Windows®
HPC Server 2008

Using Windows HPC Server 2008 Job Scheduler

Microsoft Corporation

Published: June 2008, Revised September 2008

Abstract

Windows® HPC Server 2008 includes a new Job Scheduler that provides greater scalability and supports advanced policies. The new Job Scheduler supports a new Service-Oriented Architecture (SOA) mode that provides access to interactive applications through the Windows Communication Foundation (WCF). The graphical interface for the Job Scheduler is now fully integrated into the Administration Console, and the Command Line Interface (CLI) now uses Windows PowerShell™ for all Job Scheduler functions. In addition, core Job Scheduler functionality is exposed using the Open Grid Forum's HPC Basic Profile Web service. This paper explains new Job Scheduler functionality.

Microsoft®

Contents

Windows HPC Server 2008 Operations Overview

Windows® HPC Server 2008 brings high performance computing (HPC) to industry standard, low-cost servers that support larger and heterogeneous clusters. Jobs—discrete activities scheduled to perform on the HPC cluster—are the key to operating Windows HPC Server 2008. 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 2008 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 System Management Overview.”

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.

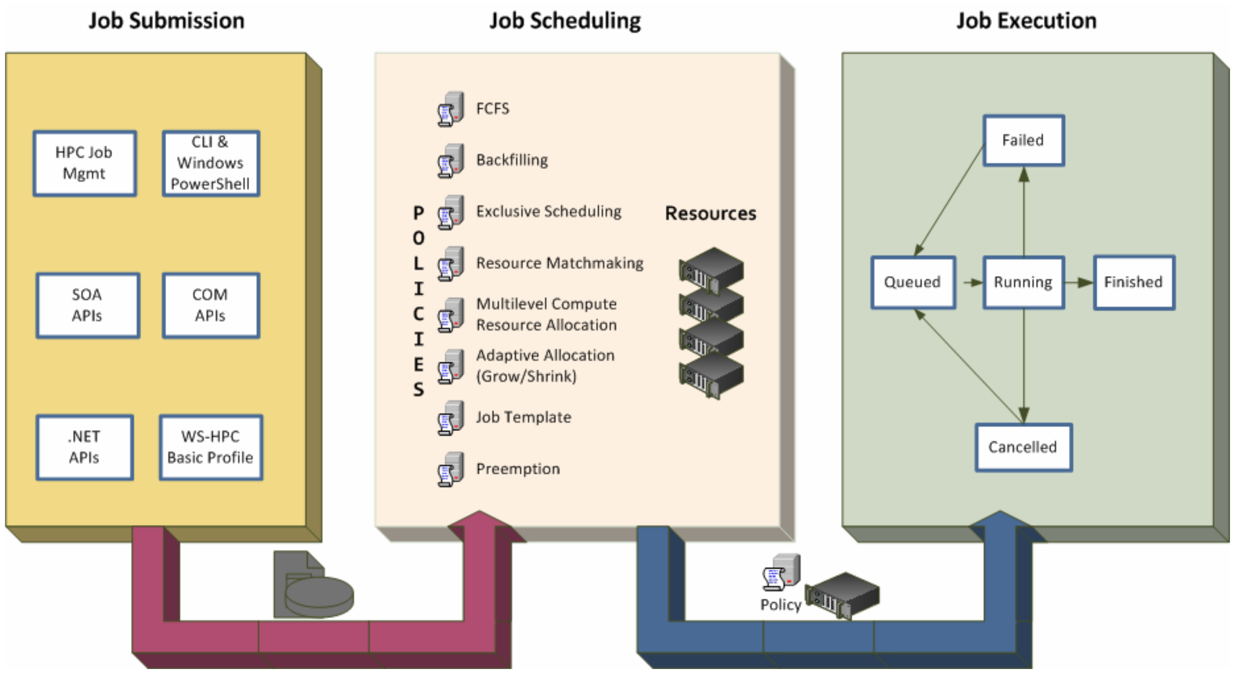


Figure 1. The HPC job life cycle

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

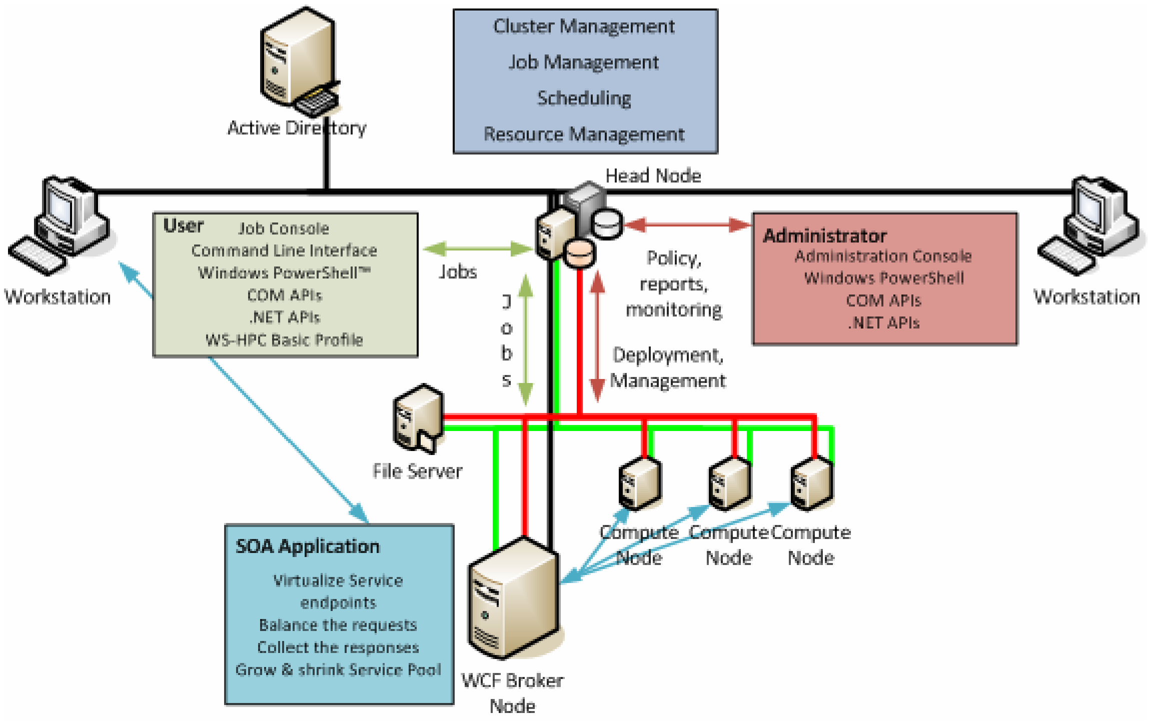


Figure 2. The elements of a compute cluster

A basic cluster consists of a single head node (or a primary and secondary head node if the deployed cluster uses Windows Server® 2008 Failover Clustering) 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 2008 provides transparent execution, access to data, and integrated security.

Windows HPC Server 2008 Terminology

Although compute clusters of all types share some terminology, Windows HPC Server 2008 has some specific terminology that's useful to define.

Cluster

A cluster is the top-level unit of Windows HPC Server 2008. A cluster contains the following elements:

- **Node.** A single physical or logical computer with one or more processors. Nodes can be head nodes, compute nodes, or WCF Broker nodes.
- **Queue.** An element providing queuing and job scheduling. Each Windows HPC Server 2008 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.
- **Task.** 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 multithreading, Message Passing Interface [MPI], or OpenMP).

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.

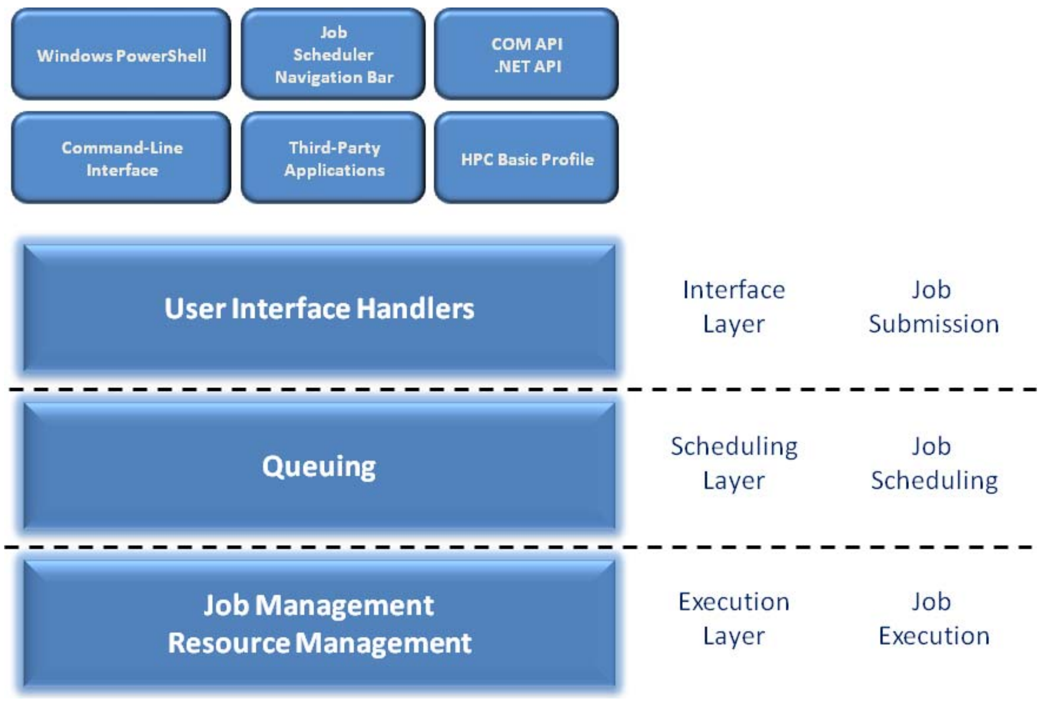


Figure 3. The Job Scheduler stack

- 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, in addition to application-specific launching and recovery mechanisms.

Administration Console

The Administration Console is the interface for cluster administration. Based on the Microsoft System Center user interface, Administration Console uses navigation bars to quickly change the context and view. The Job Manager navigation bar provides a graphical interface for job management and scheduling.

Scheduling Policies

Windows HPC Server 2008 uses nine scheduling policies, which are explained in more detail later in this paper:

- Priority-based first come, first served (FCFS)
- Backfilling
- Exclusive scheduling
- Resource matchmaking
- Job templates
- Multi-Level Compute Resource Allocation (MCRA)
- Preemption
- Adaptive (grow/shrink) allocation

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 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 (SOA)

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-I Web Services, architecture has made service-oriented software development feasible with mainstream development tool 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, it is possible to 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—for example, a batch computation solver can be rendered as interactive solver services.

Windows Communication Foundation (WCF)

The unified programming model for building service-oriented applications from Microsoft. Using WCF developers can 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 Scheduler.

Job Submission

Windows HPC Server 2008 supports multiple job submission methods:

- The Job Management section of the HPC Cluster Manager client application
- The HPC Job Scheduler user interface client application
- The Command Line Interface (CLI)
- The Windows PowerShell™ interface
- High Performance Computing Basic Profile—an Open Grid Forum standards-based Web service that uses specifications from the Web Services Interoperability organization
- A Component Object Model (COM) interface for integration with Microsoft Visual C++® applications; the COM interface also supports other languages and scripting
- Microsoft .NET APIs to use with .NET applications
- SOA APIs to use with SOA applications

A variety of additional scripting languages are supported in the COM interface, including Perl, Python, Microsoft JScript®, and Microsoft Visual Basic® Scripting Edition (VBScript).

The Windows HPC Server 2008 Job Scheduler includes powerful features for job and task management, including 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, so that jobs and their tasks have the access rights and permissions only of the initiating user. All features of the Job Scheduler are also available from the command line.

Each task within a job can be performed either serially or in parallel. For example, when performing a parametric sweep, a job usually consists of multiple iterations of the same serial executable that are run concurrently but which use different input and output files. There is typically no communication between tasks, but the Job Scheduler achieves parallelism by running multiple instances of the same application at the same time. This is sometimes referred to as an *embarrassingly parallel* application.

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; 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 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 might have to wait for those resources to be free so 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 Scheduler has identified.

Another factor that affects execution time is node exclusivity. By default, jobs require node exclusivity, although this can vary based on job template settings. However, if jobs are defined with 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 job properties, including priority, the run-time limit, the number of required processors, requested nodes, and node exclusivity. After defining 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, in addition to properties similar to those of the job in terms of requested nodes, required processors, the run-time 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 stand-alone Job Management console, or the command line, using either Windows PowerShell or the CLI. In the Job Management Console (or using the **Job Management Navigation** button in the Administration Console), to create a new job, on the **Actions** menu, click **Create New Job**.

Creating Jobs Using the Command Line

To create a job using the Windows HPC Server 2008 CLI, type the following command:

```
job new [options] [/f:<job_file>] [/scheduler:<host>]
```

In this command, *job_file* is the source file used for running the job and *host* is the name of the host on which the job will run.

Table 1 lists the standard options available with the **job** command. Note that when dealing with job priorities, users have access to Normal, BelowNormal, and Lowest priorities, while administrators have access to all 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 to access higher-priority job templates.

To create a job using Windows PowerShell, type:

```
New-HpcJob [options]
```

Windows PowerShell greatly expands the options for managing jobs and their tasks in Windows HPC Server 2008.

Table 1. Job Properties

Property	Description
Job ID	The numeric ID of this job.
Job Name	The name of this job.
Project	The name of the project of which this job is a part.

Property	**Description**
Template	The name of the Job Template against which this job is being submitted.
Priority	The priority of the job; it can be one of five values: Lowest, BelowNormal, Normal, AboveNormal, or Highest.
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.
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.
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.
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.
Exclusive	If set to True, no other jobs can be run on a compute node at the same time as this job.
Requested Nodes	A list of nodes; the job can be run only on nodes that are in this list.
Node Groups	A list of node groups; the job can be run only on nodes that are in all of these groups.
Minimum Memory	The Minimum amount of memory (in megabytes) that must be present on any nodes on which this job is run.

Property	Description
Minimum Processor Count	The Minimum number of processors that must be present on any nodes on which this job is run.
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.
Preemptable	If set to True, this job can be preempted by a higher-priority job. If set to False, the job cannot be preempted.
Fail On Task Failure	If set to True, failure of any task in the job will cause the entire job to fail immediately.
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.
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.

Creating Jobs Using the HPC Pack 2008 Job Management Console

The new Microsoft HPC Pack 2008 Job Management Console, shown in Figure 4, makes creating jobs and assigning tasks straightforward. The Job Management Console is the stand-alone version of the Job Management navigation pane in the Administration Console. It has the same contents and uses the same steps—everything you can do in the Job Management Console you can also do in the Job Management navigation pane of the Administration Console.

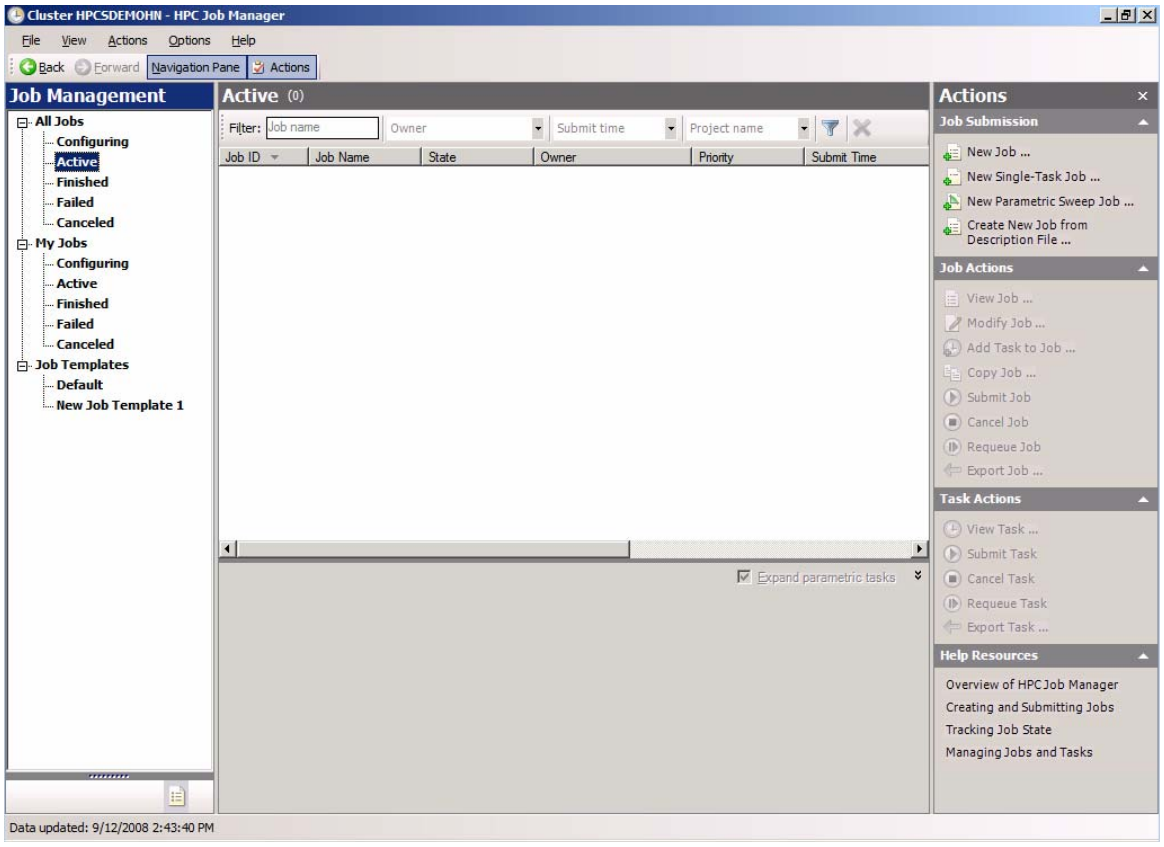
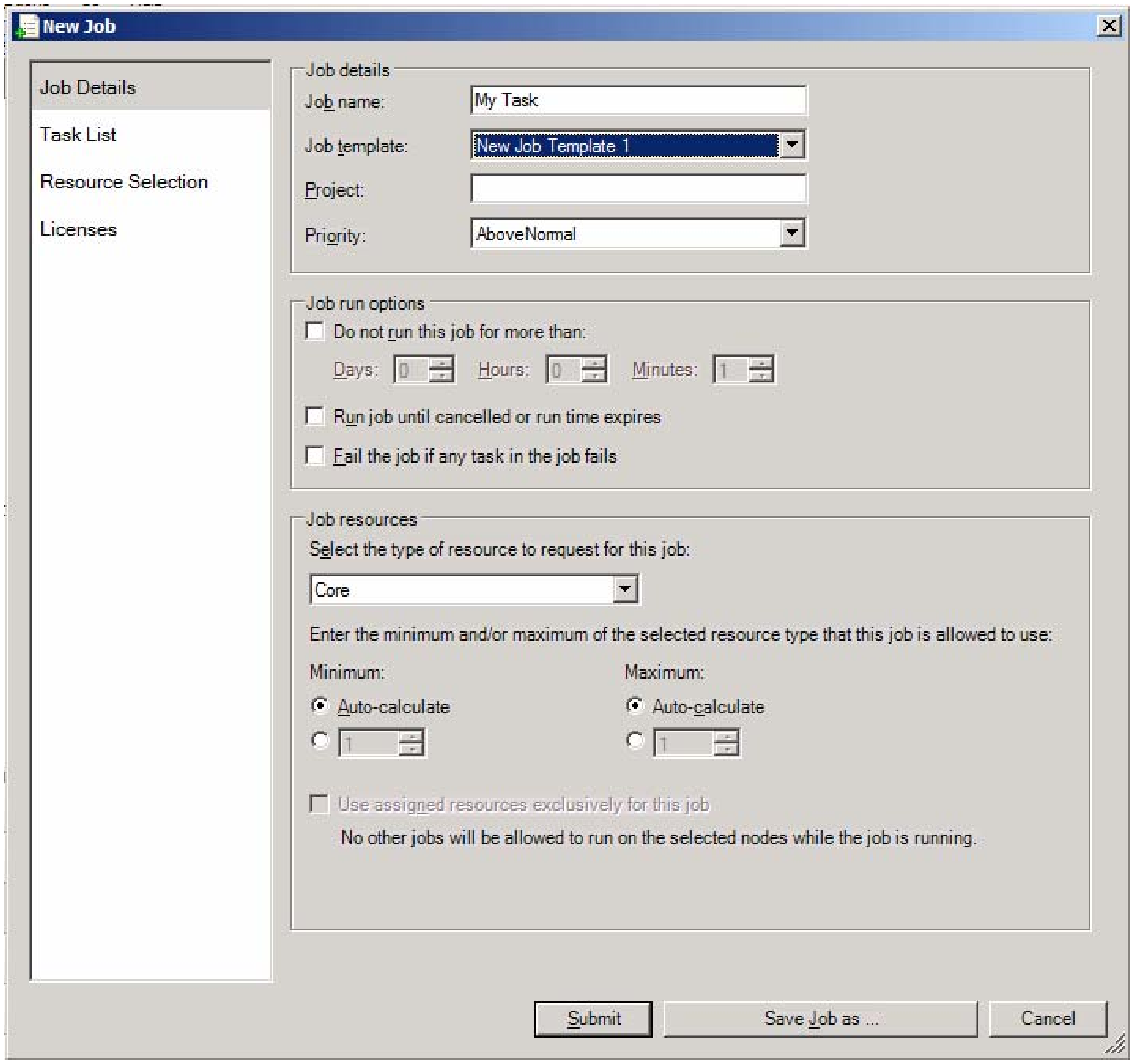


Figure 4. The HPC Pack 2008 Job Management Console

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


New Job

Job details

Job name: My Task

Job template: New Job Template 1

Project:

Priority: AboveNormal

Job run options

☐ Do not run this job for more than:

Days: 0 Hours: 0 Minutes: 1

☐ Run job until cancelled or run time expires

☐ Fail the job if any task in the job fails

Job resources

Select the type of resource to request for this job:

Core

Enter the minimum and/or maximum of the selected resource type that this job is allowed to use:

Minimum: ● Auto-calculate ○ 1

Maximum: ● Auto-calculate ○ 1

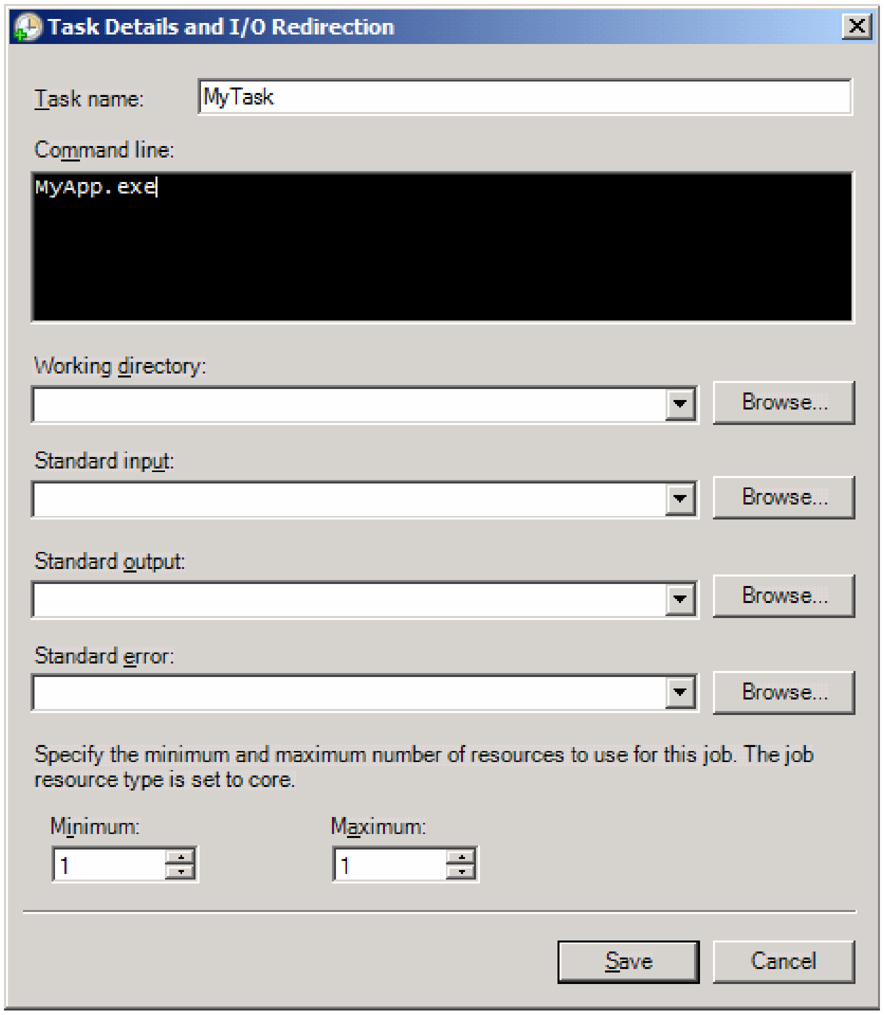
☐ Use assigned resources exclusively for this job

No other jobs will be allowed to run on the selected nodes while the job is running.

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 **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 an MS-MPI executable, the **task** command must be preceded by **mpiexec**. Tasks can run executables directly or can consist of batch files or scripts that perform 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 name: MyTask

Command line:

```
MyApp.exe
```

Working directory:

Standard input:

Standard output:

Standard error:

Specify the minimum and maximum number of resources to use for this job. The job resource type is set to core.

Minimum: 1

Maximum: 1

Figure 6. The Task Details and I/O Redirection page

To create a task using the CLI, type the following command:

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

In this command, *jobID* is the number of the job, *template_file* is the file that contains job commands, and *command* is additional actions for the job to perform.

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

```
        Add-HpcTask [-JobId <jobID>] [options]
```

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

Table 2. Task Properties

Property	**Description**
Task ID	The numeric ID of this task.
Task Name	The name of this task.

Property	Description
Command Line	The command line that will be executed by this task.
Working Directory	The working directory to be used during execution of this task.
Standard Input	The path (relative to the working directory) to the file from which the `stdin` of this task should be read.
Standard Output	The path (relative to the working directory) to the file to which the `stdout` of this task should be written.
Standard Error	The path (relative to the working directory) to the file to which the `stderr` of this task should be written.
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.
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.
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.
Required Nodes	Lists the nodes that must be assigned to this task and its job in order to run.
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.
Exclusive	If set to True, no other tasks can be run on a compute node at the same time as this task.
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.

Property	Description
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*.
Sweep Start Index	The starting index for the sweep.
Sweep End Index	The ending index for the sweep.
Sweep Increment	The amount to increment the sweep index at each step of the sweep.

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 2008 supports parallel and high-performance file systems to improve access to very large data sets. Users of small and medium data set 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 using MS-MPI require the **mpiexec** command; commands for parallel tasks must be in the following format:

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

In this command, *Myapp.exe* is the name of the application to run. Users can submit MS-MPI jobs through the Job Scheduler or at the command line.

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

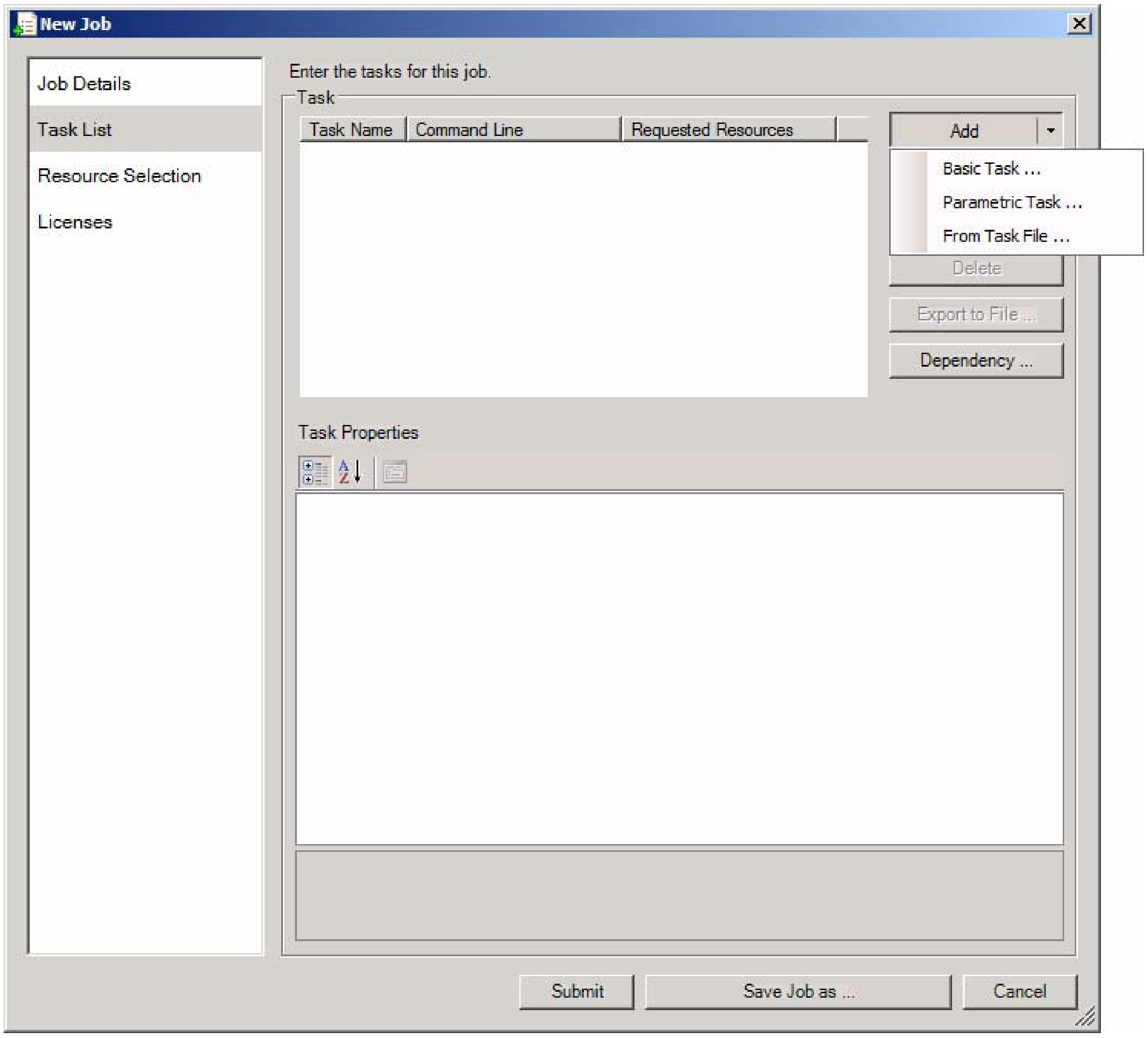
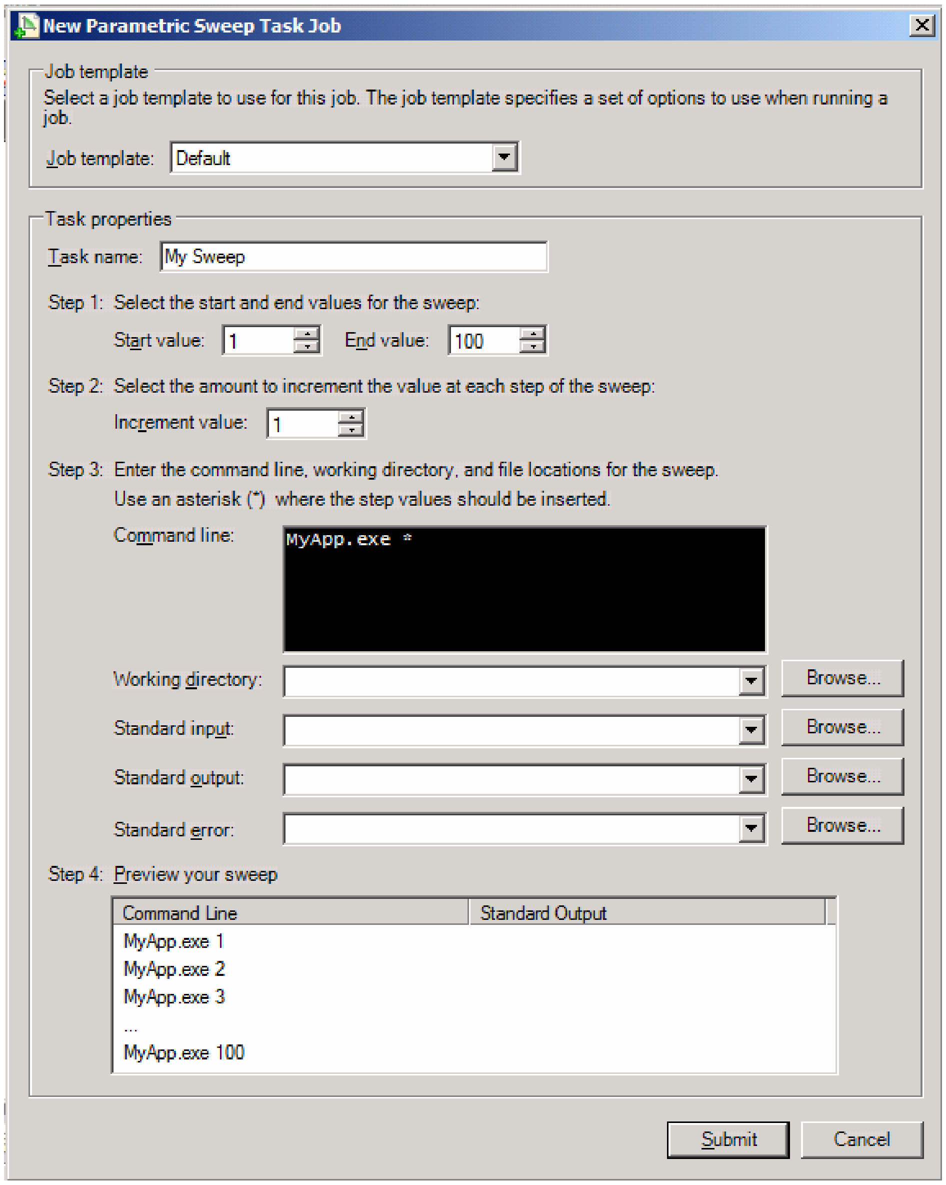


Figure 7. Adding a Parametric Task


New Parametric Sweep Task Job

Job template

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

Job template: Default

Task properties

Task name: My Sweep

Step 1: Select the start and end values for the sweep:

Start value: 1 End value: 100

Step 2: Select the amount to increment the value at each step of the sweep:

Increment value: 1

Step 3: Enter the command line, working directory, and file locations for the sweep. Use an asterisk (*) where the step values should be inserted.

Command line:

```
MyApp.exe *
```

Working directory:

Standard input:

Standard output:

Standard error:

Step 4: Preview your sweep

Command Line	Standard Output
MyApp.exe 1	
MyApp.exe 2	
MyApp.exe 3	
...	
MyApp.exe 100	

Figure 8. The Parametric Task Wizard

In addition to the **job** and **task** commands, users can call the **cluscfg** command, which provides access to information about the cluster itself. See the appendix for a list of the options available for the **job**, **task**, and **cluscfg** commands.

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 recreation, 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**.

Jobs can also be submitted using simple command-line commands—for example:

```
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 HPC Pack 2008 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 the HPC Pack 2008 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 **IScheduler.Connect** method. Create a job using the **IScheduler.CreateJob** method. To create a task, use the **ISchedulerJob.CreateTask** method. To add a child task to a job, use the **ISchedulerJob.AddTask** method. Submit the job using **ISchedulerJob.SubmitJob()**. To be notified of job state changes, use a delegate **job_OnJobState.**

The following sample code shows an example of how you can use the HPC Pack 2008 API following the five-step process just outlined.

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

namespace JobSubV2
{
    class Program
    {
        private static ManualResetEvent jobFinishEvt = new ManualResetEvent(false);
        static void Main(string[] args)
        {
            Scheduler scheduler = new Scheduler();


            scheduler.Connect("localhost");
            ISchedulerJob job = scheduler.CreateJob();

            job.MinimumNumberOfNodes = 1;
            job.MaximumNumberOfNodes = 1;


            ISchedulerTask task = job.CreateTask();

            task.CommandLine = "ComputeTask.exe";
            task.WorkDirectory = @"\\filer\appData";

            job.AddTask(task);

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

            scheduler.SubmitJob(job, null, null);

            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();
            }
        }
    }
}
```

Using the HPC Basic Profile Web Service

A specification for an HPC Web Services interface (WS-HPC) known as 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 that 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 compliant Web service environment; for example, Java, C, C++, or Python.

The following sample source code encapsulates message-generation 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;
        }
```

The Windows HPC Server 2008 SDK contains additional examples (in the Samples directory) of how to use the HPCBP Web service, including a sample client to interact with the Web service and example JSDL files with which you can perform job execution or file staging, or use the SOA infrastructure on a Windows HPC Server 2008 cluster.

Job Scheduling

Job Scheduling allocates jobs—both the ordering within the queue and resource allocation for the execution of tasks. Both are controlled using scheduling policies. Windows HPC Server 2008 supports nine policies, each focused on a specific scheduling issue:

- Priority-based FCFS
- Backfilling
- Nonexclusive schedules
- Resource matchmaking
- Job templates
- MCRA
- Grow/shrink job scheduling policy
- Preemption

Priority-Based FCFS

This scheduling policy combines *first come, first served* (FCFS) and priorities to run jobs. Jobs are placed 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 AboveNormal Priority jobs, and so on.

Backfill

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 use of the cluster by allowing smaller jobs to jump the queue when resources are available.

Exclusive Scheduling

When a job is set to **exclusive**, no other job can run on a node with that job. In addition, 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. In a cluster where the types and capabilities of compute nodes and networking systems are many, and where applications might not be accessible from all the nodes, having the user control job placement can lead to poor time-to-result because of the difficulty of choosing optimally. If a large simulation is run on small memory nodes or a four-way parallel job run exclusively on eight core nodes, the cluster is less effectively used.

The resource matching feature of the Job Scheduler allows the user to specify compute, networking, and application resource requirements so that the scheduler can perform right-sizing placement for the jobs. For example:

```
job submit /memorysize:1000000-3000000 /nodegroup:appX  myapp.exe
```

Job Templates

The administrator creates job templates that define the resources needed for specific processing needs and the priorities of multiple user groups. In the narrowest sense, a job template serves as a pattern for the creation of jobs. Job templates are defined by the administrator and used by users when submitting jobs. Job templates usually define the job parameters for a specific use of an application. As such, they free the user from having to learn the Job Scheduler nomenclature before they can submit jobs from the Job Scheduler interface or the CLI.

In a broader sense, job templates effectively tie the user's processing needs and the organization's resource allocation policies together. Using job templates, the administrator can enforce and mandate the delivery of resources to user groups in ways that meet the organization's productivity goals.

In the Windows HPC Server 2008 Job Scheduler, all job terms are optional. Thus, when a job is submitted, the scheduler assigns default values to the unspecified job terms. Job templates give the cluster administrator the ability to configure the appropriate defaults for the organization. For example, a site might require that all jobs share the nodes, that run time be enforced on all jobs, that valid project names be specified, or that nodes be partitioned and divided among projects in ways that reflect processing deadline constraints. In the Windows Compute Cluster Server 2003 (version 1 of Windows HPC Server 2008) Job Scheduler, the administrator had to create a submission filter to accomplish this task. This involved writing a program that parses a job term's XML file, and then validates and modifies the job parameters according to the site's policies. Using job templates, the cluster administrator can accomplish this without writing plug-ins, in most cases.

Multi-Level Compute Resource Allocation (MCRA)

The MCRA scheduling policy allows jobs to request the granularity with which their compute resources will be scheduled. This allows jobs to be placed more efficiently for their application's performance characteristics:

- *Core* allocation is ideal for applications that are CPU-bound; the more processors you can throw at them the better!
- *Socket* allocation is designed for applications whose performance is bottlenecked by memory access. The amount of data that is able to come in from memory is what limits the speed of the job. Therefore, running more tasks on the same memory bus won't result in increased speeds because all of those tasks are fighting over the path to memory.
- *Node* allocation is best for jobs that rely on per-node resources. This is the case with applications that rely heavily on access to disk or to networks resources. Running multiple tasks per node won't result in increased speed because all of those tasks are waiting for access to the same disk or network pipe.

Grow/Shrink

The grow/shrink scheduling policy allows jobs with multiple tasks to increase and decrease their resource allocations over time. Resources can be added to a job when they become available (*grow*) or reduced when there are no tasks left to start (*shrink*). This results in a higher overall cluster use and helps to ensure allocation of resources to the highest priority jobs in the system.

Preemption

Preemption allows high-priority jobs to take resources away from low-priority jobs that are already running, decreasing the time-to-solution for high-priority work. Windows HPC Server 2008 preemption comes in two modes:

- **Immediate Pre-emption**—Kills low priority jobs to make room for high priority jobs.

Graceful Pre-emption—Shrinks low priority jobs to make room for high priority jobs. Though this means a longer wait than that provided by Immediate Pre-emption, Graceful Pre-emption guarantees that no task will ever be killed, thereby preventing lost work.

Default Resource Allocation

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

- numprocessors
- requestednodes
- nonexclusive
- exclusive
- requirednodes

The Job Scheduler sorts the candidate nodes by largest memory and fastest speed, that is, it sorts the nodes according to memory size first, and then sorts the nodes by their speed. This is the default process 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 **requestednodes** term is specified, the Job Scheduler does not sort the nodes in the node list but instead allocates the processors from the requested list in the user-specified order until the minimum and maximum number of processor requirements are 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, 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 2008, users should follow these guidelines to create, submit, and run jobs:

1. Do not use the default **infinite** run time when submitting jobs. Instead, define a run time that is an estimate of the longest that the job should actually take to complete.
2. Reserve specific nodes only if the job requires the specific 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 might 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. Set a job or task to run **exclusive** only if the job or task requires it.

Following these guidelines will help you make the greatest use of resources available on the Windows HPC Server 2008 cluster.

Job Execution

Job execution starts the jobs. Jobs run in the context of the submitting user, limiting the possibility of runaway processes. Jobs can also be automatically requeued upon failure. Tasks are managed by their state transition. Task states are illustrated in Figure 9.

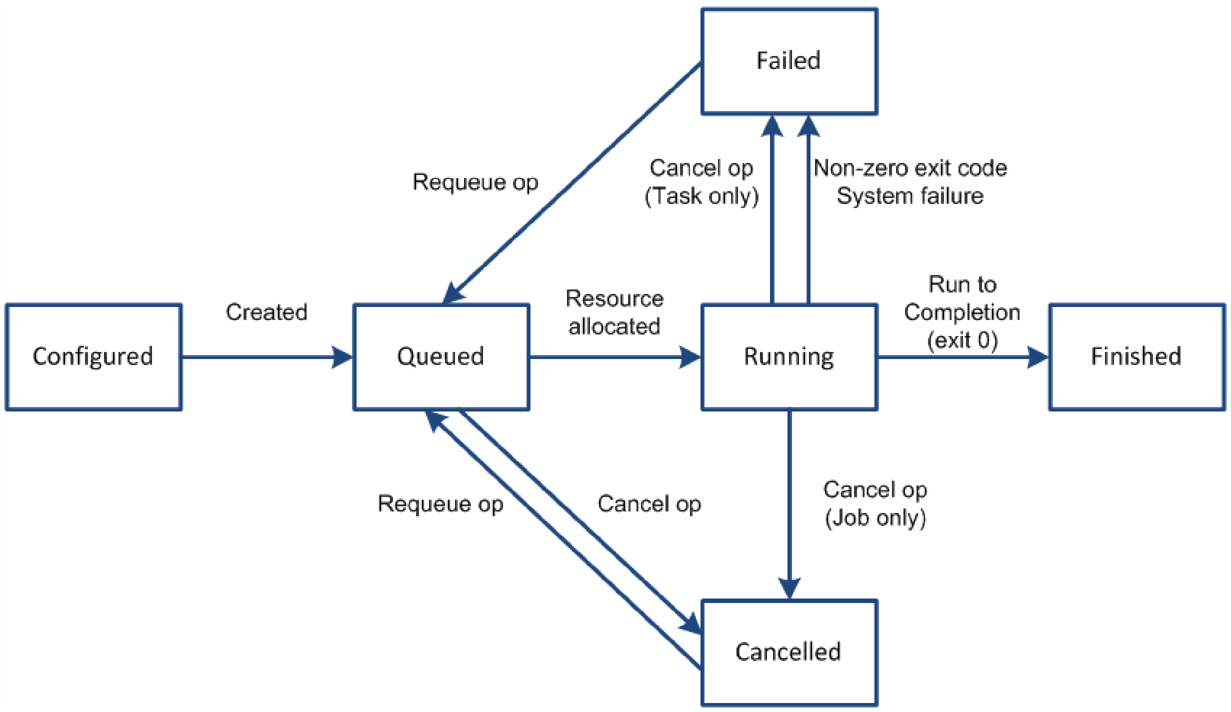


Figure 9. Task state transition

Like tasks, jobs and nodes 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. Table 3 lists Windows HPC Server 2008 status flags.

Table 3. Windows HPC Server 2008 Status Flags

Item	Flag	Description
Jobs and Tasks	CONFIGURING	The job or task has been created but not submitted.
	QUEUED	The job or task is submitted and is awaiting activation.
	RUNNING	The job or task is running.
	FINISHED	The job or task has finished successfully.
	FAILED	The job or task has failed to finish successfully.
	CANCELLED	The user cancelled the job or task.

Item	Flag	Description
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.
	PROVISIONING	The node is being configured as a compute node, including HPC setup and creation of an Active Directory® directory service 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 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 users have for all possible jobs.

In addition, job execution filters can help to assure 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. The Job Scheduler 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.

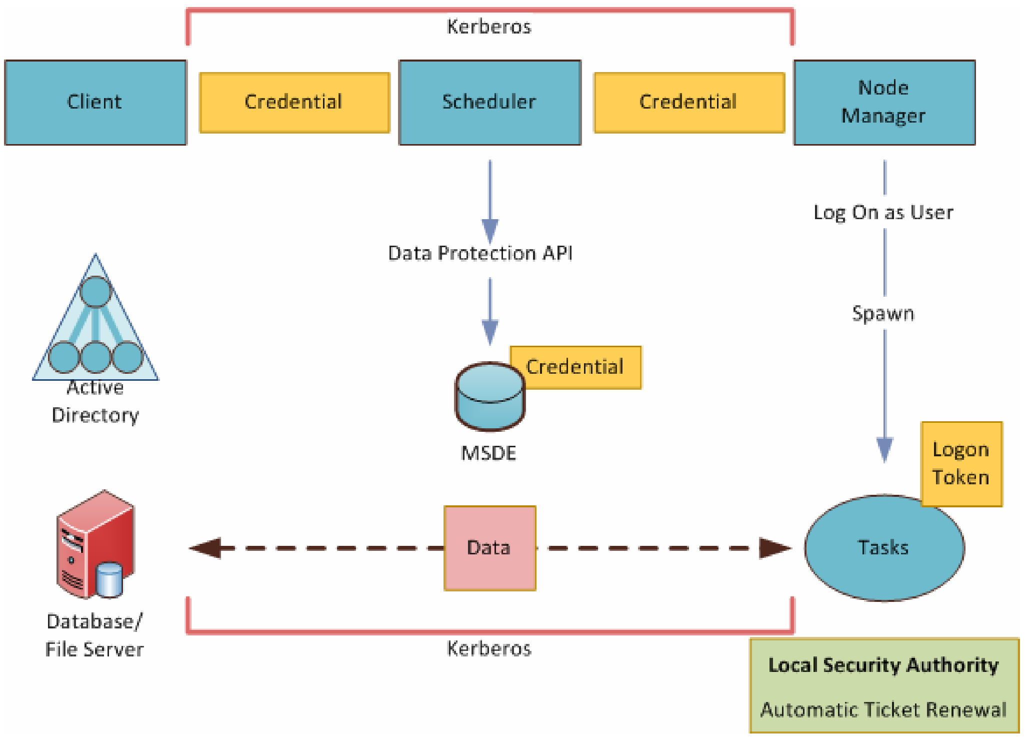
- 1: It is okay to submit the job with changed terms.
- If any other value appears, the job is rejected.

Security Considerations for Jobs and Tasks

Windows HPC Server 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 the 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 the Job Scheduler. 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 SOA Programming Model and Run Time

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 4 shows some example applications and tasks.

Table 4. 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 face 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

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

Table 5. Benefits of Windows HPC Server 2008 for SOA Applications

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

The Programming Model

The key abstraction of the programming model is that a client creates a session using 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 6.

Table 6. 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 additional information about the Session API, see the Windows HPC Server 2008 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 following the example):

```
using System;
using System.Collections.Generic;
using System.Text;
using System.ServiceModel;
using System.Threading;
using Microsoft.Hpc.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, so the developer can 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 dynamic-link library (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.

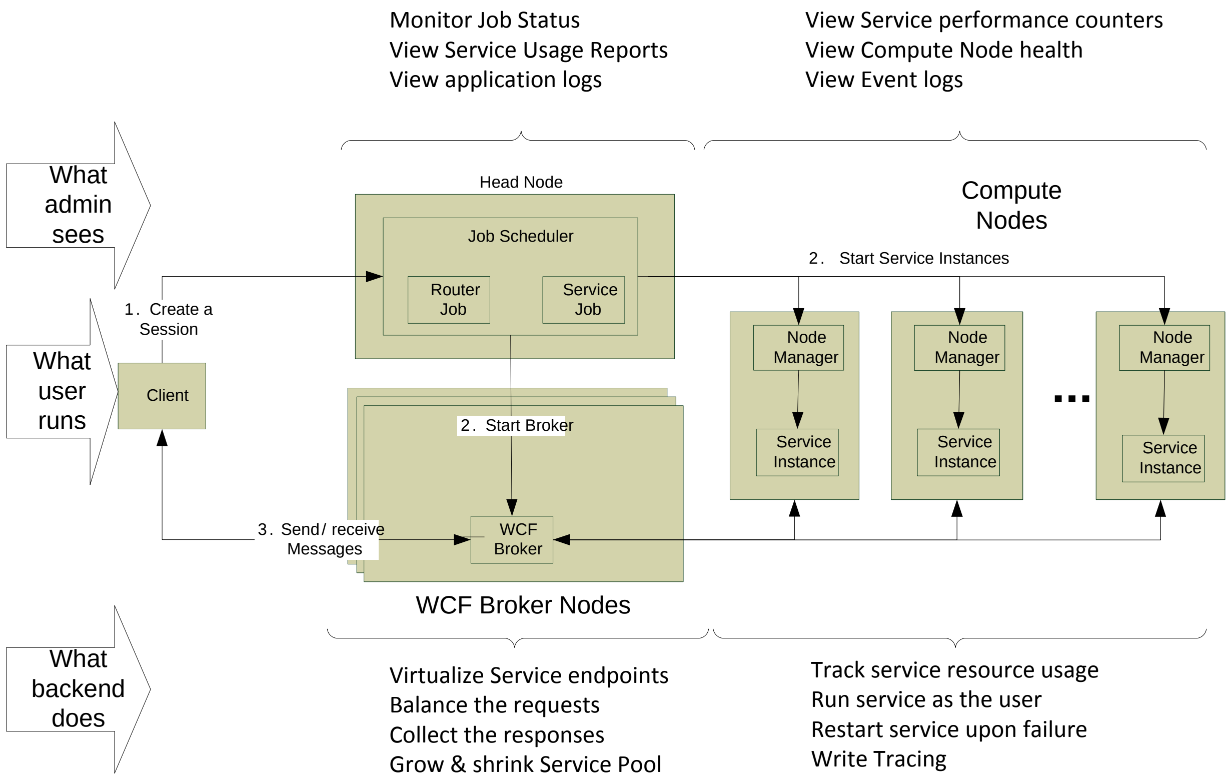


Figure 11. Architecture Overview

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 Scheduler to monitor the progress and resource usage of the session job. The resource use of services is logged so that usage reports based on users, projects, or service names can be created.

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, which provides a streamlined build/debug/deploy developer experience. The new Job Scheduler policies provide the flexibility to manage resource utilization effectively for the cluster without writing plug-ins. The graphical interface for the Job Scheduler is now fully integrated with the Administration Console, and the command-line interface now supports Windows PowerShell for all Job Scheduler functions.

Appendix – Command Line Reference

Table 7. Windows HPC Server 2008 Command-Line Commands

Command	Operators	PowerShell Cmdlet	Description
job	job new [job_terms]	New-HpcJob	Create a job.
	job add jobID [task_terms]	Add-HpcTask	Add tasks to a job.
	job submit /id:jobid	Submit-HpcJob	Submit a job created using the **job new** command.
	job submit [job_terms][task_terms]	Submit-HpcJob	Submit a job.
	job cancel jobID	Stop-HpcJob	Cancel a job.
	job modify jobID [options]	Set-HpcJob	Modify a job.
	job requeue JobID	Submit-HpcJob	Requeue a job.
	job list	Get-HpcJob	List jobs in the cluster.
	job listtasks jobID	Get-HpcTask –JobID JobID	List the tasks of a job.
	job view JobID	Get-HpcJob JobID	View details of a job.
task	task view taskID	Get-HpcTask	View details of a task.
	task cancel taskID	.Cancel()	Cancel a task.
	task requeue taskID	.Requeue()	Requeue a task.
cluscfg	cluscfg view	Get-HpcClusterOverview	View details of a cluster.
	cluscfg listparams/setparams	Get-HpcClusterProperty Set-HpcClusterProperty	View or set configuration parameters.
	cluscfg listenvs/setenvs	Get-HpcClusterProperty Set-HpcClusterProperty	List or set a cluster-wide environment.
	cluscfg delcreds/setcreds	Remove-HpcJobCredential Set-HpcJobCredential	Set or delete user credentials.
node	node list	Get-HpcNode	List the nodes in the cluster.

Command	Operators	PowerShell Cmdlet	Description
	node listcores		List the cores in the cluster.
	node view nodename	Get-HpcNode nodename	View the properties of a node.
jobtemplate	jobtemplate add	New-HpcJobTemplate	Add a job template
	jobtemplate delete	Remove-HpcJobTemplate	Remove a job template
	jobtemplate grant/deny/remove	Set-HpcJobTemplateAcl	Set job template permissions.
	jobtemplate list	Get-HpcJobTemplate	List job templates.
	jobtemplate view	Get-HpcJobTemplate template	View template details.
clusrun	clusrun [options] command		Run a command across a set of compute nodes.