David Chappell
& Associates

WINDOWS HPC SERVER AND WINDOWS AZURE

HIGH-PERFORMANCE COMPUTING IN THE CLOUD

DAVID CHAPPELL

JANUARY 2011

SPONSORED BY MICROSOFT CORPORATION

CONTENTS

HIGH-PERFORMANCE COMPUTING IN A CLOUD WORLD

The essence of high-performance computing (HPC) is processing power. Whether implemented using traditional supercomputers or more modern computer clusters, HPC requires having lots of available computing resources.

Windows HPC Server 2008 R2 Suite lets organizations provide these resources with Windows Server machines. By letting those machines be managed as a cluster, then providing tools for deploying and running HPC jobs on that cluster, the product makes Windows a foundation for high-performance computing. Many organizations today use Windows HPC Server clusters running in their own data centers to solve a range of hard problems.

But with the rise of cloud computing, the world of HPC is changing. Why not take advantage of the massive data centers now available in the cloud? For example, Microsoft's Windows Azure provides on-demand access to lots of virtual machines (VMs) and acres of cheap storage, letting you pay only for the resources you use.

The potential benefits for HPC are obvious. Rather than relying solely on your own on-premises cluster, using the cloud gives you access to more compute power when you need it. For example, suppose your on-premises cluster is sufficient for most—but not all—of your organization's workloads. Or suppose one of your HPC applications sometimes needs more processing power. Rather than buy more machines, you can instead rely on a cloud data center to provide extra computing resources on demand. Depending on the kinds of jobs your organization runs, it might even be feasible to eventually move more and more of your HPC work into the cloud. This allows tilting HPC costs away from capital expense and toward operating expense, something that's attractive to many organizations.

Yet relying solely on the cloud for HPC probably won't work for most organizations. The challenges include the following:

- Like other applications, HPC jobs sometimes work on sensitive data. Legal, regulatory, or other limitations might make it impossible to store or process that data in the cloud.
- A significant number of HPC jobs rely on applications provided by independent software vendors (ISVs). Yet because many of these ISVs have yet to make their applications available in the cloud, a significant number of HPC jobs can't use cloud platforms today.
- Jobs that need lots of computing power often rely on large amounts of data. Moving all of that data to a cloud data center can be problematic.

While the rise of cloud computing will surely have a big impact on the HPC world, the reality is that on-premises HPC clusters aren't going away. Instead, providing a way to combine the two approaches—cluster and cloud—makes sense. This is exactly what Microsoft has done with Windows HPC Server and Windows Azure, supporting all three possible combinations: applications that run entirely in an on-premises cluster, applications that run entirely in the cloud, and applications that are spread across both cluster and cloud.

This paper describes how Windows HPC Server R2 and Windows Azure can work together. After a brief tutorial on each one, it walks through the options for combining them, ending with a closer look at SOA

applications, one example of the kind of HPC application this combination supports. The goal is to provide an architectural overview of what this technology is and why HPC customers should care.

TECHNOLOGY BASICS

Understanding how Windows HPC Server and Windows Azure can be combined for high-performance computing requires understanding the basics of both technologies. This section walks through the fundamentals of each one, starting with Windows HPC Server.

WINDOWS HPC SERVER

In a typical HPC application, logic runs simultaneously on many processers. With Windows HPC Server, this means spreading the application's logic across multiple *compute nodes*, each a computer running Windows. The application might run the same logic on every compute node, with each instance processing different data, or it might run different logic on different nodes.

To assign the logic of a Windows HPC application to compute nodes, Windows HPC Server provides a *scheduler*. This scheduler runs on its own dedicated machine called a *head node*. Figure 1 shows how this looks.

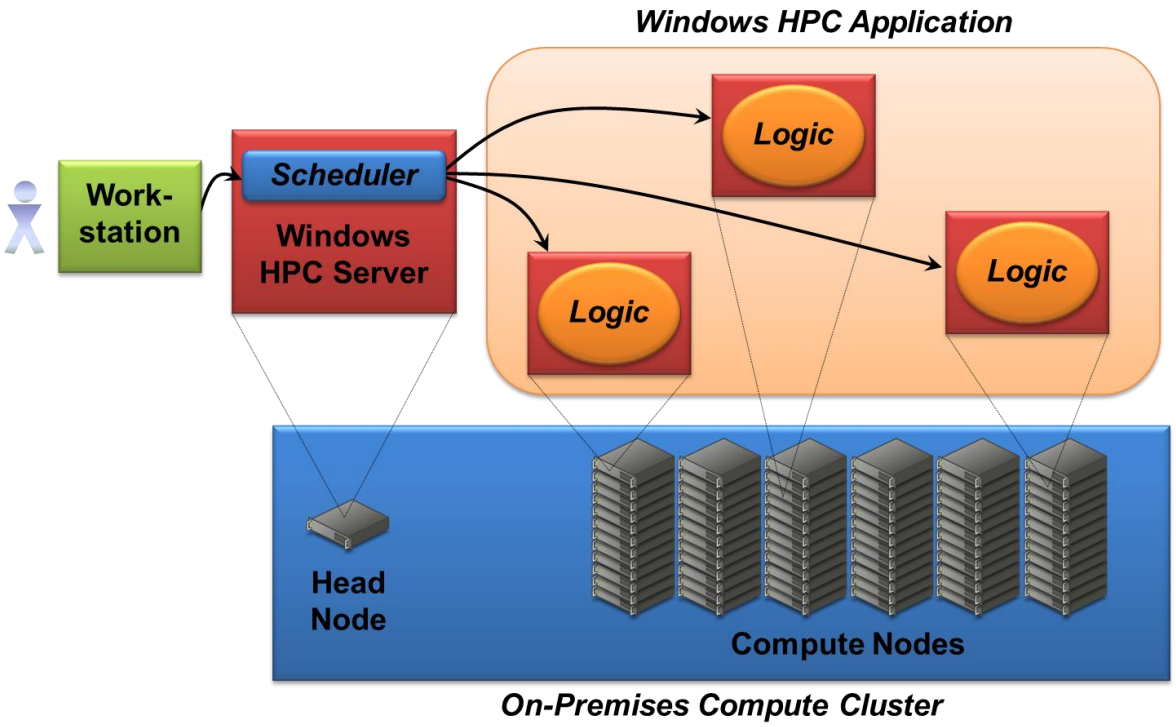

Figure 1: A scheduler running on a head node distributes the logic of a Windows HPC application across a group of compute nodes.

As the figure shows, both the head node and the compute nodes are part of an on-premises compute cluster within this organization. To use these resources, a user submits a Windows HPC job from a workstation to the head node. That scheduler parcels the job's logic across the compute nodes that this

application has access to. If necessary, an organization can create a secondary head node that automatically takes over if the primary fails.

Assigning work to compute nodes isn't a simple task, and so the scheduler is a sophisticated piece of software. Among other things, it assigns jobs to compute nodes based on their priority, allowing new high-priority jobs to jump to the head of the line. It also lets the user submitting a job specify what kind of resources the job needs, then place the job appropriately. For example, memory-intensive jobs can be assigned to compute nodes in a way that minimizes memory contention. The scheduler is also smart enough to avoid letting a few big, high-priority jobs starve all of the others by hogging all of a cluster's resources.

HPC jobs run their logic simultaneously on multiple machines, and so Windows HPC Server supports the industry-standard Message Passing Interface (MPI) to let these distributed chunks of code communicate effectively. HPC jobs also commonly exploit multiple cores on each machine they use, and so Windows HPC Server provides OpenMP, Parallel LINQ (PLINQ), the .NET Framework's Task Parallel Library, and other software to help developers do this. It also provides debuggers and profilers designed expressly for working with distributed, parallel applications.

Much of what Windows HPC Server offers is focused on creating and running applications. Yet managing a cluster and the applications that run on it is also a non-trivial task. For example, an administrator must create the cluster, then determine things such as which compute nodes are available for each job. An application might have a specific set assigned to it, or it might be able to draw compute nodes from a generally available pool. To make life easier for cluster administrators, Windows HPC Server includes HPC Cluster Manager. Figure 2 shows an example of how this tool looks.

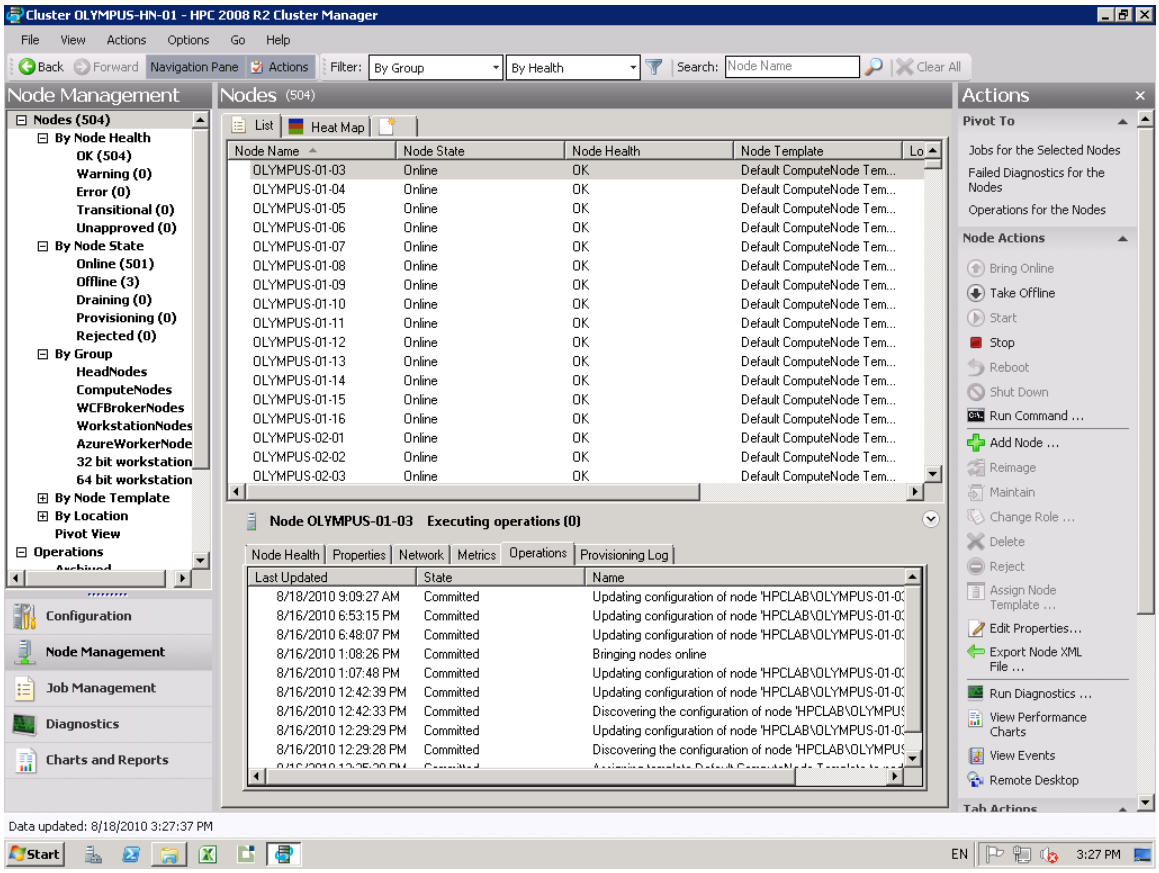

Figure 2: HPC Cluster Manager lets a cluster administrator add nodes to a cluster, monitor and manage jobs running on the cluster, and much more.

As the figure shows, an administrator uses this tool to accomplish a variety of tasks: managing a cluster's nodes, managing jobs running on that cluster, running diagnostic tests on the cluster, creating charts and reports about the cluster, and more. HPC Cluster Manager also provides graphical views, including a heat map showing the utilization of each node in a cluster.

High-performance computing isn't an especially simple area. Building and running HPC applications requires specialized software, as does managing the clusters on which they run. Windows HPC Server aims at providing a cohesive solution for all of these problems.

WINDOWS AZURE

Like Windows HPC Server, Windows Azure relies on lots of computers. Rather than running inside an organization's data center, however, these computers are located in Internet-accessible data centers run by Microsoft. Windows Azure customers can run applications and store data on these machines, paying Microsoft directly for the computing and storage resources they use. Figure 3 shows how this looks.

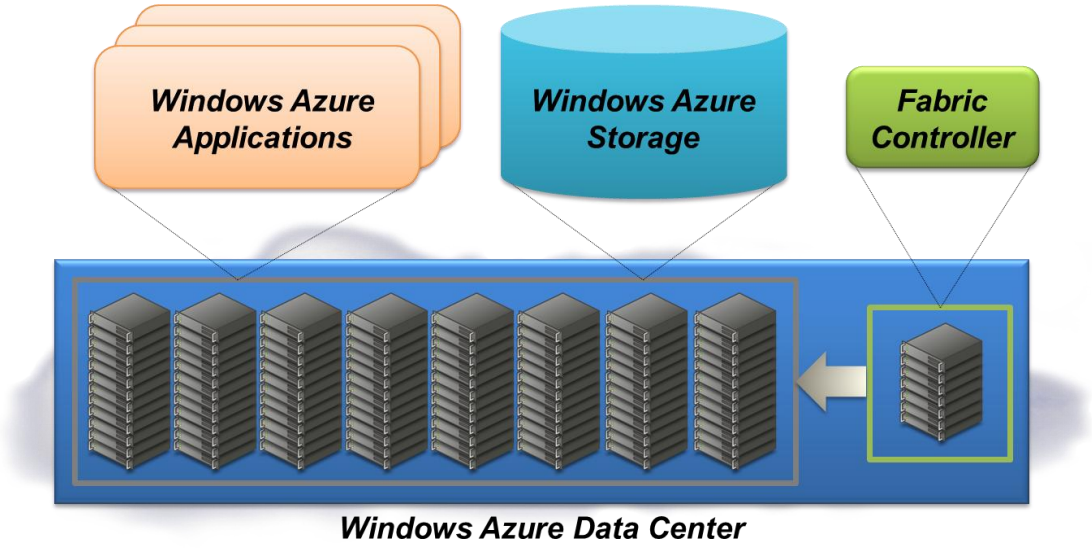

Figure 3: Windows Azure applications can use Windows Azure storage, and both rely on the Windows Azure fabric controller.

As the figure shows, every Windows Azure application runs on computers in a Windows Azure data center. Those applications (or even applications running on-premises) can use Windows Azure storage, which provides two main options:

- Blobs, offering unstructured storage for binary data.
- Tables, providing structured storage for large data sets. (Don't be confused, however—despite their name, these tables don't provide relational storage.)

All of this is managed by the *fabric controller*, an application running on its own dedicated set of machines. Among other things, the fabric controller handles any updates required to the computers in this Windows Azure data center, such as operating system patches.

To understand how Windows HPC Server and Windows Azure work together requires knowing a bit more about Windows Azure applications. Whatever its purpose, a Windows Azure application is always implemented as one or more *roles*. A *Web role*, for example, is typically used for code that accepts HTTP requests via Internet Information Services (IIS). A *Worker role* is a more general option that can be used for various kinds of processing. Whatever roles an application uses, Windows Azure requires it to run at least two instances of each role. These instances are typically interchangeable—the failure of any one won't take down the application—and each instance runs in its own VM. Depending on the load it needs to handle, an application might run two, ten, or fifty instances of a role. Figure 4 illustrates these ideas.

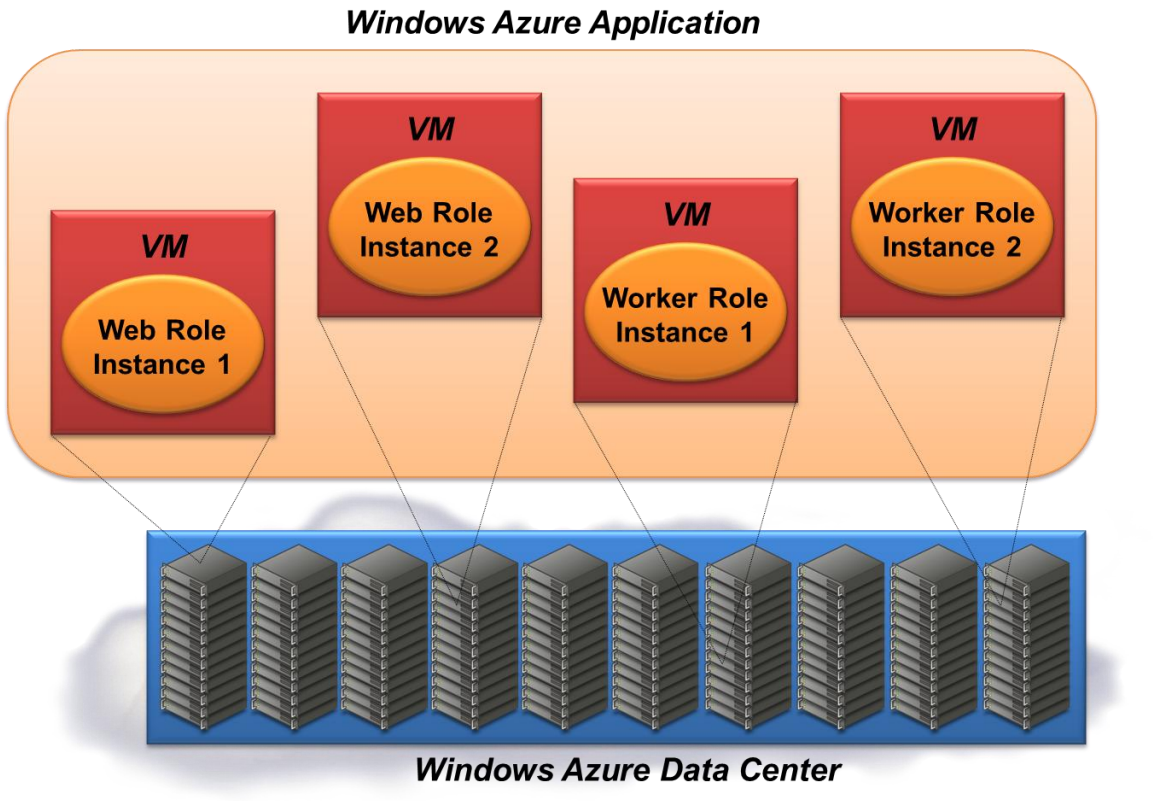

Figure 4: A Windows Azure application can consist of Web role instances, Worker role instances, or both, with each instance running in its own virtual machine.

In this simple example, the Windows Azure application is implemented using one Web role and one Worker role. It's currently running two instances of each role, but changing this is straightforward. A developer or administrator can tell Windows Azure to increase or decrease the number of instances, for example, or the application can do this itself.

As the figure shows, the Windows Azure fabric controller assigns each instance to a different computer. This helps make applications more fault-tolerant, since a single hardware failure won't take down the entire application. The fabric controller also monitors each running application: If an instance fails, the fabric controller starts a new instance of the same role within a specified time defined by the platform's service level agreement (SLA). And because each instance has one or more dedicated processer cores, an application's performance is predictable—it doesn't share those cores with other applications.

Windows Azure is based on Windows, and so the applications it runs can be created using the .NET Framework. Those applications can also be built using raw C++, however, or other technologies such as PHP and Java. And while Visual Studio is today's most commonly used tool for creating Windows Azure applications, it's not the only choice—developers are free to use whatever tools they like.

However it's built, a Windows Azure application can access data stored in various places. One common choice is to use the blobs and tables provided by Windows Azure storage. Another option is to use SQL Azure, Microsoft's cloud-based service for relational data. An application is also free to access on-

premises data within an organization—there's no requirement that a cloud application use only data stored in the cloud.

Finally, understanding when using Windows Azure makes sense requires understanding how it's priced. Here's a summary of the main things to know:

- For compute, customers are charged per instance (i.e., per VM) per hour. The price ranges from $0.05 to $0.96 an hour based on the instance's size, with options ranging from one to eight cores. Note that this charge isn't usage-based; an instance that sits unused for an hour will incur the same charge as an instance that spends an hour doing intensive calculations.

- For storage, data kept in Windows Azure blobs and tables costs $0.15 per gigabyte per month. Customers also pay $0.01 per 10,000 operations on this data, such as reads and writes.

- For bandwidth, customers pay $0.10 per gigabyte in and $0.15 per gigabyte out. These costs are incurred only for data moved into and out of a Windows Azure data center—there's no bandwidth charge for Windows Azure applications that access blobs or tables (or SQL Azure data) in the same data center.

Cloud platforms offer a new approach to deploying, using, and paying for computing resources. With Windows Azure, Microsoft provides a way for Windows-oriented developers to embrace this new style of computing.

USING WINDOWS HPC SERVER WITH WINDOWS AZURE

HPC depends on processing power. Windows Azure provides a vast amount of processing power on demand. It's obvious: The two are a natural combination.

But for all of the reasons described earlier, the rise of the cloud doesn't signal the end of on-premises HPC clusters. Instead, cluster and cloud can work together, as the marriage of Windows HPC Server and Windows Azure demonstrates. This combination supports two approaches today:

- Keeping the head node and some compute nodes on premises, with extra compute nodes in the cloud.
- Keeping only the head node on premises, with all compute nodes in the cloud.

This section looks at both options.

USING ON-PREMISES COMPUTING AND CLOUD COMPUTING TOGETHER

From an architectural perspective, running a Windows HPC application that uses both an on-premises compute cluster and Windows Azure is straightforward. Figure 5 shows how this looks.

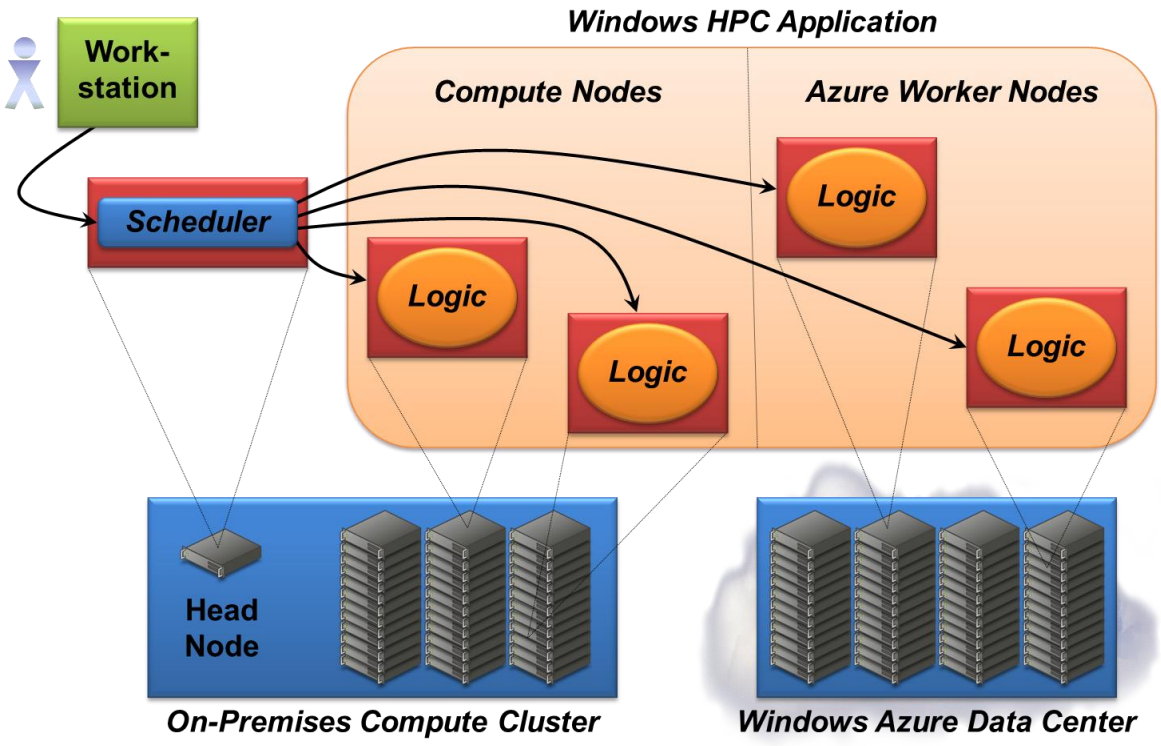

Figure 5: A Windows HPC application can use both on-premises compute nodes and Azure Worker nodes (i.e., Windows Azure Worker role instances).

The head node remains on premises, and a user submits the HPC job in the usual way. The scheduler on the head node then parcels out the application's logic across the available compute resources. In this case, those resources include both compute nodes in the on-premises cluster and Worker role instances—referred to by Windows HPC Server as *Azure Worker nodes*—in a Windows Azure data center.

To a user, submitting a Windows HPC job that runs partly on-premises and partly in the cloud looks just as it always does—there's no difference from submitting one that runs entirely in a local cluster. For the Windows HPC administrator, however, some extra work is required to make this possible. While the admin still uses HPC Cluster Manager and other familiar tools, she must set up a Windows Azure account and configure the Azure Worker nodes. The administrator also specifies which jobs are allowed to use which compute resources, whether those resources are compute nodes in the on-premises cluster or Azure Worker nodes in the cloud.

Azure Worker nodes aren't created on demand when a job is submitted. Instead, the administrator explicitly allocates these VMs, then shuts them down when they're no longer needed. For example, an administrator might create 25 Azure Worker nodes at 9 am each day, then take them down at 6 pm. This can be done manually through HPC Cluster Manager or programmatically via a (potentially automated) script. Using Azure Worker nodes costs money—each one is a Windows Azure Worker role instance—and so Windows HPC Server gives an administrator control over when those instances are created and how long they run.

Although it's not shown in the diagram, this application almost certainly reads data in and writes data out. That data might be stored on premises, with the local compute nodes having direct access while the Azure Worker nodes access it across the Internet. The application's data might also be stored in the cloud using, say, blobs in Windows Azure storage, or it might be divided between on-premises storage and cloud storage. Windows HPC Server doesn't mandate where data is stored or how it's accessed—it's up to the creator of each application to make this decision.

What kind of applications would benefit from combining on-premises compute nodes and cloud-based Azure Worker nodes? When would this approach be useful? Here are some examples:

- An application that sometimes needs extra compute power could use Windows Azure to provide this on demand. Rather than investing in more machines for an on-premises cluster, an organization can use the cloud to get the resources it needs at a lower cost.

- An organization that periodically needs extra HPC compute nodes might use Windows Azure to provide them. Think of a financial services firm, for example, that runs CPU-intensive jobs once a week. Rather than expand its existing cluster, then watch those new machines sit idle most of the time, this firm could instead allocate Azure Worker nodes when needed, then shut those instances down when it's finished using them.

- An organization might be running out of room for its on-premises cluster. Rather than expand by adding more physical machines to a crowded data center, they could instead choose to expand into the cloud by using Azure Worker nodes. As mentioned earlier, this also avoids the capital expense of buying new hardware, replacing it with the operating expense of using resources in the cloud. And because the Windows Azure fabric controller automates routine tasks like patching Windows, the administrative costs of an HPC environment can also be reduced.

Whatever the reason, developers creating HPC applications that run across Windows HPC Server and Windows Azure see a familiar world. They can still use Visual Studio (or other Windows development tools) and rely on the product's standard profiling and debugging support. The big change is that their application can now have access to significantly more computing power.

USING CLOUD COMPUTING EXCLUSIVELY

If putting some compute nodes in Windows Azure makes sense—and it often does—why not put all of them in the cloud? Windows HPC Server also supports this option, as Figure 6 shows.

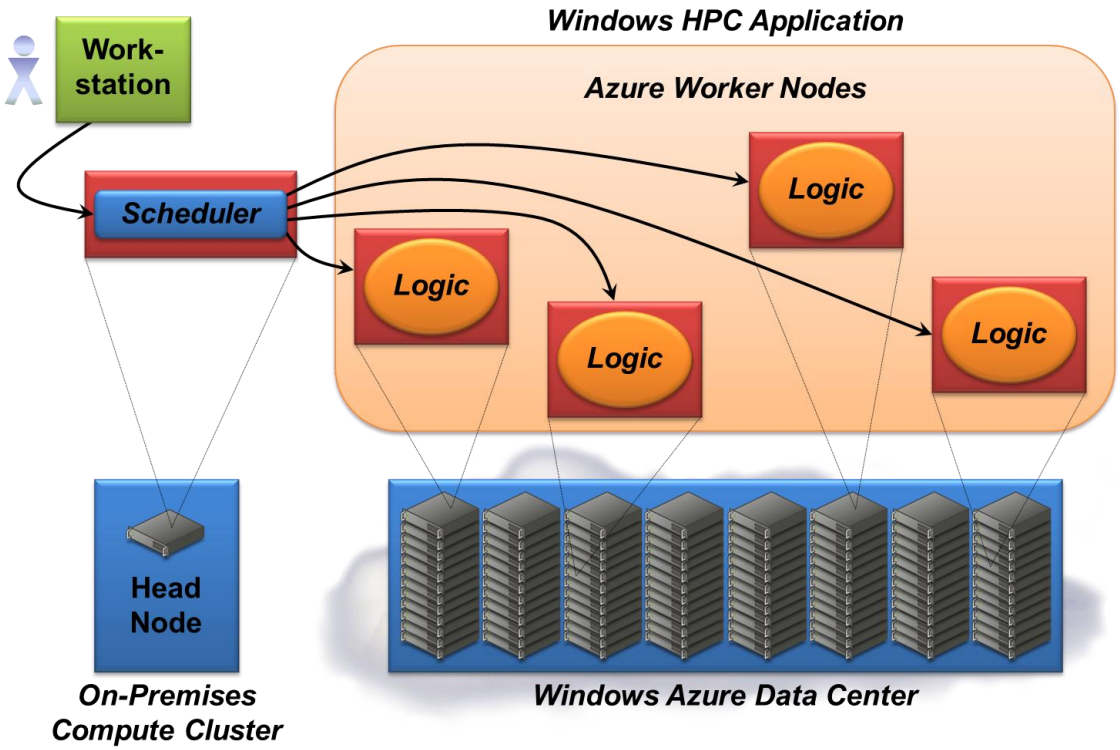

Figure 6: A Windows HPC application can rely entirely on Azure Worker nodes.

In this case, only the head node remains on premises. All of the processing is done in Azure Worker nodes running on Windows Azure. Once again, a user submitting a Windows HPC job sees no difference—the scheduler transparently spreads the application's logic across the Azure Worker nodes.

Putting all of an application's computing in the cloud can have some clear benefits. They include the following:

- An organization can run HPC jobs without buying and managing its own cluster. Maybe the utilization of an on-premises cluster wouldn't be high enough to justify the expense, or perhaps a firm needs HPC capabilities only once a month. Whatever the situation, having just the head node on premises with all computing done on Azure Worker nodes can make economic sense.

- The cost of entry to HPC can be significantly lower than with an on-premises cluster. Rather than spending a chunk of money up front on hardware, an organization can instead pay only for the HPC resources it needs when it uses them.

- Compared to dividing an HPC application between an on-premises cluster and Windows Azure, running the application entirely in the cloud can make data access easier. Rather than dividing the data between the two worlds or making one part of the application access data remotely, all of the application's data can be stored in the cloud.

- For some organizations, the SLA provided by Windows Azure might provide more reliability than their on-premises cluster. And since the Windows Azure fabric controller automatically handles updates to system software, the instability these can cause is minimized.

Whether you choose to split an HPC application between cluster and cloud or to run that application entirely in the cloud, the potential benefits of combining HPC and cloud computing are evident. While it's not right for every situation, expect to see an increasing amount of HPC workloads taking advantage of what the cloud has to offer.

SCENARIOS

HPC applications come in several different types. One large category consists of applications in which logic running on different compute nodes must interact, such as to exchange intermediate results, while the application runs. This kind of application commonly relies on a low-latency communication mechanism programmed using MPI, and it usually runs as a batch job. Another category contains applications where the logic running on each compute node can work in isolation—there's no need to interact while the application runs. This kind of job is commonly referred to as *embarrassingly parallel*, since dividing its work into parallel tasks is so easy.

Microsoft has announced that, over time, it plans to offer a range of HPC application types in the cloud. Initially, however, Windows HPC Server supports using Windows Azure only for embarrassingly parallel applications, those that don't require compute nodes to communicate with each other while they do their work. There are two approaches to creating these applications: *service-oriented architecture (SOA)* applications and *parametric sweep* applications. This section looks at both.

SOA APPLICATIONS

As the name suggests, the compute logic in a SOA application is implemented as a service. With Windows HPC Server, each service is implemented using Windows Communication Foundation (WCF), allowing the service to expose its operations via SOAP or REST or something else. This approach lets users interact with the job while it's running, something that's useful in many situations.

Windows HPC Server supports SOA applications in both on-premises clusters and in the Windows Azure cloud. What follows describes both.

Running SOA Applications on Windows HPC Server

Suppose you need to write a risk analysis application using a Monte Carlo simulation that will run on Windows HPC Server. This application is likely to be embarrassingly parallel—it can easily be broken up into independent chunks—and so it's a good candidate for a SOA application.

Each service in a SOA application provides the same operations, and each runs on its own compute node. Each of these services can also read and write data from wherever the developer chooses, such as a shared file system. Figure 7 shows how a SOA application looks running on an on-premises compute cluster.

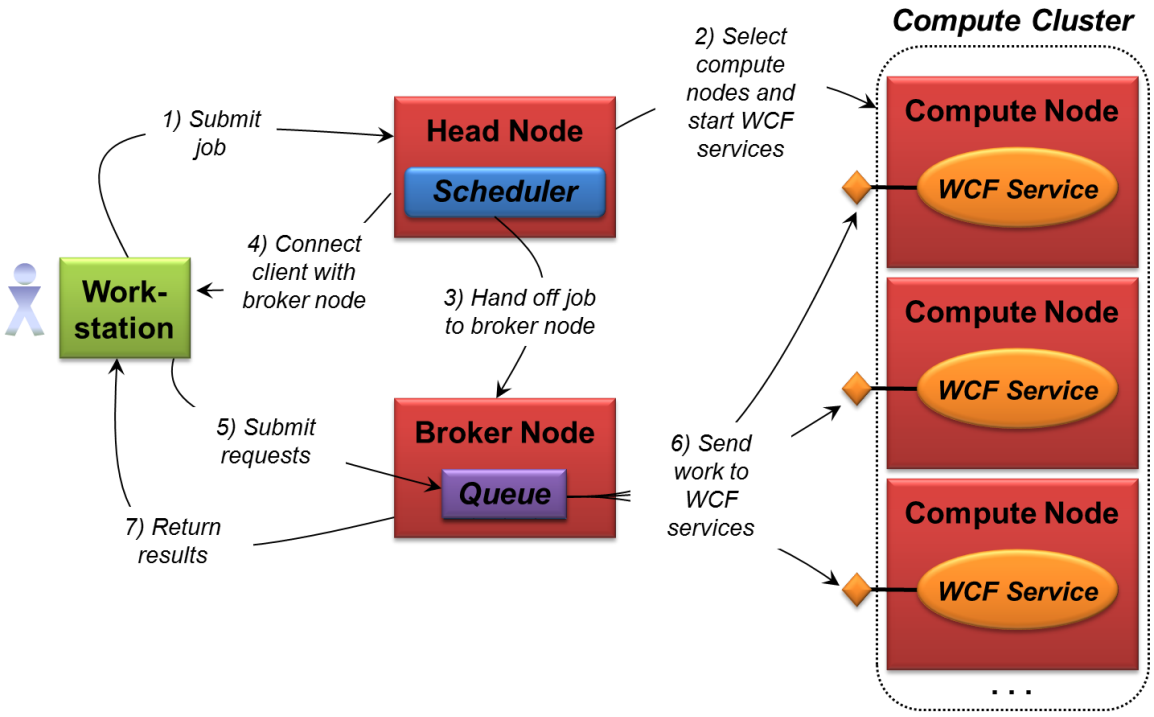

Figure 7: A SOA application is implemented as a group of WCF services, each running on its own compute node.

To kick off this application, a user submits the job from a client workstation to the head node (step 1). The head node then determines which compute nodes should be dedicated to this job and starts an instance of the application's WCF service on each one of them (step 2). The WCF services used by a SOA application are implemented as libraries, and so Windows HPC Server provides a host process that runs in each computer node. The WCF services for any SOA application can be loaded into this process on demand.

The application is now ready to receive requests, but the scheduler in the head node doesn't issue these requests itself. Instead, it hands the job off to a *broker node*, another computer in this cluster (step 3), then connects the client workstation with this broker (step 4).

Since a SOA application's processing is carried out by services, running the application means invoking those services. To do this, the client workstation submits requests to the broker node (step 5). The broker places these requests in a queue, as the figure shows, then invokes the WCF services running in compute nodes to execute each request (step 6). (If a compute node fails, the broker will notice this and re-send the request that node was working on to an instance of the service running on another compute node.) And because SOA applications are often interactive, the broker node can send any results the services return back to the user's workstation (step 7).

Why involve a broker node? Why not just let the head node handle everything? The primary answer is that adding a broker offloads a significant amount of work from the head node. Rather than require the head node to accept requests from the client workstation, queue those requests, submit them to the application's WCF services, then return the results, the broker handles all of this. If a broker node fails, the

system can automatically switch to another broker node. This new broker node has access to the same queue used by its now-dead predecessor, and so no work is lost.

While SOA applications aren't the majority of today's HPC jobs, they do represent an important part of modern high-performance workloads, especially in financial services and life sciences. And while SOA applications run perfectly well in an on-premises compute cluster, they can also take advantage of the cloud resources that Windows HPC Server can make available through Windows Azure. The next section shows how.

Running SOA Applications on Windows HPC Server and Windows Azure

Using Windows Azure to run SOA applications is quite similar to running those applications in an on-premises cluster. Everything looks the same to a user submitting a job, and the basic interactions among the components don't change. The big difference is where the computing gets done.

When SOA applications run in the cloud, they can use either of the approaches described earlier: executing on a combination of cluster compute nodes and Azure Worker nodes or running entirely on Azure Worker nodes. Figure 8 illustrates the second of these options, running a SOA application entirely in the cloud.

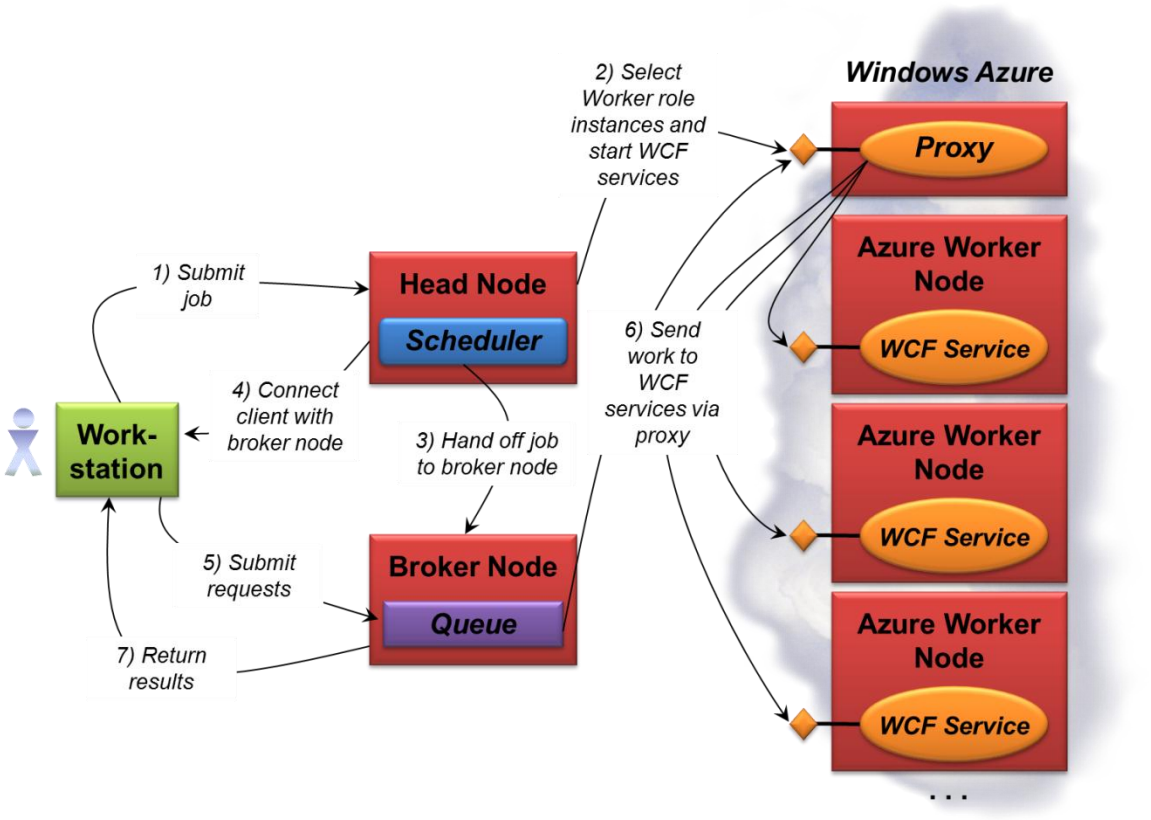

Figure 8: A SOA application running entirely on Windows Azure runs its WCF services in Azure Worker nodes.

Recall that before an application can use Azure Worker nodes, a Windows HPC Server administrator must create a Windows Azure Worker role instance for each one. Just as with on-premises compute nodes, Windows HPC Server provides a host process that runs in each Azure Worker node. The libraries that contain a SOA application's WCF services can then be loaded into this process.

Once the Azure Worker roles have been created, submitting and executing a SOA application follows the steps shown above, all of which echo what happens in the on-premises case. As before, a user at a workstation submits a job to the (on-premises) head node (step 1). This submission includes configuration information that tells the head node how many cores the job needs. In this example, the job will run entirely in the cloud, and so the head node determines how many Azure Worker nodes to use and starts this application's WCF service in each one (step 2). As Figure 8 shows, the head node relies on a proxy running in the cloud to carry out its requests.

Once the Azure Worker nodes are available, everything works much like the on-premises case. The head node hands off the job to a broker node (step 3), then connects the client to this broker (step 4). (Like the head node, the broker node remains on premises even when the application runs entirely on Windows Azure.) The client submits requests to the broker (step 5), which queues them, then passes them on to the application's WCF services via the proxy (step 6). Any results those services return are sent back to the user's workstation (step 7). While the components and their interactions are like those shown earlier, there's an important difference: This SOA application's WCF services are now running in virtual machines in a Windows Azure data center rather than in an on-premises cluster.

As in the on-premises case, queued requests sitting in a failed broker node can be resumed by a new broker that takes its place. And if the broker sends a request to an Azure Worker node that fails, the broker will re-send that work to another instance. Recall that the Windows Azure fabric controller monitors each running instance, starting new ones if any fail, so there's no need for the Windows HPC Server administrator to worry about this.

PARAMETRIC SWEEP APPLICATIONS

In a SOA application, logic is implemented as a WCF service. In a parametric sweep application, however, the compute logic is implemented as an ordinary executable. Rather than exposing services that let a user interact with the job while it's running, the submitter instead can pass different initial parameters to the executable on each compute node when the job starts running. This lets each executable apply the same logic to different data. And while parametric sweep applications can't interact with users while they're running, they also don't require using WCF. This can make them easier for scientists and other non-professional developers to create.

Both SOA applications and parametric sweep applications fit in the embarrassingly parallel category, and both can run in an on-premises cluster, on the Windows Azure cloud, or spread across cluster and cloud. Figure 9 illustrates the basics of running a parametric sweep job entirely in the cloud.

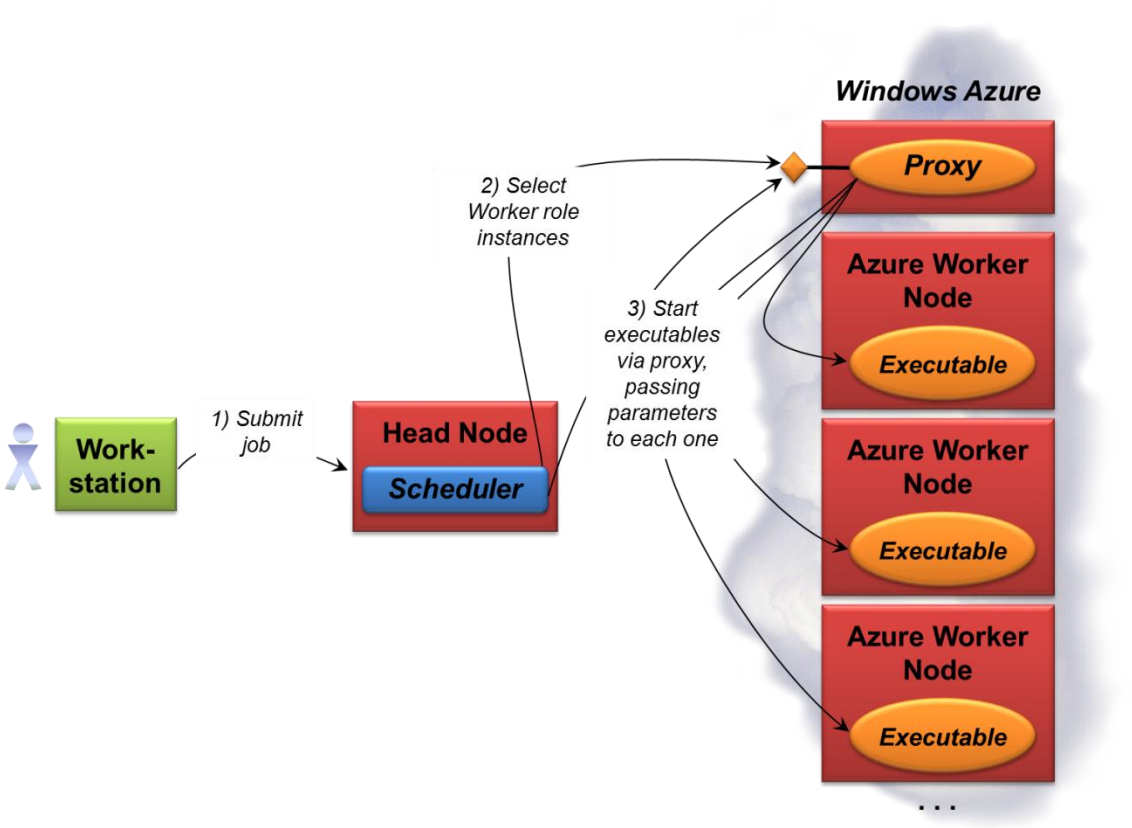

Figure 9: A parametric sweep application running entirely on Windows Azure runs each of its executables in Azure Worker nodes.

As the figure shows, a user submits a parametric sweep job to the on-premises head node as usual (step 1). The scheduler running on the head node then determines which Worker nodes to use (step 2) and, via the proxy, starts the job's executable in each one (step 3). Each executable gets different parameters passed into it, letting each one work on different data. The job then runs to completion with no further input from the user.

Running a parametric sweep application in an on-premises cluster is similar. The primary difference is that there's no proxy involved. Instead, the scheduler on the head node interacts directly with the cluster's compute nodes, starting an executable on each one.

Whether an embarrassingly parallel application runs entirely in Windows Azure or divides its processing between Windows Azure and an on-premises cluster, using the cloud can make sense. On-demand access to pay-as-you-go processing power is an attractive thing for HPC applications.

CONCLUSIONS

High-performance computing is entering a new era. The enormous scale and low cost of cloud computing resources is sure to change how and where HPC applications run. Ignoring this change isn't an option.

Yet the rise of the cloud doesn't imply the death of today's clusters. Data issues, hardware limitations, and other concerns mean that some HPC jobs remain best suited for on-premises compute clusters. This reality is why Windows HPC Server supports all three possibilities:

- Running applications entirely in an on-premises cluster.
- Running applications entirely in the cloud.
- Running applications across both an on-premises cluster and the cloud.

While the cloud options are initially available only for embarrassingly parallel applications—SOA and parametric sweep—expect the choices to broaden over time.

HPC applications can help solve some of the world's most important problems. Cloud computing is surely among the most important technology trends today. It shouldn't be surprising that the two are a natural combination.

ABOUT THE AUTHOR

David Chappell is Principal of Chappell & Associates (www.davidchappell.com) in San Francisco, California. Through his speaking, writing, and consulting, he helps people around the world understand, use, and make better decisions about new technologies.