Introduction

Digipede customers have expressed interest in distributing calculations done in Matlab across a grid of high-performance workstations to deliver dramatically improved throughput. This scenario is quite familiar to Digipede; some of our customers already use the Digipede Network to distribute Matlab workloads today, with excellent results. Below we describe the process. Far more detail on development patterns referenced below is available in the Digipede Framework SDK (included with every edition of the Digipede Network).

Distributing Matlab DLLs on the Digipede Network

While there are several methods that can be used to distribute Matlab calculations on a Digipede grid, we recommend using Matlab .NET Builder to create DLLs containing the relevant calculations, and then using Digipede's Worker Pattern for an application that manages execution of those calculations on the grid. This pattern depends on the Matlab Compute Runtime being installed on each compute node on the grid.

Worker Pattern

Digipede's Worker Pattern allows the .NET developer to create a class that encapsulates all data and code to be distributed on the Digipede Network. The Digipede Network will distribute the assemblies (executables and DLLs) necessary to work on the job to each node, and then will distribute individual objects instantiated from the class to individual compute nodes for computation. See the Digipede Developers Guide for more information about this pattern.

Steps to invoke the Matlab computation.

Create a .NET DLL from your M code	Using the Matlab Builder for .NET, create a .NET DLL that contains the function(s) you want to distribute.
Create a Worker class	In the .NET application that will launch the job on the grid, derive a class from the Digipede Worker class. Ensure that this class has member variables representing the inputs to and outputs from your Matlab logic. If the application is a large application like a GUI, it may be advisable to create this class in a DLL (Worker Library pattern); if the application is smaller (a simple console app, for example) the class may be implemented in the .EXE itself (Worker Executable pattern).
Add a reference to the Matlab DLL	In the assembly that implements your Worker class, add a reference to the .NET DLL you built from Matlab. Override the Worker.DoWork method, and instantiate and invoke your Matlab method in that override. Store the results in member variables.

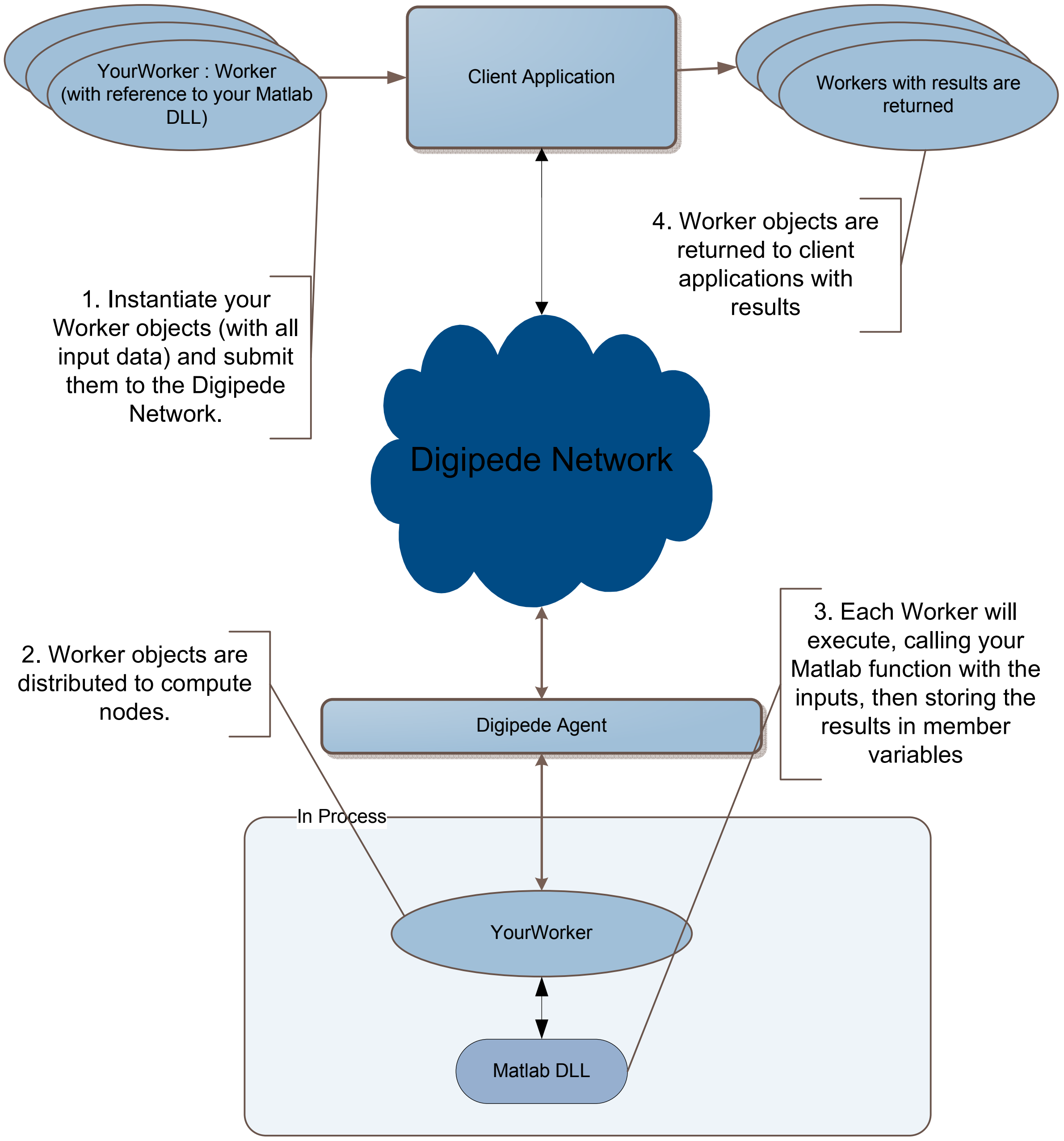
In order to launch the job on the grid, you would do the following:

Create a JobTemplate	A JobTemplate describes the files that need to

	move to work on the job. Use NewWorkerJobTemplate() to create a JobTemplate based on your Worker type. This will automatically detect the dependency on your Matlab DLL, so that will get distributed to the compute nodes with the job. Add a FileDef representing your CTF file to ensure that it gets distributed as well.
Create and submit the Job	For each task in the job, instantiate an object from your Worker class and set up the member variables with the correct inputs.
Monitor events	As tasks complete on the compute nodes, the TaskCompleted event will be raised in your client applications. The Worker objects will be passed into those events, with the results of their calculations in their member variables.

The following diagram shows how the different components work together with the Digipede Network.

How it works


YourWorker : Worker
(with reference to your Matlab DLL)
Client Application
Workers with results are returned
1. Instantiate your Worker objects (with all input data) and submit them to the Digipede Network.
4. Worker objects are returned to client applications with results
Digipede Network
2. Worker objects are distributed to compute nodes.
3. Each Worker will execute, calling your Matlab function with the inputs, then storing the results in member variables
Digipede Agent
In Process
YourWorker
Matlab DLL