Grid and Cluster Computing:

Options for Improving Windows® Application Performance

Table of Contents:

Introduction

The terms "grid computing" and "cluster computing" have been used almost interchangeably to describe networked computers that run distributed applications and share resources. They have been used to describe such a diverse set of distributed computing solutions that their meanings have become ambiguous. Both technologies improve application performance by executing parallelizable computations simultaneously on different machines, and both technologies enable the shared use of distributed resources.

However, cluster and grid computing represent different approaches to solving performance problems; although their technologies and infrastructure differ, their features and benefits complement each other. A cluster and a grid can run on the same network at the same time, and a cluster can even contribute resources to a grid.

Both of these forms of distributed computing have their roots in the UNIX operating system. However, as operating systems and networks have evolved, more operating systems have been adapted for use in both clusters and grids. Recent software releases have greatly improved both cluster and grid computing on the Microsoft® Windows® operating systems.

This document defines grid computing and cluster computing from the perspective of the Windows operating systems and development environments. After reading this document you will have a clear understanding of the Windows cluster computing and grid computing options, and you will understand how the two technologies complement each other.

What's the Difference?

The computers (or "nodes") on a cluster are networked in a tightly-coupled fashion--they are all on the same subnet of the same domain, often networked with very high bandwidth connections. The nodes are homogeneous; they all use the same hardware, run the same software, and are generally configured identically. Each node in a cluster is a dedicated resource--generally only the cluster applications run on a cluster node. One advantage available to clusters is the Message Passing Interface (MPI) which is a programming interface that allows the distributed application instances to communicate with each other and share information. Dedicated hardware, high-speed interconnects, and MPI provide clusters with the ability to work efficiently on "fine-grained" parallel problems, including problems with short tasks, some of which may depend on the results of previous tasks.

In contrast, the nodes on a grid can be loosely-coupled; they may exist across domains or subnets. The nodes can be heterogeneous; they can include diverse hardware and software configurations. A grid is a dynamic system that can accommodate nodes coming in and dropping out over time. This ability to grow and shrink at need contributes to a grid's ability to scale applications easily. Grids typically do not require high-performance interconnects; rather, they usually are configured to work with existing network connections. As a result, grids are better suited to relatively "coarse-grained" parallel problems, including problems composed primarily of independent tasks. There is no dominant programming paradigm in grid computing today, and a key challenge to increasing the acceptance of grid computing is creating grid-enabled applications with familiar programming models. Digipede's object-oriented programming for grid (OOP-G) is one such model.

Grids can incorporate clusters. Often the best way to make use of all available resources is to manage the cluster resources as part of a larger grid, assigning jobs and tasks to the resources best suited to those jobs and tasks. For example, jobs requiring MPI would be assigned exclusively to the cluster, while loosely-coupled jobs could be assigned to all grid nodes, including those in the cluster (when available). Indeed, cluster compute nodes make excellent grid nodes, and many grids are composed exclusively of dedicated servers.

On the Windows operating system compute clusters are supported by Windows Compute Cluster Server 2003 and grid computing is supported by the Digipede Network™. The chart below gives an overview of the two solutions.

	Windows Compute Cluster Server 2003	**Digipede Network**
Operating system	Homogeneous - Windows Server 2003 64-bit	Heterogeneous - Microsoft Windows Compute Cluster Server 2003, Windows 2000 SP4, Server 2000 SP4, XP SP2, Server 2003
Supported programming languages	C, C++, Fortran77, Fortran90, for MS MPI; any language for non-MPI	Any language that supports .NET 1.1, .NET 2.0, or COM
Development tools	Visual Studio 2005 Professional and Visual Studio 2005 Team	Visual Studio .NET, Visual Studio 2005, any IDE that supports .NET or COM interfaces
MPI	Yes	No
.NET API for distributed applications	No	Yes (1.1 and 2.0)
Distributed applications	Executables and scripts	.NET objects, COM Servers, executables, and scripts
CPU supported	64-bit only	32-bit and 64-bit
Task execution	Tightly coupled or loosely coupled	Loosely coupled
File distribution model	Staged by user, usually via scripts	Automatically staged by the Digipede Network
Compute nodes	Dedicated	Dedicated or shared
Scheduling model	Job Scheduler on head node (push model)	Agent driven (pull model)

Table 1 Clusters and Grids

Both systems use similar terminology to define submitted requests: A job defines the work submitted to the system which includes the required resources and the tasks to execute. A task is an individual unit of work that can be executed concurrently with other tasks.

Cluster Computing on Windows

Cluster computing on Windows is provided by Windows Compute Cluster Server 2003 (CCS) from Microsoft. CCS is a 64-bit version of Windows Server 2003 operating system packaged with various software components that greatly eases the management of traditional cluster computing. With a dramatically simplified cluster deployment and management experience, CCS removes many of the

obstacles imposed by other solutions. CCS enables users to integrate with existing Windows infrastructure, including Active Directory and SQL Server.

CCS supports a cluster of servers that includes a single head node and one or more compute nodes. The head node controls and mediates all access to the cluster resources and is the single point of management, deployment, and job scheduling for the compute cluster. All nodes running in the cluster must have a 64-bit CPU.

How Does It Work?

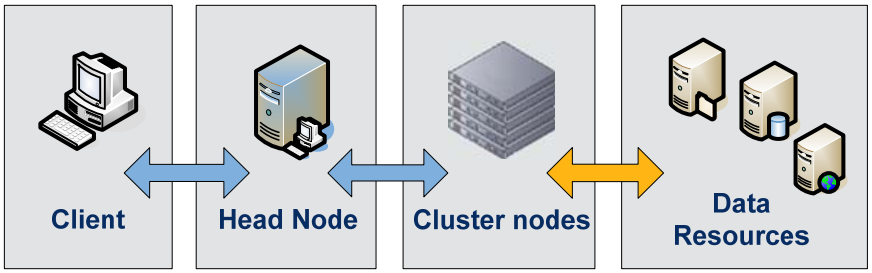

Figure 1 Compute Cluster Topology

As shown in **Figure 1**, a user submits a job to the head node. The job identifies the application to run on the cluster. The job scheduler on the head node assigns each task defined by the job to a node and then starts each application instance on the assigned node. Results from each of the application instances are returned to the client via files or databases.

Application parallelization is provided by Microsoft MPI (MSMPI), which supports communication between tasks running in concurrent processes. MSMPI is a "tuned" MPI implementation, optimized to deliver high performance on the 64-bit Windows Server OS. MSMPI calls can be placed within an application, and the mpiexec.exe utility is available to control applications from the command-line. Because MSMPI enables communication between the concurrently executing application instances, the nodes are often connected by a high-speed serial bus such as Gigabit Ethernet or InfiniBand.

Development Considerations

To understand the CCS development options, it is important to understand how CCS defines serial tasks and parallel tasks. All tasks are defined in executables or scripts. All executables must be command-line applications that do not require user interaction. Serial tasks are tasks that have no need to communicate with any other tasks. Parallel tasks communicate with the other running application instances.

Serial Tasks

Serial tasks are executables or scripts that do not communicate with concurrently running application instances; consequently they can be developed using any Microsoft language and tool.

Parallel Tasks

Parallel tasks are executables or scripts that do communicate with concurrently running application instances and require MSMPI. A parallel application cannot be written for CCS without MSMPI; language bindings for MSMPI are available for C++, C, Fortran90, and Fortran77. MSMPI development is supported in Visual Studio 2005 Professional and Visual Studio 2005 Team. At this time .NET support for a subset MSMPI functions is provided via a P/Invoke call (which allows a .NET application to make a COM interface call).

Grid Computing on Windows

The Digipede Network is a grid computing solution that provides the advantages of traditional grid solutions with additional features to simplify job creation and allow developers to grid-enable applications. The Digipede Network is a grid infrastructure which comprises a server to manage the system and many agent-nodes to execute the distributed work. The Digipede Server receives all job requests and maintains a prioritized queue of work to be done, along with a history of work completed on the system. It also guarantees execution of work on the system by monitoring the status of all Digipede Agents working on the grid. The Digipede Agent software is installed on each of the grid compute nodes; it manages that compute node's work on the grid. Although the Digipede Server receives all the job requests, the Digipede Agent decides what work can be performed on the compute node. It moves files and data as appropriate, controls the execution of the task, and returns results and status information. The Digipede Network also includes comprehensive job creation tools, detailed below.

The Digipede Server runs natively on all editions of Windows Server 2003; the Digipede Agent runs on any 32-bit or 64-bit Windows operating system since Windows 2000. As a result, the Digipede Network can be installed quickly and easily onto new or existing networked Windows computers.

How Does It Work?

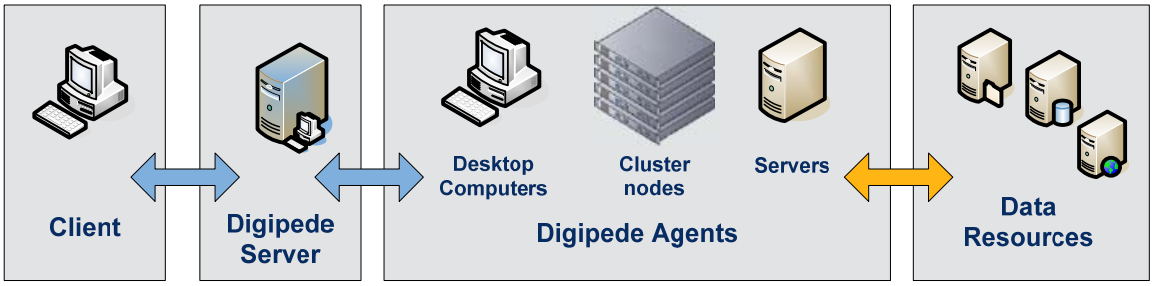

Figure 2 Digipede Network Topology

As shown in **Figure 2**, a user submits a job to the Digipede Server. The Digipede Agents check in with the Digipede Server and, if the compute node has the resources required by a job, the Digipede Agent takes a task. The executable, script, .NET object, or COM server assigned to the task is executed and the results are returned to the Digipede Server, which then returns the results to the client. Note that the distributed application may take advantage of other data resources on the network—file shares, web servers, and databases—without piping all communications through the Digipede Server.

More detailed information about the Digipede Server and the Digipede Agents can be found in the white paper *Distributed Computing with the Digipede Network™*.

On the Digipede Network, distributed application execution can be accomplished several ways. Traditional grid computing solutions invoke command-line applications on compute nodes with varying arguments and data files. The Digipede Network supports this traditional distributed grid architecture. Jobs involving command-line applications can be quickly defined and submitted using a tool called the Digipede Workbench.

The Digipede Workbench is GUI application that allows users to distribute work without writing scripts. Using wizards and design forms, the user indicates the files that need to be distributed, designates input and output files, and sets command line parameters. The Digipede Network handles the file distribution by automatically moving files to the compute node when an Agent accepts a task. For users who prefer or require command-line interaction with the server, the Digipede Network contains a complete command-line interface.

Beyond distributing command-line executables, the Digipede Framework SDK is a powerful tool that allows programmers to add the power of grid computing directly into applications. The Digipede Framework facilitates the distribution of .NET objects (or COM servers) instead of executables, allowing a programmatic rather than command-line interaction with the distributed application. With the Digipede Framework SDK programmers are able to take advantage of programming methodologies and tools that they are already trained in. The Digipede Framework SDK is described in more detail in the whitepaper entitled *The Digipede Framework™ Software Development Kit (SDK)*.

Development Considerations

With the Digipede Network the development considerations are mainly architectural. Command-line applications and scripts can be written without the Digipede Framework SDK and easily distributed using the Digipede Workbench or command-line interface.

Applications writing using the Digipede Framework initiate the distribution of work themselves and can be any type of application: GUI, command-line, or script.

Executables and Scripts

Distributable applications can be written using any Windows development environment and language. These applications must be command-line applications or scripts that do not require user interaction. The user can then use the Digipede Workbench to define jobs to quickly and easily distribute execution. The Digipede Network will distribute the appropriate applications and files and manage remote execution.

Grid-enabled Applications

Grid-enabled applications require the Digipede Framework SDK and can be written using any development environment and language that supports COM, .NET 1.1, or .NET 2.0, including all versions of Visual Studio. Because the Digipede Framework supports existing API methodologies, Windows developers can quickly and easily grid-enable applications and scripts.

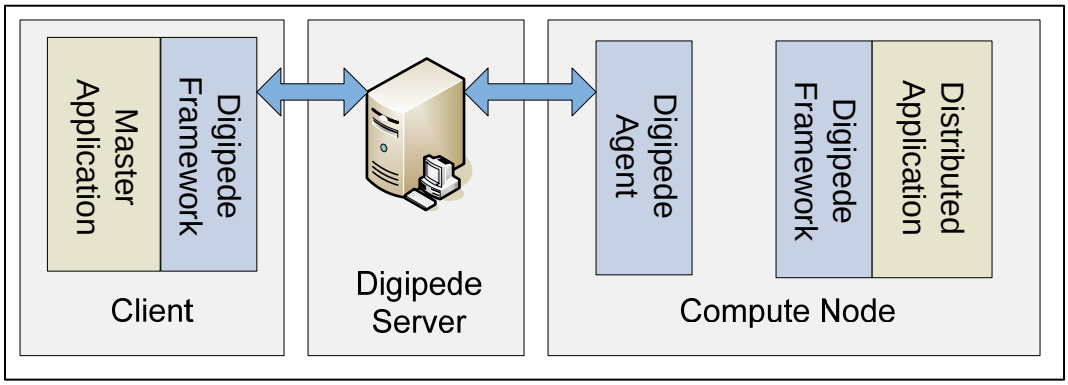

Figure 3 Digipede Framework Topology

In the client application, the programmer designates the classes that will be distributed. For each task in the job, the programmer instantiates an object from that class, initializes it, and adds it to the job. When the job is ready the programmer submits the job to the Digipede Network. Using serialization, each of the objects (and any assemblies necessary to instantiate it) are moved to the assigned compute node and deserialized by the Digipede Agent. The Digipede Agent starts the task and, when the work is completed, returns the object to the Digipede Server which then returns the object to the application.

There are many advantages to grid-enabling applications in this fashion:

- Parallelization is accomplished simply by instantiating objects from a class—the developer does not need to master complex threading schemes. Instead, by creating jobs consisting of multiple instances of objects, the parallel execution is handled automatically.
- Data is moved natively in the development language. In a typical grid system, all data moved from a client application to a distributed node must either be written to file or passed on the command line—a very limiting design restriction. By passing data with an API, the developer is able to pass data natively in their language of choice.
- Tasks are executed in parallel but the client application is notified via events so the results are processed serially—this allows for the power of parallel processing but avoids the complication of complex threading schemes in the client application.

Clusters and Grids Together

Clusters and grids solve the problem of performance improvement in different ways and each has classes of applications which it services better than the other (and, of course, there is some overlap). However, the true power and flexibility of distributed computing is realized when the techniques of clusters and grids are combined.

By running a Digipede Agent on Windows CCS cluster nodes, applications running on the grid can take advantage of the power of the cluster. When a cluster node is idle the Digipede Agent on the node will process grid tasks. The Digipede Agent monitors the CCS Job Scheduler and only accepts work for a cluster node when it is idle. If the CCS Job Scheduler assigns work to the cluster node, the Digipede Agent will stop work and defer to the cluster. The stopped work is reassigned to another compute node on the Digipede Network. This lets the cluster and grid users maximize the use of available hardware by taking advantage of the cluster when it is not otherwise in use.

Conversely, many companies that have embraced cluster computing find that their cluster systems are often backlogged with job requests. Using a grid allows the companies to offload some jobs (e.g., 32-bit jobs, or jobs that do not require MPI) from the cluster onto the grid. Running these jobs on other grid resources frees the cluster for the jobs that require cluster features. The advantages to freeing up the cluster in this fashion include completing user requests faster, utilizing available resources efficiently, and saving money on infrastructure and development.

As processing power has increased so have the processing demands of applications. Grid computing and cluster computing provide a hardware and software infrastructure that significantly increases the processing power available to applications.

References

"Develop Turbocharged Apps for Windows Compute Cluster Server"
http://msdn.microsoft.com/msdnmag/issues/06/04/ClusterComputing/default.aspx

"Deploying and Managing Microsoft Windows Compute Cluster Server 2003"
http://technet2.microsoft.com/WindowsServer/en/Library/9330fdf8-c680-425f-8583-c46ee77306981033.mspx?mfr=true

"Types of Parallel Computing Jobs"
http://technet2.microsoft.com/WindowsServer/en/Library/23afa6ab-bdaa-4c8d-9d89-44ac67196d5b1033.mspx?mfr=true

"Distributed Computing with the Digipede Network™"
http://www.digipede.net/downloads/Digipede_Network_Whitepaper.pdf

"The Digipede Framework™ Software Development Kit (SDK)"
http://www.digipede.net/downloads/Digipede_SDK_Whitepaper.pdf