# A DSL for Cluster Computing (DSL4CC)

## Glossary

Host

Computer used to load and run the application; it does no work apart from loading the application and ensuring that it terminates correctly and recovers all the resources used.

Cluster

A group of nodes all undertaking the same operation on different objects. The sequence of clusters forms an execution pipeline.

Node

A computer which runs part of the application, one or more nodes are used to create a cluster. The number of nodes in each cluster in the application can vary.

Worker

A process running in parallel on a node. All workers in a node (hence also cluster) undertake the same method. All nodes in a cluster have the same number of workers.

Manager

A Manager processes are invoked by the host such that one Manager deals with outputs from one cluster and inputs to the following cluster. A Worker process requiring work informs the Manager and it uses this to inform a Worker in the previous cluster to send work accordingly. The requests for work by inputting Workers are queued such that as work becomes available the output Worker can send it directly to the Worker inputting the object.

## Process Diagrams

### Worker

Worker - index

inputWork

outputWork

requestWork

toHost

sendTo

fromHost

requestIndex

*in-end*

*out-end*

All channels are NetChannels.

The outputWork channel will be dynamically connected to the inputWork channel of a Worker in the next cluster. The combination of Workers in adjacent Clusters and their associated Managers will be organised to form a crossbar switch between any of the outputWork channels in one Cluster to any of the inputWork channels of the next Cluster. The shared toManager channel is used in the initialisation phase only.

The shared requestWork channel is connected to a Manager process and is used to send the index of the Worker that is ready to read a new input object from the preceding cluster. The shared requestIndex channel is used to indicate that a Worker is ready to send work to the next cluster. The one-to-one sendTo channels are used to read the index of a Worker. The Worker processes and Manager process are in a client-server relationship.

The operation of Workers and the Mangers is as follows:

On completion of a work method or when the Worker is initially ready the process sends its index to the Manager using the requestWork channel, which will be read immediately. The Manager forms a queue of such index values.

The Manager will receive a Worker index on the requestIndex channel, provided the Manager has a queued index of a process in the next cluster that has made a requestWork communication. It will then respond, immediately, with the first or only index in the queue, using the useIndex channel with the subscript the same as the index value just read from the requestIndex channel.

### Manager Process

Manager

useIndex  
nodes\*workers

requestIndex  
vcn = outManagerIndex

requestWork  
 vcn = inManagerIndex

toHost

fromHost – internal channels

toOutNode

*out end*

*in-end*

Channels toHost and fromHost are internal channels with the Host process The Manager processes are run within the Host process using ProcessManager. The remaining channels are NetChannels.

The Manager process receives the Structure object as a parameter when it is constructed. The toHost and fromHost channels are used to send acknowledgements to and from the Host as necessary.

A Worker process sends its index to the shared requestWork channel when it has completed processing an object and is ready to process the next one. A Worker asks for an index using the shared requestIndex channel and receives the response on the useIndex channel when it has completed processing an object and is ready to send it to a Worker in the next cluster. Thus, readyToSend is connected to the requestIndex channel. The useIndex channel is connected to the sendTo channel of one of the Worker processes. The Worker sends its index to the Manager which returns the index of a Worker, in the next cluster, that requires more work.

During initialisation, the requestWork and requestIndex channels are used as a shared channel by the Nodes to transfer channel address location data.

### Node Process

A Cluster comprises several Nodes. Each Node comprises the same number of Worker processes. The Node process starts each Worker using the ProcessManager capability so that when all the Worker processes terminate this can be automatically registered by the Node. It is the user’s responsibility to initiate the required number of Nodes. Node processes must be invoked after the Host process. A Node process must be passed the IP address of the Host as an argument when it starts.

fromManager vcn = 2

Node

toHost

fromHost vcn = 1

During initialisation a Node will create its fromHost channel and will connect to the shared toHost channel. The Node process will then send its IP address to the Host so it can create the fromHost connection to each Node. The Host can then send the application structure to the Node which can commence initialisation of the application channels. Internally, the fromHost and toHost channels are passed as parameters to each of the Worker processes in the Node. These channels are Code Loading channels which means that classes required by any of the Workers can be obtained by the Worker processes in the Node. The fromManager channel is used to inform the node that its Manager process(es) are running and to transfer channel data, it is only required for Emit and Work node types that have an out-end.

### Host Process

The Host process is invoked by the DSL4CCbuild script. Initially the script reads in the file containing the parsed specification, extracts sufficient information and then invokes the Host process. The computer running the Host process must have access to all the class files used by the application.

Host

fromNodes vcn = 1

Initially the Host process only has a net input channel at virtual channel number 1 to which the Nodes as they are started can connect. The Host can then create code loading net channels to each Node using the IP string each node sends as its first communication to the Host.

### DSL4CC Specification Rules

The specification is written in the order the data objects flow through the clusters. Values supplied by the user are shown *italicised*. Optional values are shown between []. Repeated entries are shown {}*repetitions*. The lines and arguments MUST be presented in the order specified.

**host -ip** *IPaddress* the IP Address of the host machine

**emit** **-n** *nodes* the number of nodes in the emit cluster

**-w** *workers* the number of workers in each node

**[{-ip***n* *IPaddress*}] the optional fixed IP address of each node (*n* = 1 .. *nodes*)

**-c** *className* the name of the class used to create application objects

[**-p** *space-separated-parameter-values*] parameter values for the single class constructor

The ***work*** line is repeated for the number of required work clusters

**work -n** *nodes* the number of nodes in this work cluster

**-w** *workers* the number of workers in each node

**[{-ip***n* *IPaddress*}*nodes*] the optional fixed IP address of each node (*n* = 1 .. *nodes*)

**-m** *methodName* the name of the method used to modify application objects

[**-p** *space-separated-parameter-type value pairss*] parameter type and value

The number of nodes in different work clusters does not have to be the same.

**collect -n** *nodes* the number of nodes in the collect cluster

**-w** *workers* the number of workers in each node

**[{-ip***n* *IPaddress*}*nodes*] the optional fixed IP address of each node (*n* = 1 .. *nodes*)

**-m** *methodName* the name of the method used to collect application objects

[**-p** *space-separated-parameter-type value pairss*] parameter type and value

The argument prefixes can be extended as follows:

n(odes), w(orkers), c(lass), m(ethod) p(arameters)

### An example specification

host -ip 127.0.0.1  
emit -nodes 1 -workers 1 -c BuildTests.EmitTest1 -p int 100  
work -n 2 -w 4 -m updateMethod -p int 500  
collect -n 1 -w 1 -m collect

The above specification creates an application that has one work cluster comprising 2 Nodes of 4 Worker processes which undertake the method testMethod on the application object. The Emit and Collect nodes comprise a single node each with 1 Worker process in each Node. The class used to create application objects is called TestEmit and has the single parameter 100. The TestEmit class MUST have a method called create() which returns a new application object. The Emit node must be run on the computer with IPAddress 192.168.1.68. The methods testMethod and testCollect must be methods contained in the application object. The Work and Collect nodes can be allocated to any available computers in the 192.169.x.x network.

### Specification of a Concordance Application

## Parsing the Specification

The specification is parsed by DSL4CCparse script. It takes a specification in a file name.dsl4cc. The parser outputs a file called name.dslstruct.

## Running the Specification

The specification is invoked using the DSL4CCbuild script. It reads a file produced by DSL4CCparse, extracts the structure, and asks the user to create the required number of nodes, indicating any nodes that must be used. The invocation comprises three distinct phases: initialisation, execution, and termination.

### Phases of Initialisation

Using the example specification, the following phases are invoked.

#### Phase 1 Initialisation of the Nodes

The required number of nodes are started including using specific computers (IPAddresses) as required. Each node creates a net input channel, which it refers to by the name fromHost. Each Node sends its IP address to the Host using the shared toHost net channel. Using the IP addresses the Host can create net output channel to each of the nodes, ensuring that any node requiring a specific IP address is satisfied. The host can then send the application structure to each of the nodes using the newly created toNode channels (shown dashed). The Node processes can then create all the required net input channels.

Host

Emit Cluster

Work Cluster

Collect Cluster

Node

Node

Node

Node

toHost

This needs to be updated as per the first working version of the system to be known as v1.0.0. This version uses the pcicli parser to process the DSL specification, but is somewhat limited as identified below.

Version 1.0.1 will have the following changes.

The Collect process will use a dedicated object to collect resultant data objects, using a collect method, which can have parameters and a finalise method, also possibly with parameters, that is invoked once all the data objects have been read, ie once the termination signal has been received.

It is also proposed to introduce a different variant of the Work process. Currently, it is a single phase process in that the input object is modified and then output. The other option would be a 3-phase mechanism whereby several data objects could be input, or until termination, then processed and the result could then be output possibly creating a new object. In other words, we have a three-phase mechanism of; input some data, process it, and then output it, possibly as a revised data object. The mechanism can be repeated several times if the amount of input data is very large. This may lead to randomising of the input data stream, especially if it is being input from several Emit processes, but in many applications that will not be such a large problem if some form of machine learning or data mining is being undertaken.