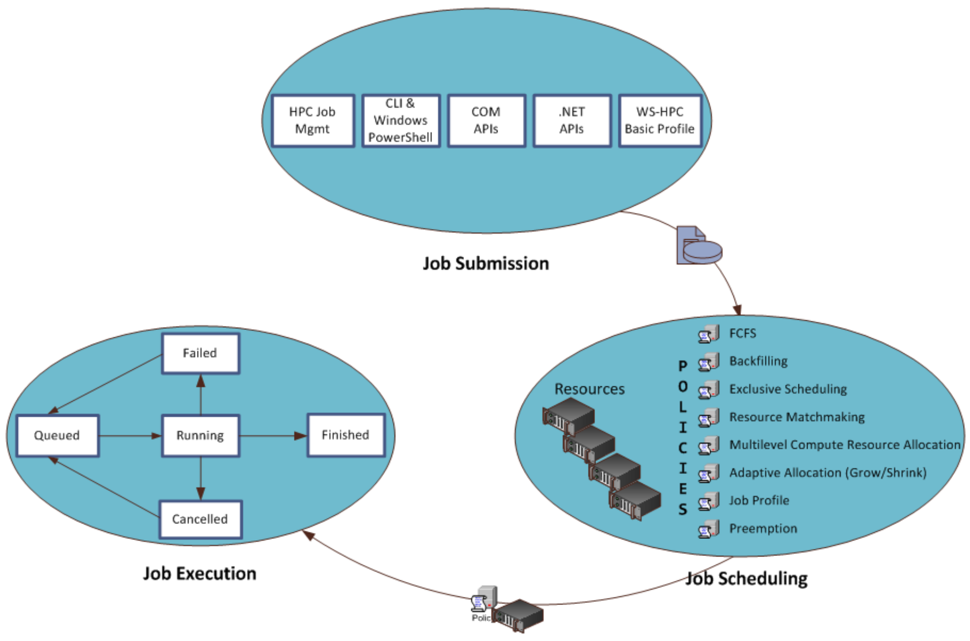


Figure 1 The HPC job life cycle

To understand how a job operates, you need to understand the components of an HPC cluster. Figure 2 shows the components and how they relate to each other.

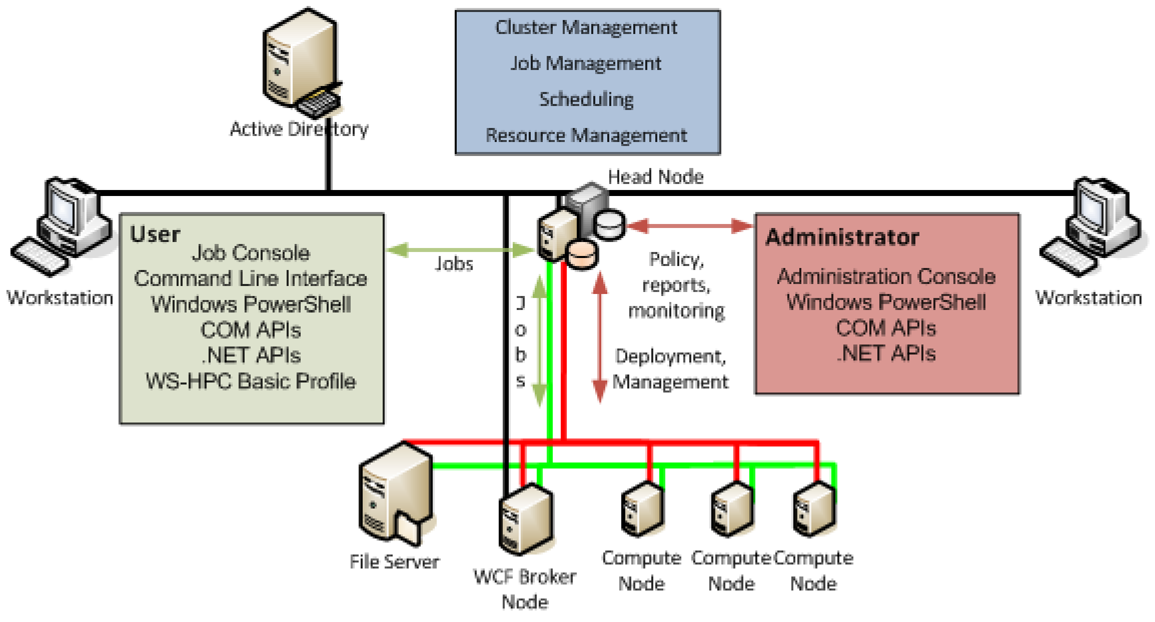


Figure 2 The elements of a compute cluster

A basic cluster consists of a single head node (or a primary and secondary head node if the deployed cluster is made ‘high available’) and compute nodes; it can also include one or more Windows Communication Foundation (WCF) Broker nodes. The head node, which can also operate as a compute node, is the central management node for the cluster, and manages nodes, jobs, and reporting for the cluster. The head node deploys the compute nodes, runs the Job Scheduler, monitors job and node status, runs diagnostics on nodes, and provides reports on node and job activities. WCF Brokers, used for interactive SOA applications, create interactive sessions that submit jobs to the scheduler, load-balance the assigned nodes, and finally return results to the client session. Compute nodes execute job tasks.

When a user submits a job to the cluster, the Job Scheduler validates the job properties and stores the job in a Microsoft® SQL Server® database. If a template is specified for the job, that template is applied, or the default template is used, and the job is entered into the job queue based on the policy specified. When the resources required are available, the job is sent to the compute nodes assigned for the job. Because the cluster is in the user’s domain, jobs execute using that user’s permissions. As a result, the complexity of using and synchronizing different credentials is eliminated, and the user does not have to employ different methods of sharing data or compensate for permission differences among different operating systems. Windows HPC Server provides transparent execution, access to data, and integrated security.

Windows HPC Server 2008 Terminology

Although all compute clusters share some terminology, there is also terminology specific to Windows HPC Server. To make sure we're all on the same page, it's useful to define the nomenclature you'll need when running Windows HPC Server.

Cluster

A cluster is the top-level unit of Windows HPC Server. A cluster contains the following elements:

- **Node.** A single physical or logical computer with one or more processors. Nodes can be a head node, compute nodes, or WCF Broker nodes.
- **Queue.** An element providing queuing and job scheduling. Each Windows HPC Server cluster contains only one queue, and that queue contains pending jobs. Completed jobs are purged periodically from the queue.
- **Job.** A collection of tasks that a user initiates. Jobs are used to reserve resources for subsequent use by one or more tasks.
- **Tasks.** A task represents the execution of a program on given compute nodes. A task can be a serial program (single process), or a parallel program (using multi-threading, OpenMP, or MPI).

Job Scheduler

The Job Scheduler queues jobs and their tasks. It allocates resources to these jobs; initiates the tasks on the compute nodes of the cluster; and monitors the status of jobs, tasks, and compute nodes. Job scheduling uses scheduling policies to decide how to allocate resources. Figure 3 shows the Job Scheduler stack.

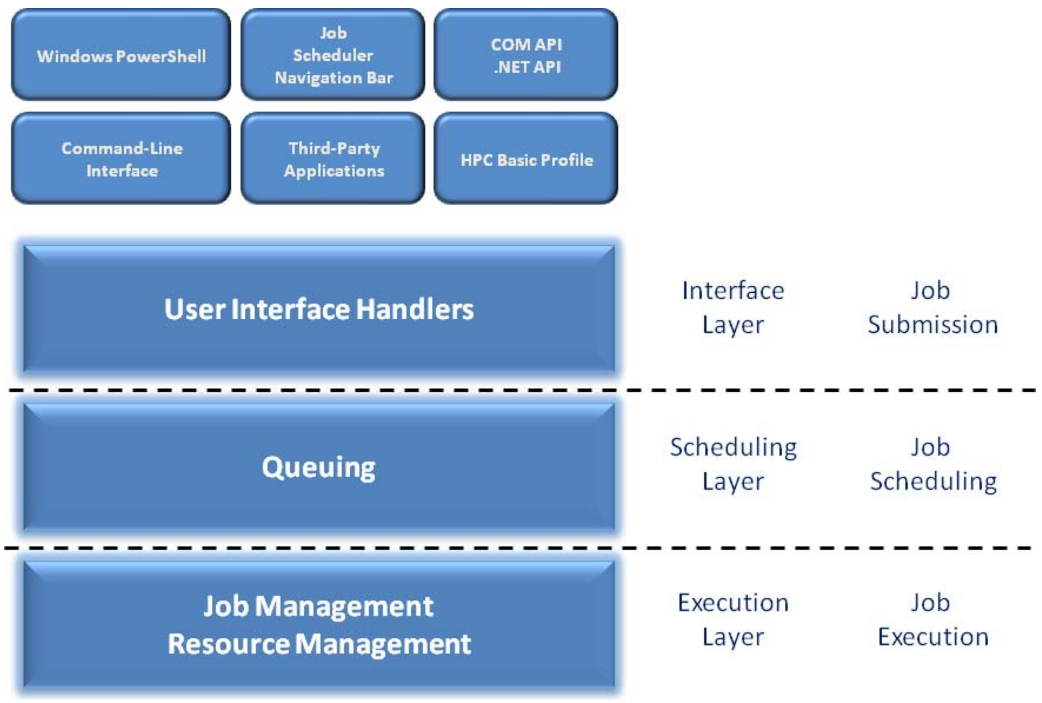


Figure 3 The Job Scheduler stack

- The interface layer provides for job and task submission, manipulation, and monitoring services accessible through various entry points.
- The scheduling layer provides a decision-making mechanism that balances supply and demand by applying scheduling policies. The workload is distributed across available nodes in the cluster according to the job profile.
- The execution layer provides the workspace used by tasks. This layer creates and monitors the job execution environment and releases the resources assigned to the task upon task completion. The execution environment supplies the workspace customization for the task, including environment variables, scratch disk settings, security context, and execution integrity as well as application-specific launching and recovery mechanisms.

Administration Console

The Administration Console is the overall management interface for cluster administration. Based on the Microsoft® System Center user interface, it uses Navigation Bars to quickly change the context and view. The Job Manager Navigation Bar provides a graphical interface to job management and scheduling.

Scheduling Policies

Windows HPC Server uses nine scheduling policies, which are explained in detail later in this paper:

- Priority-based first come, first served (FCFS)
- Backfilling
- Exclusive scheduling
- Resource matchmaking
- Job template
- Multilevel compute resource allocation (MCRA)
- Preemption (Beta 2)
- Grow/Shrink Job Scheduling Policy

Task Execution

Windows HPC Server 2008 has two types of tasks—basic tasks and parametric tasks.

A basic task uses a single command line that includes the command to run, along with the metadata that describes how to run the command. A basic task can be a parallel task and can be run across multiple nodes or cores. Parallel tasks typically communicate with other parallel tasks in the job using the Microsoft Message Passing Interface (MS-MPI), or through shared memory when running on multiple cores on a single node.

A parametric task contains a command-line with wildcards, allowing the same task to be run many times with different inputs for each step. A parametric task can be a parallel task and can be run across multiple nodes or cores.

WCF Broker

Stores and forwards request/response messages between client and service instances.

Node Manager

The job agent and authorization service on the compute node. Starts the job on the node.

Service Oriented Architecture

Service orientation provides an evolutionary approach to building distributed computing software that facilitates loosely coupled integration and resilience to change. With the advent of the WS-* Web Services, architecture has made service-oriented software development feasible with mainstream development tools support and broad industry interoperability. Although most frequently implemented using industry standard Web services, service orientation is independent of technology and its architectural patterns and can be used to connect with legacy computing packages. Service orientation need not require rewriting functionality from the ground up. By wrapping existing HPC code into modular services, the customer can extract more value from what is already there and extend and evolve the existing applications beyond the boundaries of what they were designed to do, e.g., a batch computation solver can be rendered as interactive solver services.

Windows Communication Foundation

The unified programming model for building service oriented applications from Microsoft. WCF enables developers to build secure, reliable, transacted solutions that integrate across platforms and interoperate with existing investments.

Session

A connection between the application and the services. A session consists of a managed pool of service instances on the compute nodes so that the application can decompose the domain and distribute the calculation requests to the pool to accelerate the processing speed.

Navigation Buttons

A set of buttons at the lower left of the Administration Console that shift the view and context to different areas of Windows HPC Server 2008 management and administration. For example, clicking the Job Management Navigation button opens the Job Manager.

Job Submission

Windows HPC Server supports multiple job submission methods:

- The Job Manager UI in the Administration Console
- A standalone Job Console for use by non-administrators
- The Microsoft® Windows command-line interface
- The Microsoft® Windows PowerShell™ command-line interface
- High Performance Computing Basic Profile—an Open Grid Forum standards-based Web service that uses a Web Services Interoperability Organization base (Beta 2 feature)
- A COM interface for integration with Microsoft Visual C® or Microsoft Visual C++® applications. The COM interface also supports other languages and scripting
- .NET API's for use with .NET applications

A variety of additional scripting languages are supported in the command-line interface and the COM interface, including Perl, Microsoft Visual C, Visual C++, and Microsoft® Visual C#®, and Java.

The Windows HPC Server Job Manager includes powerful features for job and task management, among them, automatic retrying of failed jobs or tasks, identification of unresponsive nodes, and automated cleanup of completed jobs. Each job runs in the security context of the user, ensuring that jobs and their tasks have only the access rights and permissions of the initiating user. All the features of the Job Manager are available from the command line as well.

Each task within a job can be either serial or parallel. For example, when performing a parametric sweep, the job consists of multiple iterations of the same executable that are run concurrently but use different input and output files. There is typically no communication among the tasks, but parallelism is achieved by the scheduler running multiple instances of the same application at the same time. This is also sometimes referred as "embarrassingly parallel."

Users submit jobs to the system, and each job runs with the credentials of the user submitting the job. Jobs can be assigned priorities upon submission. By default, users submit jobs with the Normal priority, but the default priority can be set in the Job Template. If the job needs to run sooner, cluster administrators can assign a higher priority to the job, such as AboveNormal or Highest. A Job Template can specify a different default priority, and users can be given permission to use a specific Job Template.

Jobs also consume resources—nodes, processors, and time—and these resources can be reserved for each specific job. Although one might assume that the best way to execute a job is to reserve the fastest and most powerful resources in the cluster, in fact, the opposite tends to be true. If a job is set to require high-powered resources, it may have to wait for those resources to be free so that the job can run. Jobs that require fewer resources with shorter time limits tend to be executed more quickly, especially if they can fit into the backfill windows (the idle time available to resources) that the Job Manager has identified.

Another factor that affects execution time is node exclusivity. By default, jobs require node exclusivity, though this can vary based on Job Template settings. However, if jobs are defined to require nonexclusive node use, they have faster access to resources because resources can be shared with other jobs. (Any idle resource that can be shared can run other jobs as soon as it is available.)

Creating Jobs

Users create jobs by first specifying the job properties, including priority, the runtime limit, the number of required processors, requested nodes, and node exclusivity. After defining the job properties, users can assign tasks to the job. Each task must include information about the commands to be executed; input, output, and error files to be used; as well as properties similar to those of the job in terms of requested nodes, required processors, the runtime limit, and node exclusivity. Tasks also include dependency information, which determines whether a job runs in serial or parallel mode.

Users create jobs by using either the Job Management Navigation button in the Administration Console, the standalone Job Management console, or at the command line, using either Windows PowerShell or the earlier command line interface. In the Job Management Console (or by using the Job Management Navigation button in the Administration Console), select Create New Job from the Actions menu to create a new job.

Creating Jobs Using the Command Line

To create a job through the Windows HPC Server command-line interface, users type the following command:

```
job new [options]
```

where *job_file* is the source file used for running the job and *host* is the name of the hosts the job will run on.

Table 1 lists the standard options available with the **job** command. Note that when dealing with job priorities, users have access only to Normal, BelowNormal, and Lowest, while administrators have access to all the available priorities. If users need to have their jobs run sooner than scheduled, they must ask a cluster administrator to increase the priority of their jobs, or they must use a Job Template that has a higher priority. Administrators can grant permission to users for Job Templates with higher priority.

To create a job through Windows PowerShell, the command is:

```
New-job [options]
```

Windows PowerShell greatly expands the options for managing jobs and their tasks in Windows HPC Server. There will be an entirely new set of PowerShell cmdlets in Beta 2.

Table 1. Job Properties

Property	Description
Project name	Specifies the project name with which the job is associated, for accounting purposes.
Job name	Name of job.
Number of Processors	Specifies the minimum and maximum number of compute processors to be reserved across a set of nodes.

Property	Description
Asked nodes	Specifies a list of nodes eligible to be used by the job.
Priority	Priority group in which the job is placed in the queue: **Highest** **AboveNormal** **Normal** **BelowNormal** **Lowest** (**Highest** and **AboveNormal** are available only to administrators.)
Run time	Specifies the runtime limit of the job.
Run until canceled	Tells the job to reserve its resources until the job is canceled or the runtime limit is reached.
Exclusivity	Specifies whether the nodes are allocated to the job exclusively.
License	Specifies the license features and number of each required for the job to run.

Creating Jobs Using the Compute Cluster Job Console

The new Compute Cluster Job Console, shown in Figure 4, makes creating jobs and assigning tasks straightforward. The Compute Cluster Job Console is the standalone version of the Job Management Navigation Pane in the Administration Console, and has the same contents, and uses the same steps. Everything you can do in the Compute Cluster Job Console can also be done in the Job Management Navigation Pane of the Administration Console.

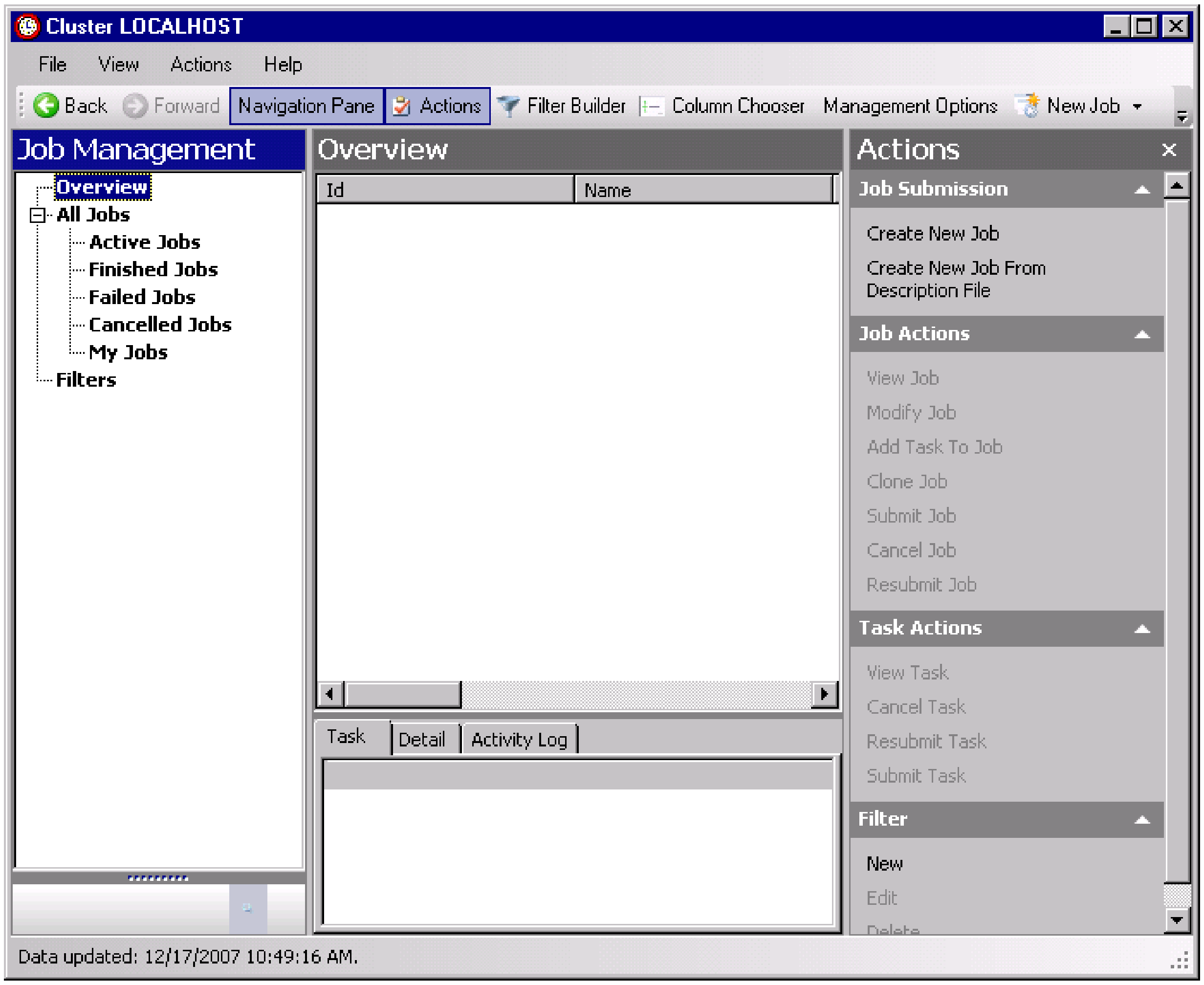
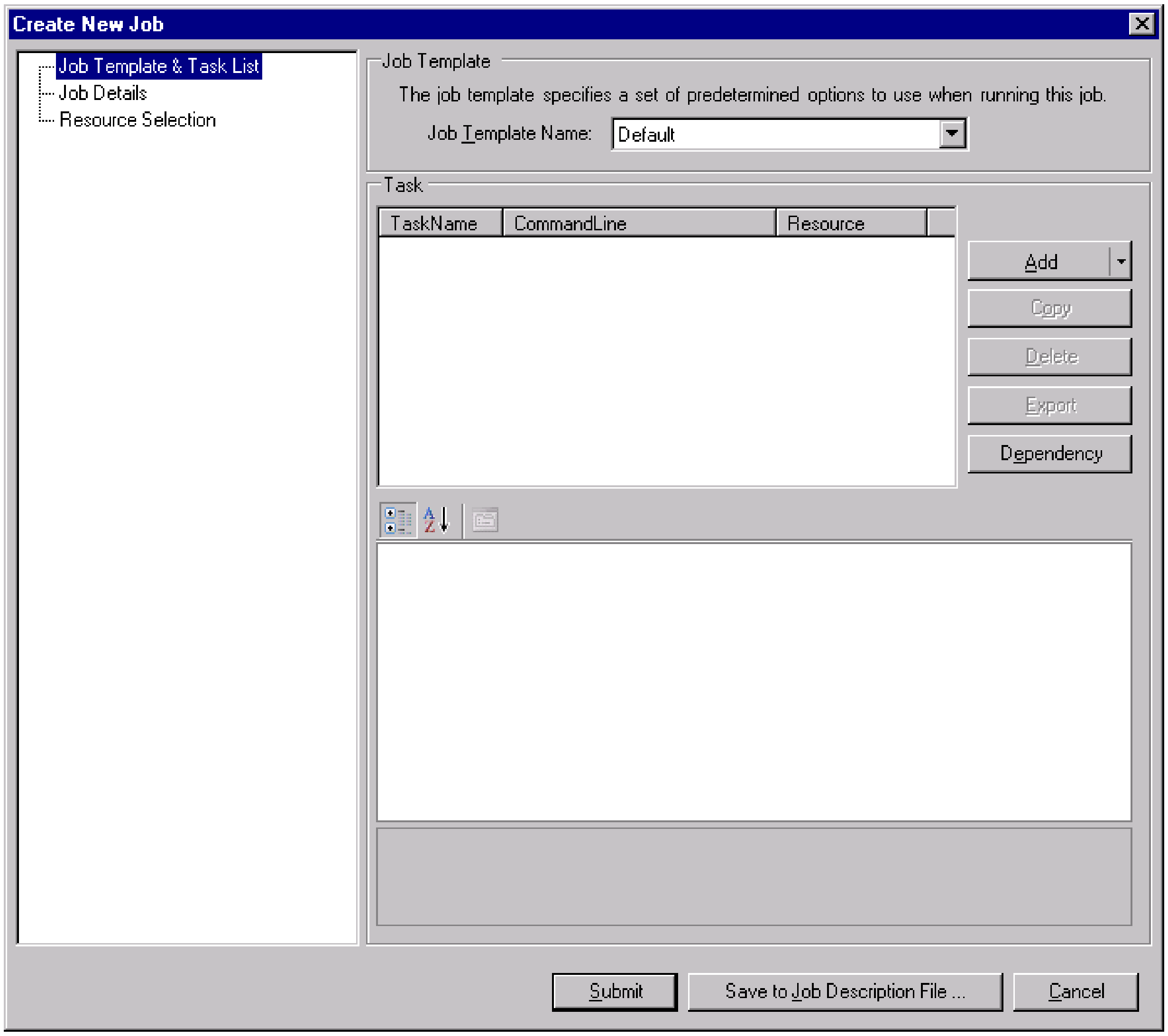


Figure 4 The Compute Cluster Job Console

To create a job in the Job Console, on the **Actions** menu, select **Create New Job** to open the **Create New Job Wizard** shown in Figure 5.



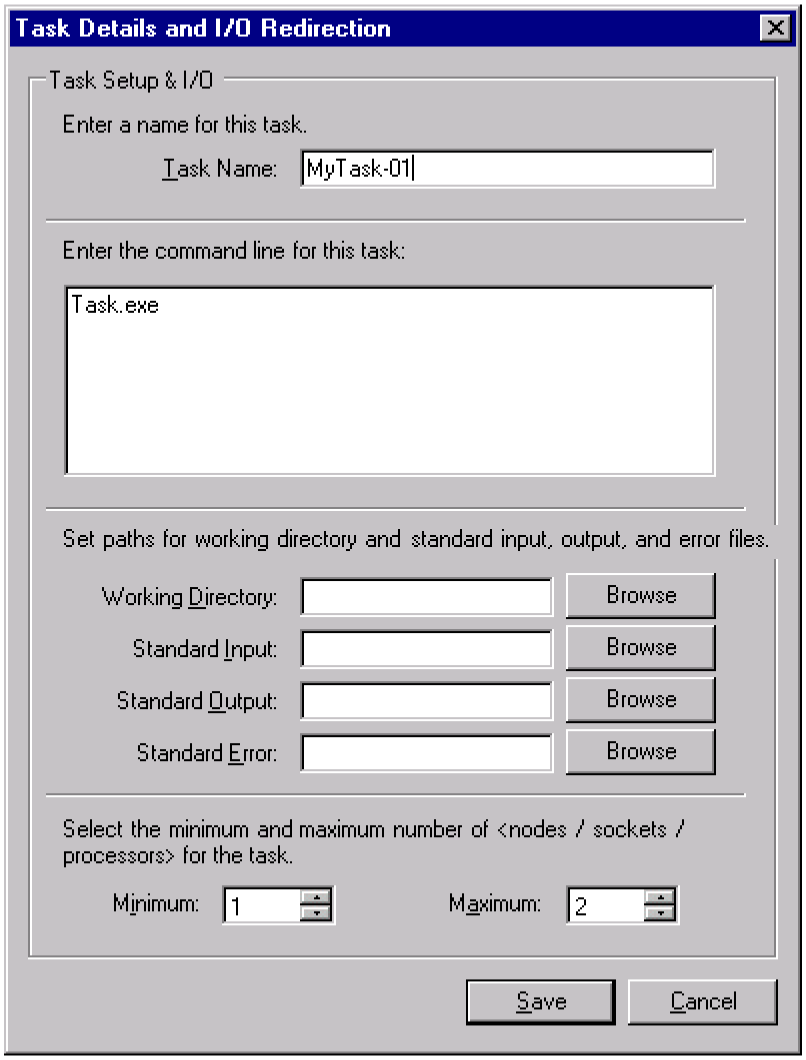
The job template specifies a set of predetermined options to use when running this job.

Job Template Name: Default

Figure 5 The Create New Job Wizard

Adding Tasks to a Job

Tasks are the discrete commands that jobs execute. You can add tasks to a job on the **Job Template** and **Task List** page of the **Create New Job Wizard**. Each task is named, but task names do not have to be unique within a job. Tasks consist of executables that run on cluster resources, so when a task is created, a command must be entered to tell the system which executable to run. If the task uses a MS-MPI executable, the **task** command must be preceded by **mpiexec**. Tasks can run executables directly or can consist of batch files or scripts performing multiple activities. To add a task to a job, click **Add** to open the **Task Details and I/O Redirection** page shown in Figure 6.


Task Details and I/O Redirection

Task Setup & I/O

Enter a name for this task.

Task Name: MyTask-01

Enter the command line for this task:

```
Task.exe
```

Set paths for working directory and standard input, output, and error files.

Working Directory:

Standard Input:

Standard Output:

Standard Error:

Select the minimum and maximum number of <nodes / sockets / processors> for the task.

Minimum: 1

Maximum: 2

Figure 6 The Task Details and I/O Redirection page

To create a task through the command-line interface, the user types the following command:

```
job add <jobID> [standard_task_options] [/f:<template_file>]
<command> [arguments]
```

where jobID is the number of the job, template_file is the file containing job commands, and command is additional actions for the job to perform.

To create a task using Windows PowerShell, use the following command:

```
    Add-Task [-id <jobID>] [options]
```

In addition to the job command, the command-line interface also supports a specific **task** command. Table 2 lists the operators available with the task command.

Table 2. Task Properties

Property	Description
Task name	Name of task.

Property	Description
Environment variables	Environment variables for the task, with name and value of each.
Working directory	Directory where the job looks for input files and writes output files.
Number of processors	Specifies the minimum and maximum number of compute processors to be reserved across a set of nodes.
Required nodes	Nodes that must be reserved for a task.
Exclusivity-nonexclusivity	Exclusive allocation of nodes for a task.
Dependency	Inter-task dependencies.
Runtime	Specifies the runtime limit of the task.
Rerunnable	Specifies that a task is to be rerun automatically after a failure when the failure is due to system error.
Checkpointable	Specifies that the task can have assigned checkpoints
Input file	Redirect standard input of the task from this file.
Output file	Redirect standard output of the task to this file.
Error file	Redirect standard error of the task to this file.

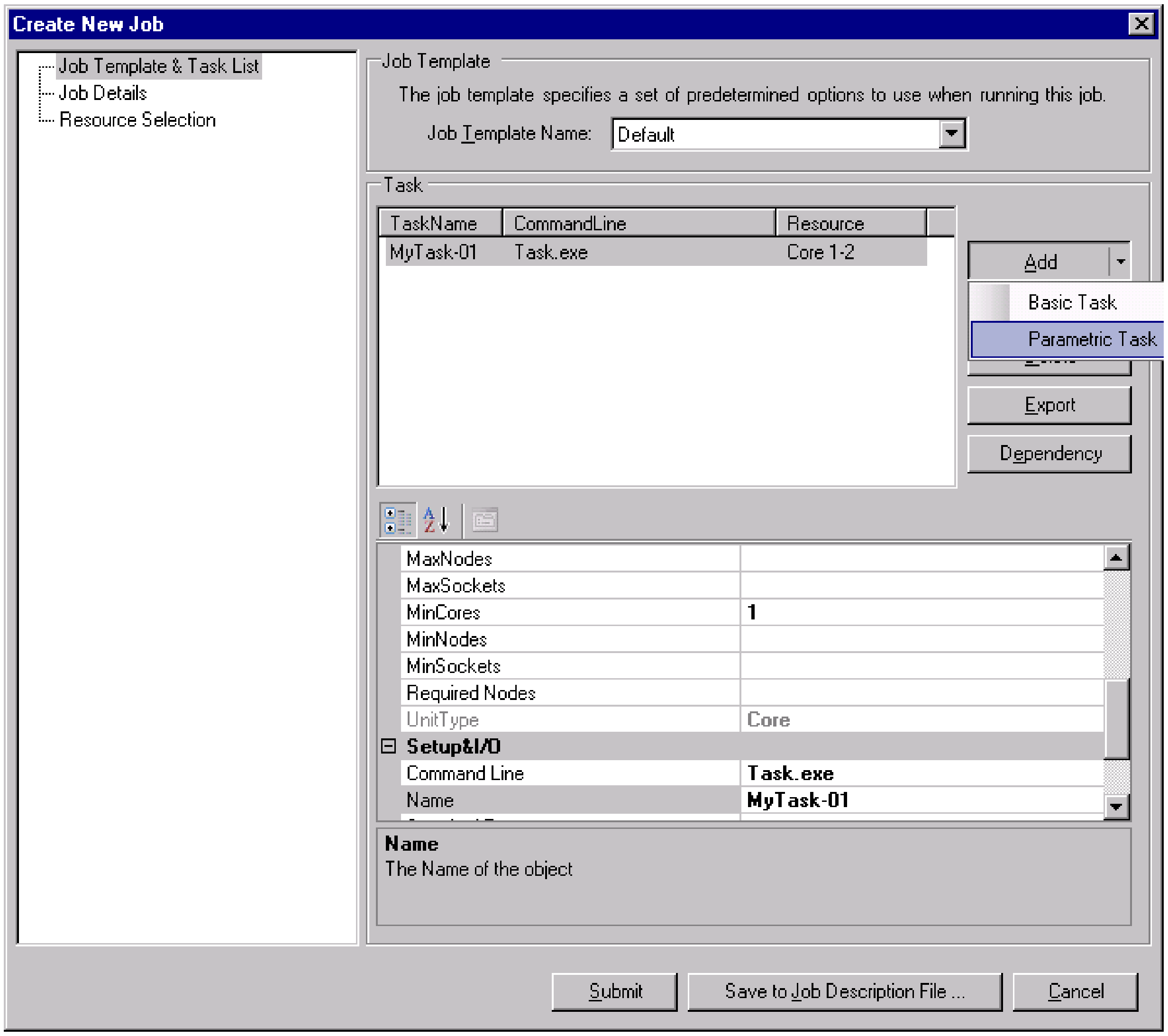
Setting how tasks access necessary data is an important factor in job submission for optimal performance of tasks. This setting should vary depending on the size and stability of the data set. If a data set does not change often, and is relatively large, it should be local to the tasks. If the data set is small, it can be accessed through a file share. If the data set is large and changes often, transfer the data to the nodes. Windows HPC Server supports parallel and high performance file systems to improve the access to very large datasets. Users of small and medium dataset sizes will have the best out-of-the-box experience by specifying the working directory. When the task starts, compute nodes see all the files in this working directory and can properly handle the task.

Jobs that must work with parallel tasks through MS-MPI require the use of the mpiexec command, so commands for parallel tasks must be in the following format:

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

where Myapp.exe is the name of the application to run. Users can submit MS-MPI jobs either through the Job Manager or the command line.

Windows HPC Server 2008 adds direct support for parametric tasks. To create a parametric task, click the down arrow on the **Add** button in the **Create New Job Wizard**, as shown in Figure 7, to open the **Parametric Task** page shown in Figure 8.



The job template specifies a set of predetermined options to use when running this job.

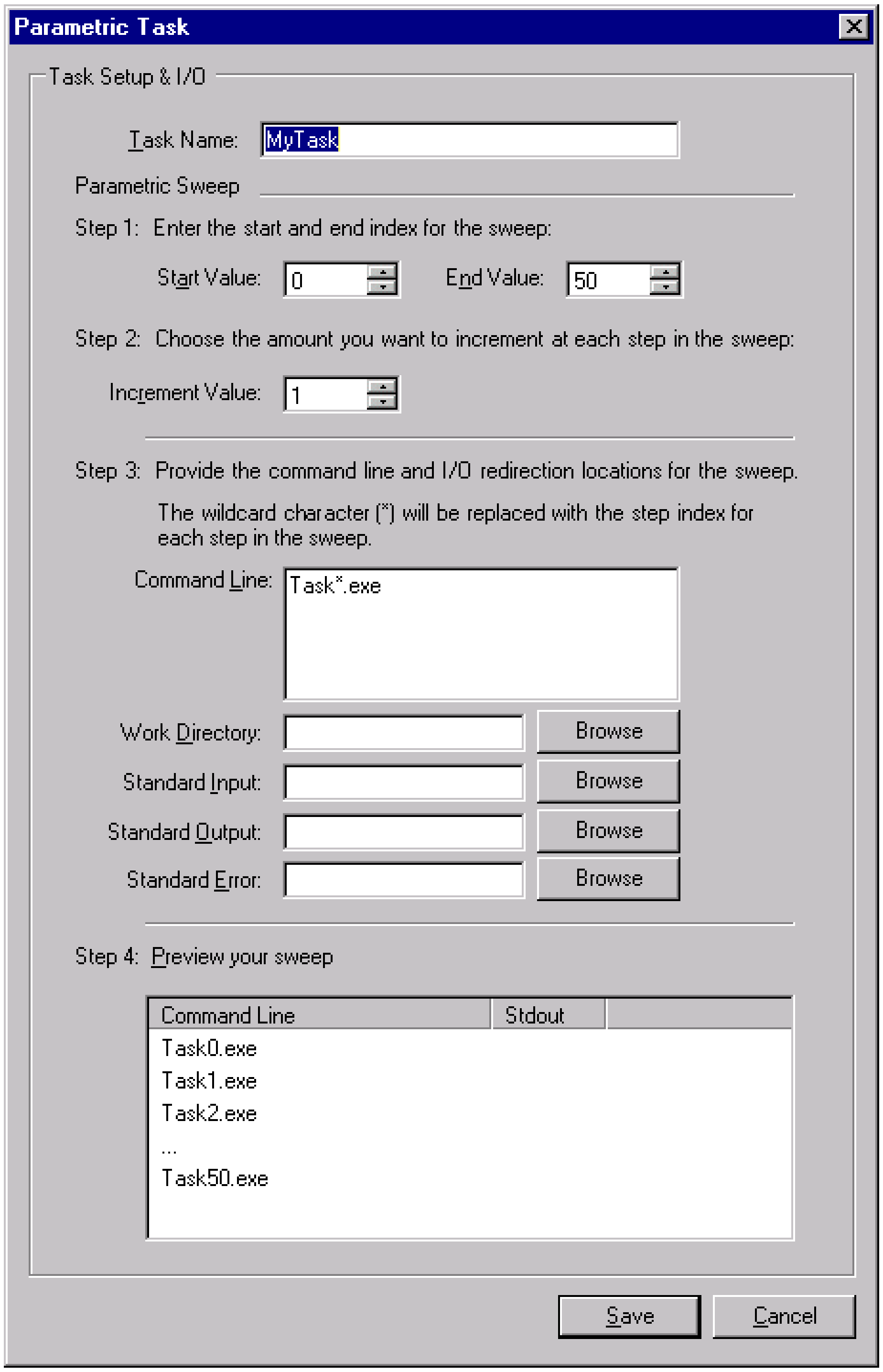
Job Template Name: Default

TaskName	CommandLine	Resource
MyTask-01	Task.exe	Core 1-2

MaxNodes	
MaxSockets	
MinCores	1
MinNodes	
MinSockets	
Required Nodes	
UnitType	Core
Setup&I/O	
Command Line	Task.exe
Name	MyTask-01

Name
The Name of the object

Figure 7 Adding a Parametric Task



Parametric Task

Task Setup & I/O

Task Name: MyTask

Parametric Sweep

Step 1: Enter the start and end index for the sweep:

Start Value: 0 End Value: 50

Step 2: Choose the amount you want to increment at each step in the sweep:

Increment Value: 1

Step 3: Provide the command line and I/O redirection locations for the sweep.

The wildcard character (*) will be replaced with the step index for each step in the sweep.

Command Line: Task*.exe

Work Directory:

Standard Input:

Standard Output:

Standard Error:

Step 4: Preview your sweep

Command Line	Stdout
Task0.exe	
Task1.exe	
Task2.exe	
...	
Task50.exe	

Figure 8 The Parametric Task Wizard

In addition to the **job** and **task** commands, users can call on the **cluscfg** command, which provides access to information about the cluster itself. A complete list of the options available for the **job**, **task**, and **cluscfg** commands appears in Table 3.

Table 3. Windows HPC Server 2008 Command-Line Commands

Command	Operators	Description
job	job new [job_terms]	Create a job container.
	job add jobID [task_terms]	Add tasks to a job.
	job submit /id:jobid	Submit a job created through the **job new** command.
	job submit [job_terms][task_terms]	Submit a job.
	job cancel jobID	Cancel a job.
	job modify [options]	Modify a job.
	job requeue JobID	Re-queue a job.
	job list	List jobs in the cluster.
	job listtasks	List the tasks of a job.
	job view JobID	View details of a job.
task	task view taskID	View details of a task.
	task cancel taskID	Cancel a task.
	task requeue taskID	Re-queue a task.
cluscfg	cluscfg view	View details of a cluster.
	cluscfg params/setparams	View or set configuration parameters.
	cluscfg listenvs/setenv	List or set cluster-wide environment.
	cluscfg delcreds/setcreds	Set or delete user credentials.

Working with Job Files and Submitting Jobs

Jobs can be saved as Job Description files by clicking **Save To Job Descriptor File** in the **Create New Job Wizard**. Job Description files can be exchanged in an XML format that allows for their reuse, editing and generation 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 through the command-prompt window.

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

Jobs can also be submitted through simple command-line commands—for example, the command:

```
job submit /numprocessors:8 mpiexec mympitask.exe
```

In this command, Mympitask.exe is the name of the submitted application, which submits an MPI job requiring eight processors.

Using the Job Scheduler C# API:

The Windows® HPC Pack v1 API provides access to the Job Scheduler. By writing applications or scripts using these interfaces, you can connect to a cluster and manage jobs, job resources, tasks, nodes, environment variables, extended job terms and more.

Using HPC Pack v1 API, in its simplest terms, is a five-step process:

1. Connect to the cluster.
2. Create a job.
3. Create a task.
4. Add the task to the job.
5. Submit the job/task for execution.

To connect to the cluster use the **ICluster::Connect** method. Create a job using the **ICluster::CreateJob** method. To create a task, use the **ICluster::CreateTask** method. To add a child task to a job, use the **ICluster::AddTask** method. Submit the job using **ICluster::SubmitJob()**.

The sample code below shows an example of how the Windows HPC Server Pack API can be used following the five step process outlined above.

```
using System;
using System.Collections.Generic;
using System.Text;
using Microsoft.ComputeCluster;

namespace SerialJobSub
{
    class Program
    {
        static int Main(string[] args)
        {
            Cluster cluster = new Cluster();

            try
            {
                cluster.Connect("myheadnode");

                IJob job = cluster.CreateJob();
                ITask task = new Task();
                task.CommandLine = @"c:\myprog.exe arg1 arg2";
                task.Stdout = @"c:\pi.out";
                job.AddTask(task);

                int jobId = cluster.AddJob(job);
                cluster.SubmitJob(jobId, @"mydomain\myuserid", null, true, 0);
                Console.WriteLine("Job " + jobId + " submitted successfully");
                return 0;
            }
            catch (Exception ex)
            {
                Console.WriteLine(ex.Message);
                return 1;
            }
        }
    }
}
```

Using the HPC Basic Profile Web Service

The High Performance Computing Basic Profile (HPCBP) specification has been produced by the Open Grid Forum (OGF) (http://www.ogf.org). The HPCBP specification is used to define a Web services-based interface to securely submit, manage, and monitor jobs. It was developed from use cases that submitted computationally intensive jobs to a specific cluster within a user's own organization. The HPCBP provides a framework upon which the community can prototype and build extensions to develop the profile to support other use cases.

Windows HPC Server 2008 provides an implementation of an HPCBP compatible Web service that is deployed on the head node and can be activated by the cluster administrator. The detailed specification can be found at http://www.ogf.org/documents/GFD.114.pdf. It can be used to generate a Web service client on any platform that supports a complaint Web service environment, for example, Java, C, C++, Python, etc.

This following sample source code encapsulates the generation of the messages to the HPCBP Web service and the conversion of the responses to C# objects. It submits a job to the HPCBP Web service that is described by the Job Submission Description Language (JSDL) and XML document schema, and then waits for the job to be executed on the specified cluster.

```
    public class HPCBPClient
    {
        static int main(string[] args)
        {
            if (args.Length != 4) {
                Console.WriteLine("HPCBPClient [service URL] [JSDL job file] [username] 
[password]");
                return 1;
            }

            string serviceUrl = args[0];
            string jsdlFileName = args[1];
            string userName = args[2];
            string password = args[3];

            HPCBPClient hpcbp = new HPCBPClient(serviceUrl, userName, password);

            EndpointReferenceType[] eprs = new EndpointReferenceType[1];
            eprs[0] = hpcbp.CreateActivity(jsdlFileName);
            GetActivityStatusResponseType[] status = null;
            do { // Loop polling the service to see if the job is over
                Thread.Sleep(5000);
                status = hpcbp.GetActivityStatuses(eprs);
            } while (status[0].ActivityStatus.state == ActivityStateEnumeration.Pending ||
             status[0].ActivityStatus.state == ActivityStateEnumeration.Running);

            return 0;
        }
```

Job Scheduling

Job Scheduling allocates jobs – both the ordering within the queue and resource allocation for the execution of tasks. Both are controlled through scheduling policies. Both are controlled through scheduling policies. Windows HPC Server supports nine policies, with each focusing on a specific scheduling issue:

- **Priority-based FCFS.** This scheduling policy combines FCFS and priorities to run jobs. Jobs are placed in based on priority and submission time. All jobs are in the queue on a first-in, first-out basis, *within the priority order of the jobs.* So all Highest Priority jobs are ahead of all Above Normal Priority jobs, and so on.
- **Backfilling.** The user provides the Job Scheduler with the time and resources required to execute a job. Resources are reserved based on the job submission. If the Job Scheduler identifies open windows for resources within the reserved timelines, it can backfill by selecting smaller jobs for execution within this window, increasing utilization of the cluster by allowing smaller jobs to jump the queue when there are available resources.
- **Nonexclusive schedules.** When a job is “Exclusive,” no other job can run on a node with that job. When a task is “Exclusive,” no other task can run on a node with that task.
- **Resource Matchmaking.** The user specifies compute, networking, and application resource requirements, and the Job Scheduler performs right-sizing placement for the job.
- **Job Templates.** The administrator creates job templates that define the resources needed for specific processing needs and the priorities of multiple user groups.
- **Multilevel Compute Resource Allocation.** The Job Scheduler optimally places memory-intensive jobs to avoid contention of memory, delivering maximum and predictable application performance.
- **Preemption.** (Beta 2) High priority jobs take resources away from already running, lower priority jobs.
- **Grow/Shrink Job Scheduling Policy.** Jobs can be dynamically sized over time to decrease queue wait time and job run time. Jobs might start at a lower resource allocation, but this allocation can be increased over time if the job can take advantage of additional resources, and then resources can be removed as and when the job has no need for them.

The default resource allocation strategy processes the following terms (see Table 1):

- numprocessors
- askednodes
- nonexclusive
- exclusive
- requirednodes

By default, the Job Scheduler sorts the candidate nodes using the “fastest nodes with the largest memory first” criterion—that is, it sorts the nodes according to memory size first, and then sorts the nodes by their speed. This behavior is the default for all the resource-allocation strategies, but it can be customized on a per job basis. Next, the scheduler allocates the CPUs from the sorted nodes to satisfy the minimum and maximum processor requirements for the job. The Job Scheduler attempts to satisfy the maximum number of CPUs for the job before considering another job.

If the **askednodes** term is specified, the Job Scheduler does not sort the nodes in the node list but allocates the processors from the requested list in the user-specified order until the minimum and maximum number of processor requirement is met.

Proper configuration of the jobs in the cluster leads to the best use of the cluster by balancing the allocation of resources and the requirements of each job. In particular, the use of backfill windows and adaptive allocation (Grow/Shrink), along with default user job templates can simplify the effective use of the cluster. To facilitate the optimal use of resources when working with Windows HPC Server, users should follow these guidelines to create, submit, and run jobs:

1. Do not use the default **infinite** runtime when submitting jobs. Instead, define a runtime that is an estimate of how long the job should actually take.
2. Reserve specific nodes only if the job requires the special resources available on those nodes.
3. When specifying the ideal number of processors for a job, set the number as the maximum, and then set the lowest acceptable number as the minimum. If the ideal number is specified as a minimum, the job may have to wait longer for the processors to be freed so it can run.
4. Reserve the appropriate number of nodes for each task to run. If two nodes are reserved and the task requires only one, the task has to wait until the two reserved nodes are free before it can run.
5. Only set a job or task to run **Exclusive** if the job or task requires it.

These guidelines will help you make the greatest use of the resources available on the Windows HPC Server cluster.

Job Execution

Job execution starts the jobs, which run in the context of the submitting user, limiting the possibility of runaway processes. Jobs can also be automatically re-queued upon failure. Tasks are managed through their state transition. States for tasks are illustrated in Figure 9.

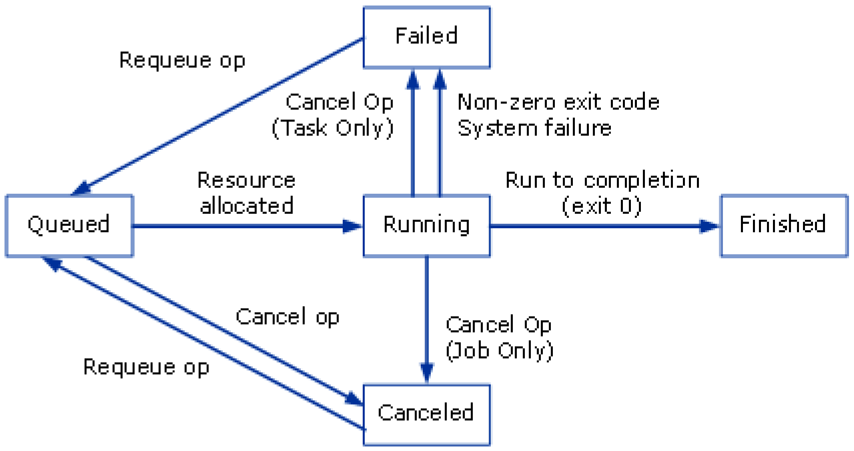


Figure 9. Task state transition

Like tasks, jobs and nodes also have life cycles represented by status flags displayed in the Administration Console. The status flags for jobs and tasks are identical, but nodes have unique status flags. Status flags are listed in Table 4.

Table 4. Windows HPC Server Status Flags

Item	Flag	Description
Jobs and tasks	CONFIGURING	Job or task has been created but not submitted
	QUEUED	Job or task is submitted and is awaiting activation.
	RUNNING	Job or task is running.
	FINISHED	Job or task has finished successfully.
	FAILED	Job or task has failed to finish successfully.
	CANCELLED	The user cancelled the job or task.
Nodes	ONLINE	The node is accepting jobs
	OFFLINE	The node is not accepting jobs.
	UNKNOWN	The node state is unknown. The node is not a part of the cluster.

Item	Flag	Description
	PROVISIONING	The node is being configured as a compute node, including HPC setup and creation of an Active Directory account for the node, and the steps in its Node Template are being applied.
	STARTING	The node is transitioning from offline to online
	DRAINING	The node is not accepting new jobs, and will finish existing jobs.
	REMOVING	The node is being removed from the cluster.
	REJECTED	The node has been rejected for inclusion in the cluster.

Controlling Jobs Through Filters

Windows HPC Server includes several features for job submission and execution. Jobs can consist of simple tasks or can include comprehensive feature sets. Administrators can control jobs through 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 runtimes 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 ensures that the user's time allocations are not exceeded. Unlike the mandatory policy, this filter limits jobs to the overall time allocations users have for all possible jobs.

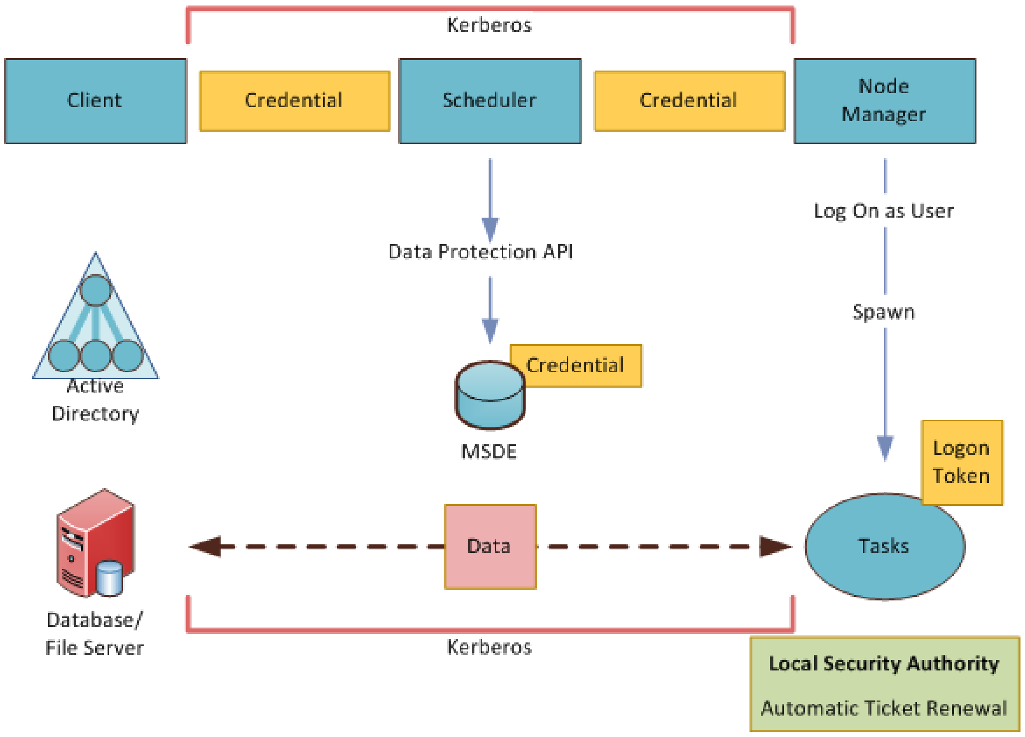
In addition, job execution filters can ensure that jobs meet licensing conditions before they run. Filters are a powerful job-control feature that should be part of every Windows HPC Server implementation. The Job Manager invokes the filters by parsing the job file, which contains all the job properties. The exit code of the filter program tells the 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.

Security Considerations for Jobs and Tasks

Windows HPC Server 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 the Job Manager, 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 through a secure Microsoft® .NET Remoting channel, then secured through 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 the Job Manager. This feature simplifies resubmission and provides integrated security for credentials throughout the job life cycle, as shown in Figure 10.

Figure 10 The end-to-end Windows HPC Server security model.

The Service Oriented Application (SOA) Programming Model and Runtime

High-performance computing applications use a cluster to solve a single computational problem or a single set of closely related computational problems. These applications are classified as either message intensive or embarrassingly parallel. An embarrassingly parallel problem is one for which no particular effort is needed to divide the problem into a very large number of parallel tasks, and there is no dependency or communication among those parallel tasks.

This section covers building, deploying, running, and managing interactive HPC applications that are embarrassingly parallel. Table 5 shows some example applications and tasks

Table 5. Examples of SOA Applications

Example Application	**Example Task**
Monte Carlo problems that simulate the behavior of various mathematical or physical systems. Monte Carlo methods are used in physics, physical chemistry, economics, and related fields.	Predicting the price of a financial instrument.
BLAST searches	Gene matching.
Genetic algorithms	Evolutionary computational meta-heuristics.
Ray Tracing	Computational physics and rendering.
Digital Content Creation	Rendering frames.
Microsoft Excel add-in calculations	Calling add-in functions.

The Problem

To solve embarrassingly parallel problems, developers need to encapsulate the core calculations as a software module. The module can be deployed and run on the cluster, can identify and marshal the data required for each calculation, and can optimize performance by minimizing the data movement and communication overhead.

With the increasing number and size of the problems being tackled on ever larger clusters, developers, end users, and administrators are facing increasing challenges in meeting time-to-result goals. Applications must be quickly developed, run efficiently on the cluster, and be effectively managed so that application performance, reliability, and resource use are guaranteed.

The Solution

Microsoft Windows HPC Server 2008 delivers a scalable, reliable, and secure interactive application platform. New cluster-enabled interactive applications can be rapidly developed and existing applications can be easily modified. The developer experience of build/debug/deploy is streamlined, the speed of processing is greatly accelerated compared to traditional cluster batch jobs, and the management of the applications and systems is simplified by the new Administration Console.

The Benefits

Table 6 details Windows HPC Server 2008 features and benefits for SOA applications.

Table 6.Benefits of Windows HPC Server 2008 for SOA Applications

Tasks	User Needs	Windows HPC Server 2008 Features
Build	Solve embarrassingly parallel problems without having to write the low-level code.	A service-oriented programming model based on WCF that effectively hides the details for data serialization and distributed computing.
	An IDE tool that allows developers to develop, deploy, and debug applications on a cluster.	Microsoft® Visual Studio® 2008 provides the tools to debug services and clients.
Deploy	Deploy line of business (LOB) applications easily and reliably.	Windows HPC Server enables administrators to deploy services to multiple nodes from a single point of control.
Run	Shorter application runtimes.	Low latency round-trip.
	User applications must run securely.	Windows HPC Server provides end-to-end Kerberos with WCF transport-level security.
	A system that decides where to run the tasks of the application and dynamically adjusts cluster resource allocation to the processing priorities of the workload.	Dynamic allocation of resources to the service instances.
Manage	Monitor application performance from a single point of control.	Runtime monitoring of performance counters, including the number and status of outstanding service calls, and resource usage.
	Monitor and report service usage.	Service resource usage reports.

The Programming Model

The key abstraction of the programming model is that a client creates a session with the Job Scheduler, which allocates a pool of service instances on the compute nodes as workers for the session. Then the client invokes the methods exposed by the service instances.

The namespace for the Session API is *Microsoft.ComputeCluster.Scheduler.Session*. It contains two key classes, *Session* and *SessionStartInfo*. This namespace is described in Table 7.

Table 7. The Microsoft.ComputeCluster.Scheduler.Session Namespace

Class	Description
Session	Session enables the client code to create a virtual pool of service instances for a given service and distribute the calculations over multiple service instances to accelerate processing speed.
SessionStartInfo	SessionStartInfo specifies a set of values used when creating a session. SessionStartInfo offers most common controls over the session you start.

For more details on the Session API, consult the Microsoft Compute Cluster SDK Documentation.

The following example shows how to create a session, bind the session to the client proxy, and process the results. (The service that this client calls is shown after this example.)

```
using System;
using System.Collections.Generic;
using System.Text;
using System.ServiceModel;
using System.Threading;
using Microsoft.ComputeCluster.Scheduler.Session;

namespace EchoClient
{
    class Program
    {
        static void Main(string[] args)
        {
            string scheduler = "localhost";
            string serviceName = "EchoSvc";

            if (args.Length > 1)
            {
                scheduler = args[0];
                if (args.Length > 2)
                {
                    serviceName = args[1];
                }
            }

            // Create a session object that specifies the head node
            // to which to connect
            // and the name of the WCF service to use.
            // This example uses the default start information for a
            // session.
            SessionStartInfo info = new SessionStartInfo(scheduler, serviceName);

            // Creates a session.
            using (Session session = Session.CreateSession(info))
{
                // Binds session to the client proxy using NetTcp
                // binding (specify only NetTcp binding). The
                // security mode must be Transport and you cannot
                // enable reliable sessions.

                EchoSvcClient client = new EchoSvcClient(new NetTcpBinding(SecurityMode.Transport,
   false), session.EndpointReference);
                AsyncResultCount = 100;
                for (int i = 0; i < 100; i++)
                {
                    client.BeginEcho("Hello, World!", EchoCallback, new RequestState(client, i));
                }

                AsyncResultsDone.WaitOne();
                client.Close();
            }
        }

        static int AsyncResultCount = 0;
        static AutoResetEvent AsyncResultsDone = new AutoResetEvent(false);
        class RequestState
        {
            int input;
            EchoSvcClient client;
            public RequestState(EchoSvcClient client, int input)
            {
                this.client = client;
                this.input = input;
            }
            public int Input

            {
                get {return input;}
            }
            public string GetResult(IAsyncResult result)
            {
                return client.EndEcho(result);
            }
        }

        static void EchoCallback(IAsyncResult result)
        {
            RequestState state = result.AsyncState as RequestState;
            Console.WriteLine("Response({0}) = {1}", state.Input, state.GetResult(result));
            if (Interlocked.Decrement(ref AsyncResultCount) <= 0)
            {
AsyncResultsDone.Set();
            }
        }
    }
}
```

The following example shows the interface definition for the service object.

```
using System;
using System.Collections.Generic;
using System.Text;
using System.ServiceModel;

namespace EchoSvcLib
{
    [ServiceContract]
    public interface IEchoSvc
    {
        [OperationContract]
        string Echo(string input);
    }
}
```

Building the applications using the programming model is easy. The model is service-oriented, allowing the developer to write the service program and client program using the widely adopted WCF platform. Visual Studio provides easy-to-use WCF service templates and service referencing utilities that enable the developer to quickly prototype, debug, and unit-test their applications.

Architecture Overview

Figure 11 shows the underlying architecture for supporting the SOA programming model.

1. The client creates a session
2. The Job Scheduler allocates nodes, and launches the service instances, which load the service .dll files. The Job Scheduler allocates a Broker node to start the WCF Broker job. The Job Scheduler uses the round-robin strategy when selecting a Broker node. At startup, the router job publishes its Endpoint Reference by setting the job's EndpointReference property.
3. The client queries the job's EndpointReference property. The client connects to the WCF Broker and sends requests that are then load-balanced across the multiple service instances.

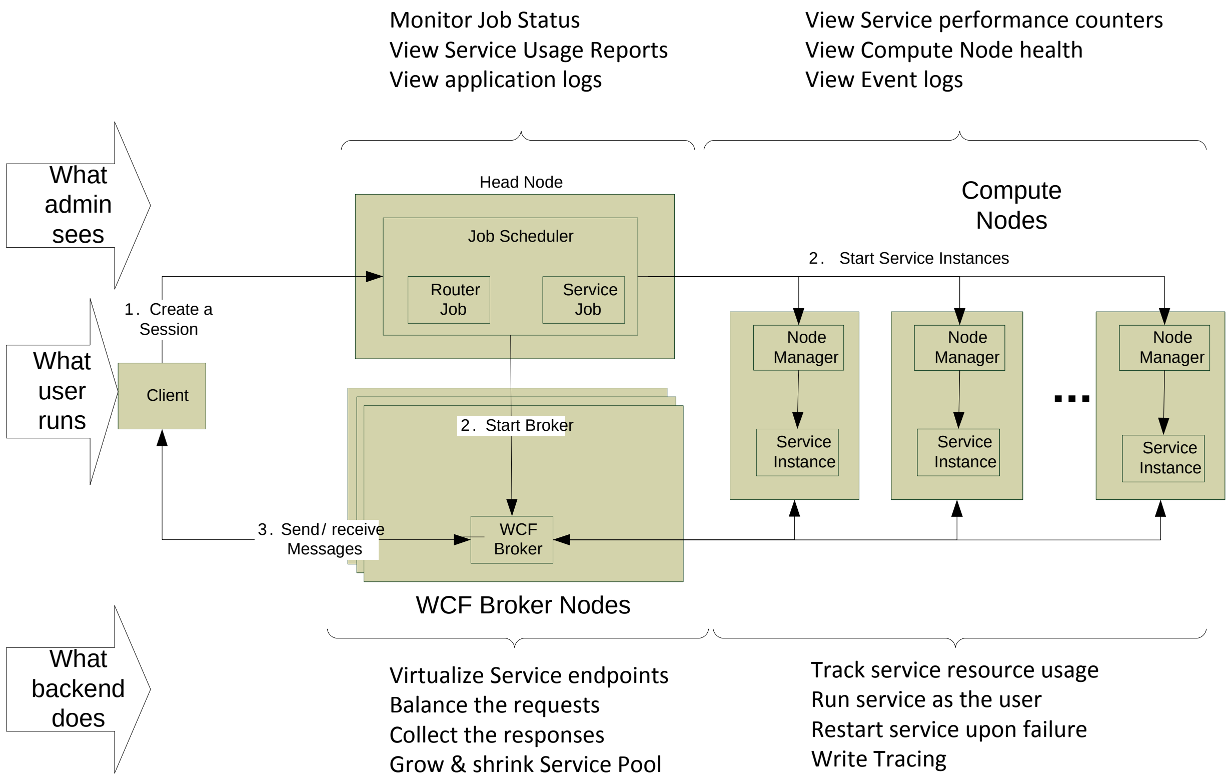


Figure 11 Architecture Overview

The number of service instances can change according to the dynamic workload of the cluster. While the job is running, the administrator can use the Windows HPC Server 2008 Administrator Console to monitor the heat map of the router and compute nodes, and the Job Manager to monitor the progress and resource usage of the session job. The resource usage of services is logged so that usage reports based on users, projects, or service names can be created.

Service Instance Resourcing Model

The Service Instance Resourcing model defines how service instances are mapped to computing resources. There are three types of models, as shown in Table 8.

Table 8. Service Instance Resource Models

Resourcing Model	Description
One service process per processor	Used to host services that are linked with non-thread safe libraries.
One service process per node	Multithreaded services.
One service process per socket	Single threaded services that are memory-bus intensive.

Table 9 shows the details for each of the resourcing models.

Table 9. Resource Model Details

Resourcing Model	**Resource Allocation Unit Type**	**WCF Service Instance Configuration**	**Example**
One service process per processor	SessionStartInfo.ResourceUnitType = Core	Singleton	C++ Analytics services in capital market firms
One service process per node	SessionStartInfo.ResourceUnitType = Node	Per call	Service code that uses multiple processors on a given node
One service process per socket	SessionStartInfo.ResourceUnitType = Socket	Per call	Solver services

The Runtime System

The following steps describe the runtime behavior of this code:

1. Clients create a session by specifying the WCF service to be used to perform the calculation.
2. The Windows HPC Server Job Scheduler assigns a WCF Broker and starts a service instance pool on multiple nodes.
3. The client connects to the WCF broker:
   a. The Windows HPC Server Job Scheduler provides the EndPoint Reference (EPR) of the router to client.
   b. The client connects to the EPR.
   c. Standard WCF request/response messages are used in the subsequent calculations.

Configuring WCF Broker Nodes

To run SOA jobs, you need to configure a WCF Broker node. Broker nodes are the nodes on which service router program runs. Broker nodes must have the same network connectivity as the head nodes.

The following procedure describes how to configure a WCF Broker node:

4. Click **Node Management** in the Navigation pane.
5. Navigate to **HeadNodes**, and then select **By Group**.
6. On the Actions pane, select **Take offline**.
7. In the Results pane, verify that the state of the node changes to **Offline**.
8. In the Actions pane, select **Change Role**.

9. In the **Change Additional Role** dialog box, click **Select WCF Broker node**, and then click **OK**.
10. In the Actions pane, click **Take online**.

In Step 6 of this procedure, Windows HPC Server checks whether the node has a network interface to a public network. If not, this procedure will fail and an error message will be displayed in the Results pane. The error message will also be logged in the Windows HPC Server event log.

Application Deployment

To deploy the WCF service code in the previous section, you must register the service DLL on each node in the cluster. To register the service DLL, create a configuration file named <servicename>.config (for example, EchoSvc.config) that contains the following XML code:

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

<configuration>

    <appSettings>

        <add key="assembly" value="c:\Services\EchoSvcLib.dll"/>

        <add key="type" value="EchoSvcLib.EchoSvc"/>

        <add key="contract" value="EchoSvcLib.IEchoSvc"/>

        </appSettings>

</configuration>
```

The <add> statements that specify the type and contract key attributes are optional if the service defines only one interface; otherwise, you must specify these attributes for each interface that the service defines.

After you define the configuration file, you must copy the configuration file to **%CCP_HOME%\ServiceRegistry** on all compute nodes in the cluster.

The <add> statement with the assembly key attribute specifies where the service DLL is located on each compute node. You must copy the service DLL to the specified path on all compute nodes in the cluster.

Monitoring a Session

When an SOA client creates a session, he Session API creates two jobs: a WCF Broker job and a Service Job. The WCF Broker job can only be started on the Broker nodes, and the Service Job can only be started on the Compute Nodes, as shown in Table 10.

Table 10. SOA Session

Jobs	Command Line	Execution Nodes
WCF Broker Job: one task	CcpWSLB -servicejobid <jobId> -servicename <servicename>	Broker nodes
Service Job: as many tasks as there are allocated units	CcpServiceHost -servicename <servicename> -LB "net.tcp://<WCFBrokerNode>:9089/LBRegistration/<jobId>/1"	Compute Nodes

If session uses the Core as the job scheduling type, then there will be as many services as there are cores allocated to the Service Job. See the Service Instance Resource Model for details.

To monitor the number of calls going through the WCF Broker, select the WCF Broker Job, and double-click the job. In the **Job Properties** dialog box, on the **Statistics** tab, the WCF Router performance counters, shown in Table 11, are displayed.

Table 11. WCF Router Performance Counters

Fields	Description
Number of Calls	Total number of calls
Number of Calls Outstanding	Number of in-progress calls to this service
Call duration	Average length of the call in seconds
Calls Per Second	Number of calls per second

Troubleshooting and Diagnosing SOA Application Runtime Errors

Some typical error conditions, their causes, and the actions to take to resolve them are described in Table 12.

Table 12. Error Conditions

Error conditions	Possible Causes	Actions
WCF Broker fails to start	The Broker node has not been configured. Start the Administration Console, navigate to the Node Management pane, and under By Group, click RouterNode. If there are no Broker nodes, then the router can't be started	Change the Head Node's role to Broker node (see Configure Broker node section).
Service Job has failed tasks	Service DLL has not been deployed to the Compute nodes <servicename>.config file does not exist in the %ccp_home%\ServiceRegistry folder The path to the DLL in the <servicename>.config is not correct	Identify the root cause and take required actions.

Summary

Windows HPC Server 2008 includes a new and more scalable Job Scheduler that now supports advanced policies, and a new Service Oriented Architecture mode that allows interactive applications and that provides a streamlined developer experience of build/debug/deploy. Support for Windows Server 2008 Failover Clustering increases the reliability and availability of the HPC cluster. The graphical interface for the Job Scheduler is now fully integrated into the Administration Console, and the command-line interface now supports Windows PowerShell for all Job Scheduler functions.