Running Microsoft® Excel® on the Digipede Network™

Table of Contents:

Executive Summary

This white paper helps developers understand how the Digipede Network can improve the performance of Microsoft Excel computations. After a brief overview with definitions of concepts and terminology, the paper discusses four design patterns for working with the Digipede Network and Excel. For a general description of Digipede Network, please read "Distributed Computing with the Digipede Network;" this document assumes the reader is familiar with the information in that paper. Developers may also find the companion tutorial, "Grid-enable Microsoft Excel Tutorial," helpful.

Overview

Excel is one of the most popular business applications in the world. Within some organizations Excel is used extensively to perform computationally intensive analysis, forecasting, modeling, and report aggregation. Many of these workbooks are parallelizable: they contain computations that can run at the same time and still produce the expected results. Common examples include parameter sweeps, Monte Carlo simulations, and other calculations where the same set of computations is executed many times. Using the Digipede Network an Excel developer can significantly improve the performance of parallelizable workbooks by grid-enabling the computations, thus running the workbooks on many machines simultaneously.

This document assumes the developer is using Excel 2003. The upcoming release of Excel 2007 implements additional features that may expand the options for using Excel with the Digipede Network. Digipede will update this document after the release of Excel 2007.

There are four basic patterns for using Excel with the Digipede Network. In each pattern, the user starts a "job" on the grid from within an Excel workbook (a workbook that starts a job on the grid is called a Master workbook). The Master workbook executes on the user's machine (also called the client machine) and creates and initiates a job. Each job contains "tasks;" a task represents one computational instance (the distributed work). The distributed work that executes on the compute nodes can take the form of a distributed workbook and/or a Worker DLL.

Excel Design Patterns

The four Excel design patterns are: Distributed Workbook, Worker DLL, Hybrid Worker DLL with Distributed Workbook, and Digipede Add-In. Deciding which Excel design pattern to use will depend on your workbook architecture, development tools, and Excel installation limitations.

For each Excel workbook to grid-enable there are questions that have to be answered:

- Where are the parallel computations? Which one(s) should execute on the grid?
- What data does the computation need?
- What is being done with the results?
- Is the parallel logic in a spreadsheet, in code-behind, or in a library loaded in code-behind?
- Is Excel installed on the nodes in the compute grid?

The answers to these questions will indicate which Excel design pattern works best for you.

	Distributed Workbook	Worker DLL	Hybrid	Digipede Add-In
No coding required	No	No	No	Yes
Can be accomplished with VBA and VBS	Yes	No	No	N/A (no code required)
Excel code-behind development required? (VBA, or .NET)	Yes	Yes	Yes	No
Microsoft IDE (VB6 or Visual Studio®) required?	No	Yes	Yes	No
Excel installations required on compute nodes?	Yes	No	Yes	Yes

Pattern 1: Distributed Workbook

Distributing a workbook is the most commonly used Excel design pattern. In this pattern the developer creates a Master workbook that uses an Excel code-behind subroutine to create a Digipede Job and Tasks. The developer also creates a Distributed workbook that contains the parallelizable computation as well as any supporting macros and worksheets. For each task in the job, the Digipede Network copies the Distributed workbook to a compute node. All development is done within Excel.

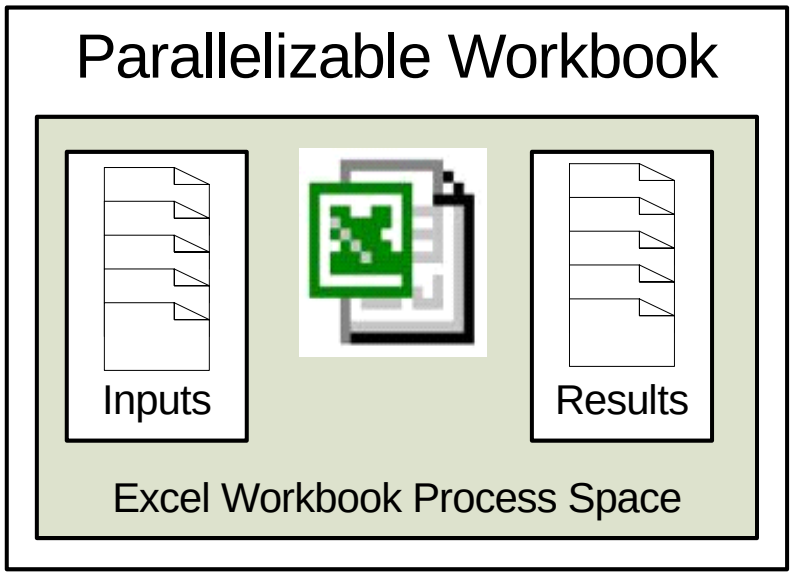

Figure 1. Basic Excel Workbook Pattern

Before a workbook is prepared for distribution it may look like **Figure 1** where there is a parallelizable computation that takes multiple inputs and produces multiple outputs. To prepare the original workbook for the Digipede Network, the developer separates the parallelizable computation from the original workbook. This creates two separate workbooks: a Master workbook and a distributed workbook. Using code-behind and the Digipede Framework SDK the developer modifies the Master workbook to define the Job and Tasks, while the distributed workbook is modified to execute a single task.

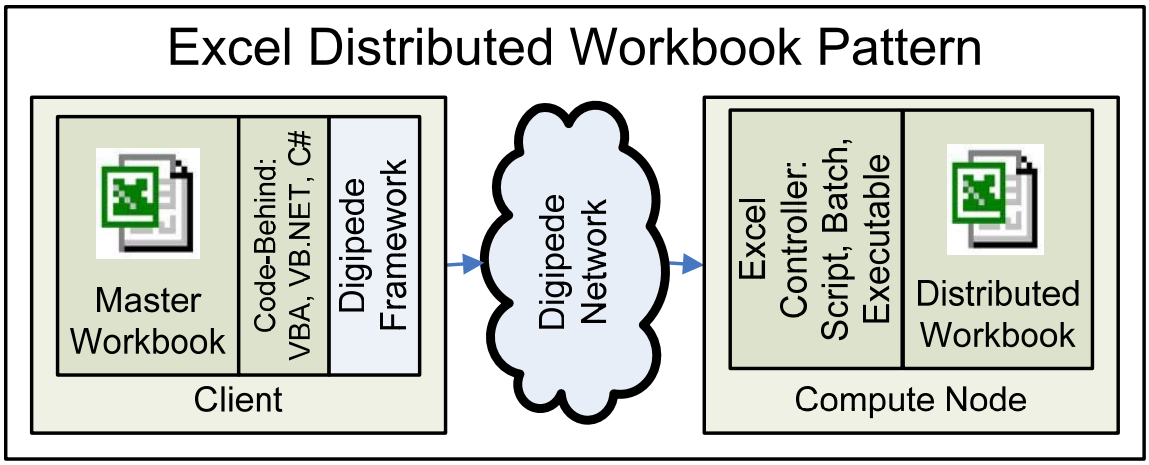

Figure 2. Basic Excel Distributed Workbook Pattern

Figure 2 contains an overview of how the Excel Distributed Worker Pattern works. The developer writes the Excel code-behind using VBA, VB.NET, C#, or any other managed language. Using the Digipede Framework the developer modifies the code-behind to create a Job and submits the Job to the Digipede Network for distribution. Each compute node checks into the Digipede Network and when work is available, executes it. Notice that the only Excel workbook linked to the Digipede Framework is the Master workbook.

In **Figure 2** each compute node uses an Excel Controller. An Excel Controller is any script, batch file, or executable that can create and manage an automated Excel COM object. Using an Excel Controller provides the developer with a powerful initialization and debugging tool, and the ability to ensure that the Excel process shuts down completely. Improper shutdown of Excel on the compute nodes will impact the compute nodes ability to process other distributed workbook requests.

Input Options

If a distributed workbook requires input from the Master workbook, Digipede Framework provides the developer with two initialization classes: Parameters and FileDefs. Parameters are strings passed to the Excel Controller on execution. FileDefs define files the Digipede Network must move to the compute node.

Parameters are sent to the Excel Controller. The developer can then use the values to update cells on the distributed workbook. To reduce bandwidth usage the developer should use Parameters when there are only a few input values. Parameters are an excellent mechanism for passing database keys into a distributed workbook. Once the distributed workbook has access to a database key, the developer can use that key to extract data from a database. For example, in **Figure 3** the developer passes Parameters to the Excel Controller. The Excel Controller updates the distributed workbook with all the Parameter values which include a database key. The database key is then used by the workbook to retrieve input values and store result values.

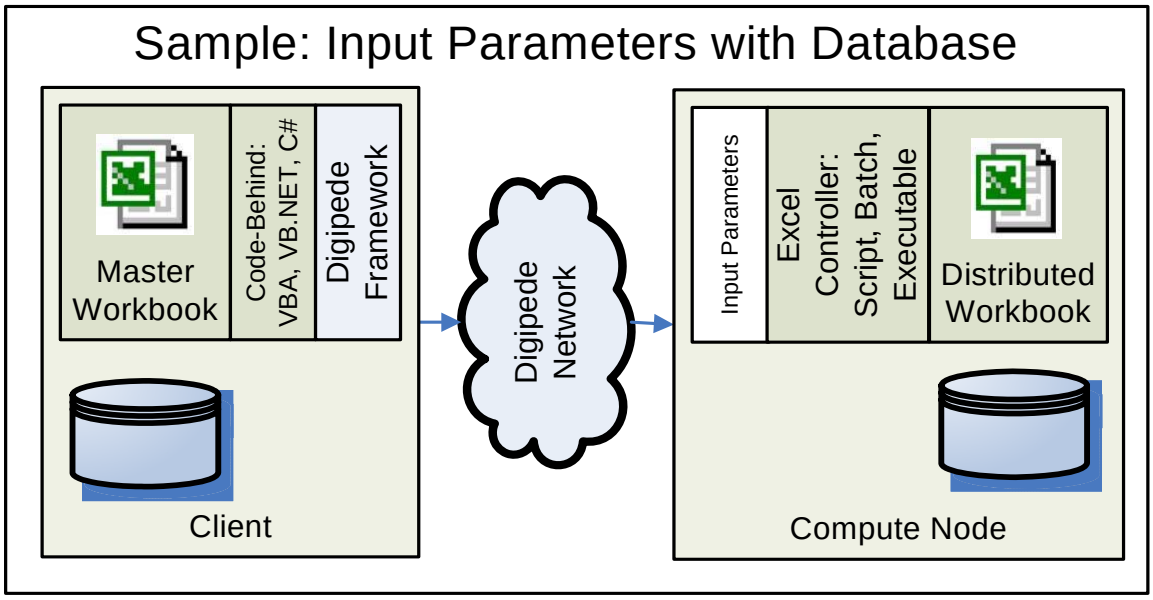

Figure 3. Distributed Workbook Pattern with Input Parameters and Database

Use FileDefs when passing data files to the distributed workbook. The files can use whatever format the developer wants. The developer can also define a FileDef such that the associated file has a unique name on the client and a standard name on each compute node. This feature allows the distributed workbook to open a hard-coded filename, which simplifies the distributed workbook's file handling. In **Figure 4** the developer has multiple input workbooks on the client machine and uses the Digipede Network to send one input workbook to each task. On the compute node the distributed workbook opens the assigned input workbook, does some processing, and produces a results workbook which is sent back to the client machine by the Digipede Network.

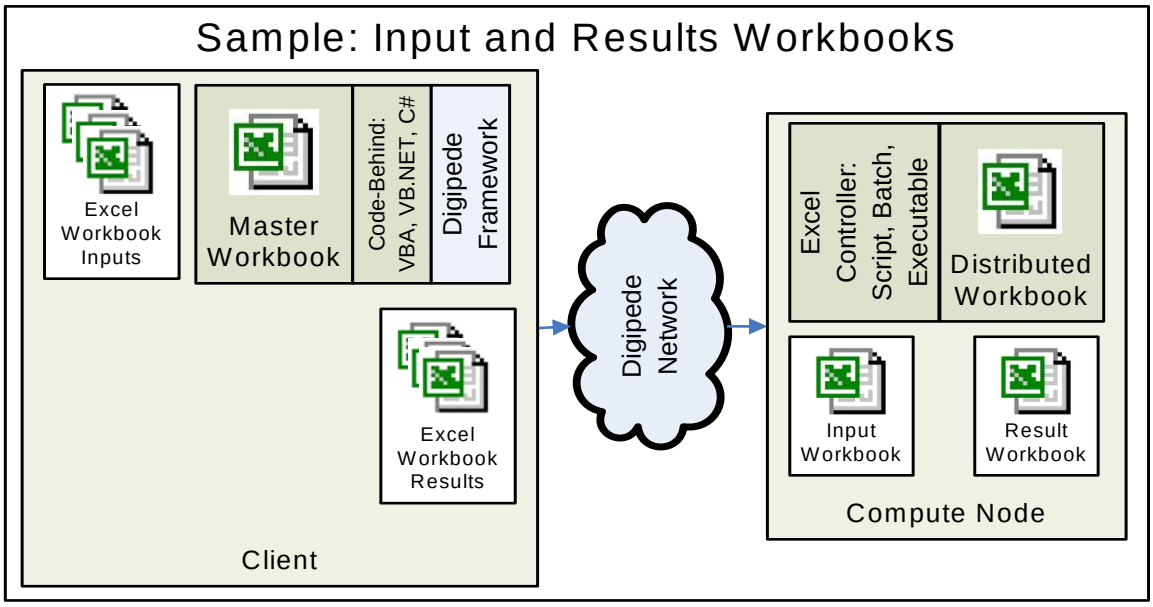

Figure 4. Distributed Workbook Pattern with Input and Result Files

Parameters and FileDefs can also be used together to initialize the distributed workbooks.

Result Options

Notice in **Figure 2** that there are no Digipede components running in either the Excel Controller or the distributed workbook. What this means for the Excel developer is that the results must be returned outside of the Digipede Network, for example saving the results in a results file or updating a database.

The developer can use the Digipede Network to return files by creating a FileDef for the result file. Just like the input files the result file can have a standard name on the compute node and a unique name on the client. If the developer is going to use a database then the database information can be passed in using Parameters.

Excel Considerations

When implementing the Distributed Workbook Pattern, Excel must be installed on all the compute nodes that will service the Job. The Professional Edition of the Digipede Network lets the developer create a resource pool composed of Excel machines and submit Excel-based Jobs only to that pool. With the Team Edition of the Digipede Network only one pool is supported, so all of the compute nodes must have Excel installed.

The Digipede Network starts the Excel Controller which runs "automated" Excel on the compute nodes. Because Microsoft Excel was designed to run interactively there are design and development constraints that must be addressed. There are two different levels of constraints: Administrative and Programmatic. Administrative constraints deal with user identity, resiliency, stability, and server-side security. Programmatic constraints deal with Excel interaction, reentrancy, scalability, and Excel idiosyncrasies.

Excel Administrative Constraints

User Identity

Excel is designed to run in an interactive manner. Because of this Excel inherits default directories and security settings from the logged in user profile. Running Excel as an automated COM server does not guarantee that the needed settings are there if the application that invokes Excel uses the system profile. On the Digipede Network the network administrator must configure each Digipede Agent on the compute nodes in the Excel pool to login using a valid user profile. See "Domain-level Agent Service User" in the Digipede Administration Guide for more information. In order for Excel to function properly, this user profile must log into the system at least once to initialize the data associated with the user profile.

Automated Excel also requires that the Digipede Agent user identity has the correct COM security settings. This must be manually set by the network administrator in the Component Services on each compute node. The user profile requires the rights to do both local launch and local activation.

Resiliency and Stability

During Microsoft Office product installation the installer has the option of performing “install on first use” for some components. If an uninstalled component is needed by an Office instance an installation wizard for the missing component is started. This causes a problem for automated Excel instances because dialogs that can’t be dealt with programmatically. The developer must ensure that any components required by the distributed workbook are available on each compute node.

Server-Side Security

The Digipede Server will not accept a Job from a Master application that does not have credentials on the Digipede Network. This ensures that only users who have been given access to the Digipede Network can request work on it.

Code level security is the responsibility of the developers. Each developer must ensure that the distributed workbook's code is secure and does not interfere with other applications running on the compute node or network.

Excel macro security must be set to Low to avoid a security dialog. This is required for running automated Excel (patterns 1, 3 and 4).

Excel Programmatic Constraints

Interactivity with the Desktop

Excel was designed around the user interface so in either the distributed workbook or the Excel Controller, the developer must programmatically disable all Excel dialogs. Digipede recommends that the Excel dialogs be re-enabled before the Excel Controller exists.

The developer must not present any user interface in the distributed workbook or Excel Controller.

Reentrancy and Scalability

Excel can sometimes close down very slowly; if the developer doesn't release all the objects, Excel may not close down at all. In addition, if the developer is using managed code (C# or VB.NET) then the developer must programmatically initiate garbage collection for the Excel object. It is the Excel developer's responsibility to make sure that the Excel process started by the Excel Controller shuts down properly before the Excel Controller exists. Failure to ensure that the Excel process closes will result in that process being left active and, because Excel does not support an unlimited number of active Excel instances, the compute node will eventually be unable to respond to Excel requests.

Excel Coding Lessons

There are also some code specific requirements for both the Excel Controller and the distributed workbook:

- When referencing a file or a macro always use a fully qualified path. There are two reasons for this. One, Excel does not use the directory the process was started in as a working directory so Excel can't find files without the complete path. And two, a call to Application.Run() looks at all the open workbooks. Without a fully qualified path to the code-behind subroutine in the distributed workbook, Excel is not able to find it.
- After completing a task, the Excel Controller should determine if there are any other Excel processes on the machine; if there are none, it should invoke Application.Quit() to close Excel. If there are any other Excel processes, the controller should not invoke Application:Quit(): A call to Application.Quit() will shut down all open instances of Excel--even those instances that are currently being used by the human user.
- Do not use Application.Calculate() because it will cause all open workbooks to calculate (even workbooks not associated with this job). Instead the developer needs to understand the worksheet calculation dependencies within the distributed workbook and make Worksheet.Calculate() calls in the correct order.
- If the developer creates an Excel Controller with a script, use a non-GUI based script launcher. For example if the script is written in VBS the developer has the option to use

either wscript.exe or cscript.exe. Because cscript.exe does not support a GUI, the developer should use cscript.exe to execute the script.

Summary

Automated Excel does have some administrative and programmatic constraints but there are solutions for each one. While the developer must be aware of the programmatic constraints when designing and writing the Excel Controller and the distributed workbook, the Digipede Network was designed to handle the administrative constraints. Constraints are discussed in detail in the "Grid-enable Microsoft Excel Tutorial: Using a Distributed Workbook."

Pattern 2: Worker DLL

Developers who have the tools to develop .NET or COM software can use the Digipede Framework API to build a Worker DLL. A Worker DLL is a dynamic linked library that contains a serializable class derived from the Digipede Worker class. Objects created from a Worker-derived class can be distributed around the grid for execution on the compute nodes. When using a Worker derived-class from Excel code-behind the developer has the option of using VBA in the code-behind with a COM-based Worker DLL or managed code (C# or VB.NET) in the code-behind with a Visual Studio Tools for Office (VSTO)-based Worker DLL.

Excel workbooks that contain code-behind computations that are not dependent on Excel-specific functionality are good candidates for Pattern 2. The parallelizable computation is moved into a Worker derived class in a Worker DLL and then the Job and distributable objects are created in the Master workbook's code-behind. Although this Pattern is less common with Excel developers because of the development tools required to create the DLL, there are some great benefits.

Benefits include:

- Excel installations are not required on the compute nodes; this eliminates the Excel automation constraints and frees you from needing Excel licenses for every node on your grid.
- Excel processes are not started on the compute nodes resulting in a performance improvement.
- Excel workbooks, which are at minimum 12K in size, are not being moved around the network thus reducing network bandwidth utilization.

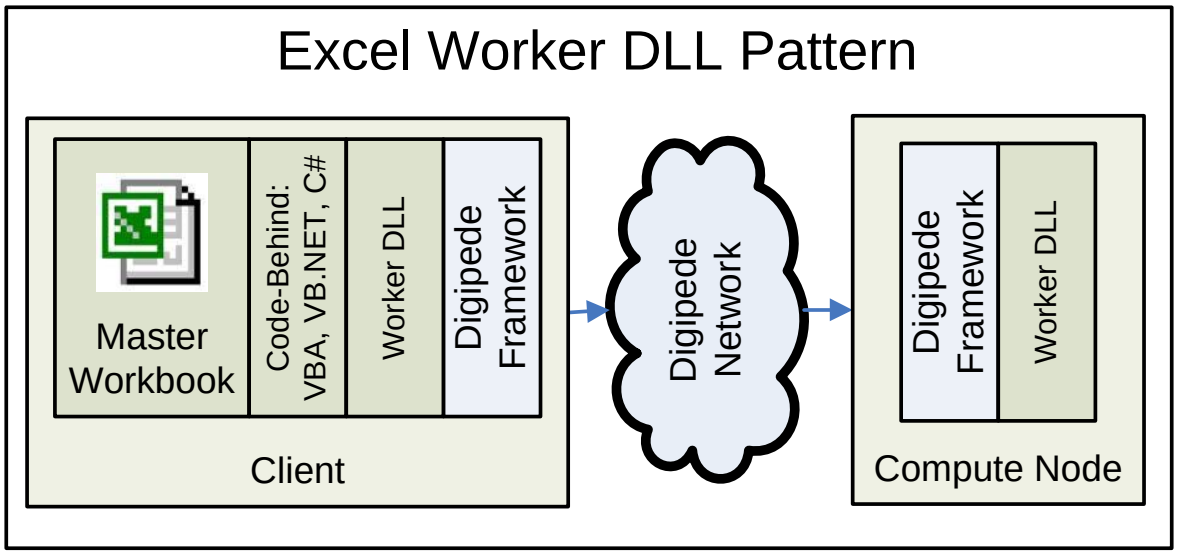

Figure 5. Basic Excel Distributed Worker DLL Pattern

Figure 5 shows the addition of the Worker DLL to the application distribution model. As in Pattern 1 the developer sets up the Job in the code behind the Master Excel workbook. For each Task the developer creates a new Worker-derived object and assigns it to the Task. When the Job is submitted to the Digipede Network the Worker-derived objects are serialized and sent to the assigned compute nodes. On each compute node the object is deserialized and the DoWork() method is called. DoWork() is a virtual method defined in the Digipede Worker class and provides an execution starting point for the distributed computation. The developer must define a DoWork() method in the derived class and can think of the DoWork() method as a main() for the distributed work.

Input Options

The Worker DLL design pattern provides the developer with three initialization options: member variables in the worker object, FileDefs, and Parameters. Member variables in the object are data structures or values that are defined in the Worker-derived class. FileDefs are Digipede objects that define files the Digipede Network must move to the compute node. Parameters are Digipede objects that are available to the object through the Worker class.

Using member variables is the most common technique for initializing Worker DLL computations, and for OOP developers it is a very familiar one. This technique allows the developer to encapsulate all the information needed for the computation within the class. Because the variables are part of the class the developer can start using them as soon as DoWork() is entered. There is no need to parse a file or search for a Parameter.

FileDefs define files that the Digipede Network will move from the client to the compute nodes. FileDefs are good for moving large quantities of data or preexisting input files. The files can use whatever format the developer wants. If the developer decides to move Excel workbooks and use automated Excel to process them, then Excel must be installed on the compute nodes. Distributing Excel workbooks also means that the developer is using Pattern 3.

Parameters are name-value pairs defined in the Master workbook code-behind that creates the Job and Tasks. By design, the Digipede Worker class contains a Task object which has a Parameter collection containing the Parameters the developer created. The developer can access the Parameters from the DoWork() method and use those values in the computation.

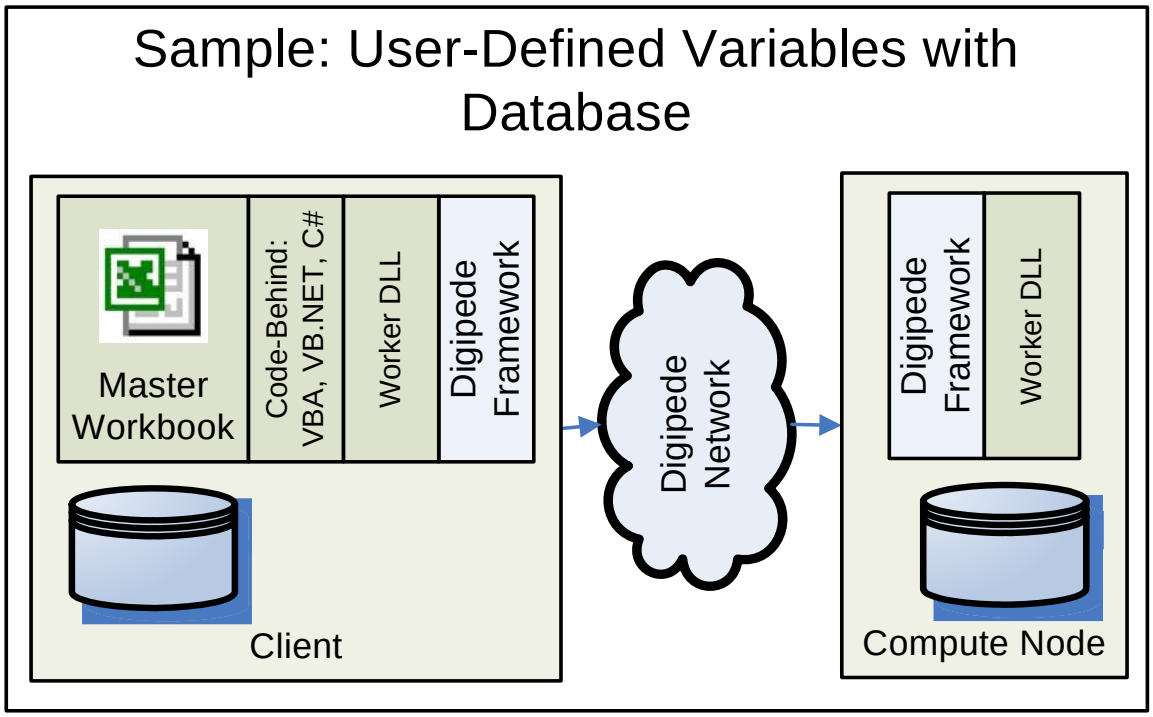

Figure 6. Sample: User-defined Variables with Database

As in Pattern 1, the developer can use all three input types in combination if required. The developer can also pass in a database key and extract computation data from a database table as shown in **Figure 6**.

Result Options

Because a developer using this pattern employs the Digipede API in the Worker DLL, she has a few more options for returning results. Following are the options for returning results to the Master workbook using Pattern 2:

- Member variables in the Worker-derived object. Returning results in the member variables is the most common technique for returning Task results to the Master workbook. Just as the developer used member variables to pass input values to the Worker-derived class, the developer can also use member variables to return results to the Master workbook. After the task has completed, the object is automatically serialized and returned to the calling application.
- Using the TaskData property. The Digipede Worker class contains a TaskResult object. The TaskResult object returns information about the Task such as exit codes to the Master workbook. The TaskResult object also contains a TaskData property that the developer can use to return a single variable. When using COM the TaskData can contain any COM automation compatible typed object, and in .NET the TaskData can be any serializable object.
- Create FileDefs to return result files. While setting up the Job and defining the Tasks the developer creates FileDefs called ResultPlaceholders that instruct the Digipede Network to move result files from the compute node to the client machine. As each Task completes the Digipede Network automatically moves the results files to the client machine.
- Use a database. The developer can store results in a database. Using user-defined variables or Parameters to pass the database information to the Worker-derived class the developer can then update a database from the DoWork() method.

Pattern 3: Hybrid Worker DLL and Distributed Workbook

Pattern 3 uses a Worker DLL as an Excel Controller for a Distributed workbook. The biggest advantage to this hybrid approach is that developer has the option to use member variables to pass data between the Master workbook and the distributed workbook. One of the limitations of Pattern 1: Distributed Workbook is that the only way to return results is by returning a file. With Pattern 3, the developer can use the member variables of the distributed objects to return results.

In **Figure 7** the developer creates a Worker DLL that is called from code-behind Excel. Within the Excel code-behind subroutine the developer creates the Job and Tasks. For each task, the Worker DLL on the compute node deserializes the object and executes DoWork(). Within DoWork() the developer uses the member variables to initialize cells on the distributed workbook, starts the computation, extracts results from the workbook, and stores the results into its member variables. The Digipede Network automatically returns the Worker-derived object back to the Master workbook when the task completes. Where the developer can easily access the results.

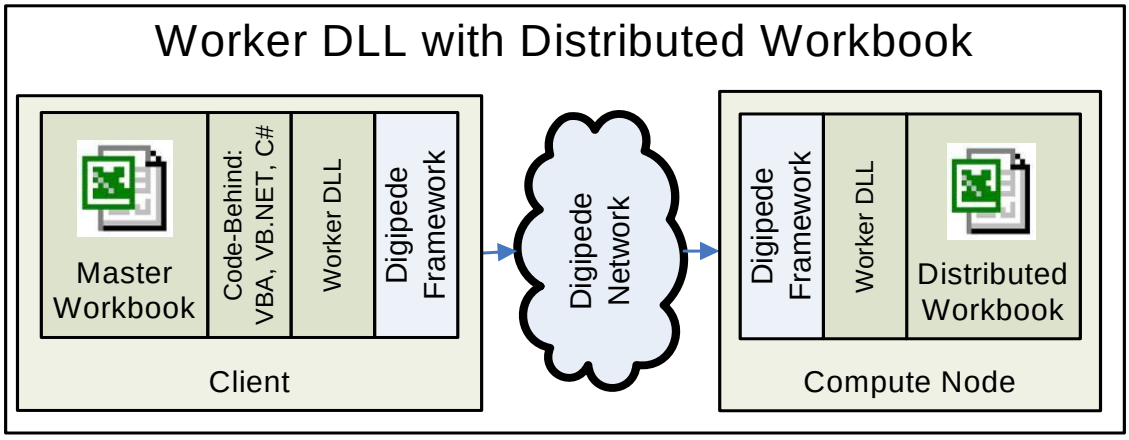

Figure 7. Worker DLL with Distributed Workbook Pattern

Pattern 4: Digipede Add-In

Digipede Technologies has created an Excel add-in that allows an Excel user to distribute workbooks without having to write any code. This Add-In can be used to distributed spreadsheets that have iterative, looping calculations. For example, a workbook may have one worksheet that has a complicated series of calculations on it and another worksheet that contains input data--each row on the input spreadsheet has a set of inputs for the calculation page. Without the Digipede Network, the user would have to write VBA code to loop through the rows on the input worksheet, copying the input data to the calculation worksheet, performing the calculation, and copying the results to a results sheet. With the Digipede Network, the data on each row on the input worksheet would be distributed to compute nodes and copied into the calculation worksheet; the results would then be returned. The user must be able to encapsulate the set of inputs and the parallelizable computation, but once that process is complete, the distributed execution is handled by the Digipede Add-In. The user does not have to programmatically create the Job or Tasks.

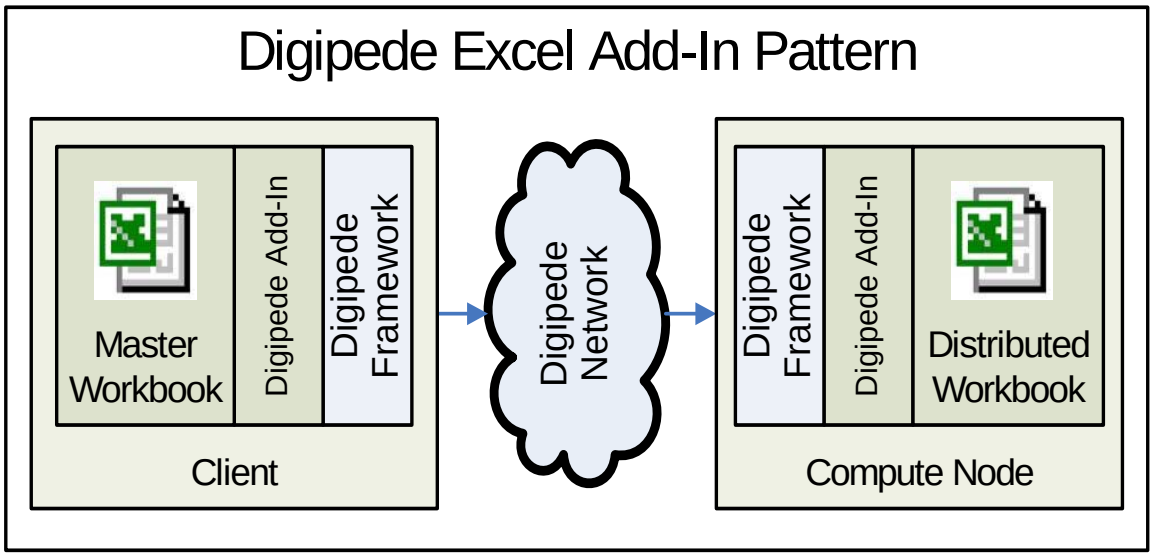

Figure 8. Digipede Excel Add-In Pattern

Figure 8 shows that the Digipede Excel Add-In acts as a bridge between the Master workbook and the distributed workbook. In the Master workbook the Digipede Excel Add-in collects input data for the Tasks from an Input worksheet. Using a form available on the toolbar in the Master workbook, the user designates the input worksheet, the calculation worksheet, and the results worksheet. When the user clicks the Submit button, the job is submitted.

On the compute node the Digipede Excel Add-in receives the input values for the Task and copies them into the worksheet. The distributed workbook then uses those values in the computation and moves the results to a Task Results worksheet. The Add-in extracts the values from the Task Results worksheet and returns those results to the Master workbook Results worksheet.

The Digipede Excel Add-in was designed for Excel users who need a code-free way to distribute Excel workbooks. To find out more about pricing and availability of the Digipede Excel Add-In please contact sales@digipede.net.

Conclusion

The Digipede Network can increase your company's ability to respond to business changes, reduce risk, and increase productivity. With usage patterns ranging from .NET development to VBA to a code-free Excel Add-In, the Digipede Network has the power and flexibility to handle many different scenarios. The Digipede Framework gives Excel developers the flexibility to distribute computations in the manner that best suits their business requirements.

References

To learn more about the Digipede Network and Digipede Technologies visit us on the web at www.digipede.net, signup for a free webinar, or contact our sales department and get started today.

Digipede Documents

"Distributed Computing with the Digipede Network"
"The Digipede Framework™ Software Development Kit (SDK)"
"Grid-enable Microsoft Excel Tutorial: Using a Distributed Workbook"
"Digipede Network Administration Guide" (installed with the Digipede Server)

"Digipede Developer's Guide" (installed with the Digipede Framework SDK)
"Digipede Samples Guide" (installed with the Digipede Framework SDK)

Microsoft Documents

"Considerations for server-side Automation of Office"
"How to Dismiss a Dialog Box Displayed by an Office Application with Visual Basic"
"Office Application does not quit after automation from Visual Studio .NET client"
"Running Subroutines and Macros from Visual Basic"
"Description of the startup switches for Excel"

Glossary

This is a partial glossary of terms used in this document.

- **Client machine**: A machine on which a **master workbook** is running; any machine on which a user is running or monitoring **jobs** on the **Digipede Network**.
- **Compute resource**: A machine on which the **Digipede Agent** is running; a machine that may execute **Tasks** for **Jobs** running on the **Digipede Network**.
- **Digipede Agent**: The software component that manages the compute resource for the Digipede Network. This is a small, unobtrusive program that does not require any interaction with any user of the compute resource.
- **Digipede Control**: The administrative component of the **Digipede Network**. **Digipede Control** is a website (usually hosted on the same computer as the **Digipede Server**) through which an administrator can monitor and administer the **Digipede Network**, and through which users can start and monitor their **Jobs**.
- **Digipede Framework**: The component that supports building and running distributed applications on the **Digipede Network**. It provides a hierarchical set of class libraries that allows developers to grid-enable .NET applications, and also allows for distribution of COM and non-.NET software.
- **Digipede Network**: Software for managing distributed applications across Windows computers. The components include the **Digipede Server**, the **Digipede Agents**, the **Digipede Framework**, **Digipede Control**, and the **Digipede Workbench**.
- **Digipede Server**: The software that manages jobs and all communication with the **Digipede Agents**.
- **Distributed workbook**: An Excel workbook that an Excel Controller manages for execution on **compute resources**.
- **Excel Controller:** Any script or assembly that the **Digipede Agent** manages for execution on compute resources. The **Excel Controller** controls the execution of the **distributed workbook**.
- **Job**: Contains the details for a specific run of a **JobTemplate**, and it is composed of one or more **Tasks**. The **Job** and its **Tasks** completely specify all details necessary to run a job against a job template: the files (by way of **FileDefs**) and the Parameters. It also contains any overrides of the **JobDefaults** values.
- **JobTemplate**: Contains information about the overall structure of a distributed application. This includes the files that will be installed on a **compute resource** to run the job and information about how to start and monitor the **Job**.

- **Master workbook**: Any Excel workbook that runs jobs on the **Digipede Network.**
- **Pool**: A collection of **compute resources** on which jobs are run. **Jobs** are submitted to individual pools, so pools are an effective way to control which **compute resources** work on which jobs. Digipede Network Team Edition features one pool; the Digipede Network Professional Edition allows for an unlimited number of pools. **Compute Resources** may belong to more than one pool.
- **Task**: The part of a **Job** that is run on an individual **compute resource**. Most **jobs** are composed of many **tasks**.