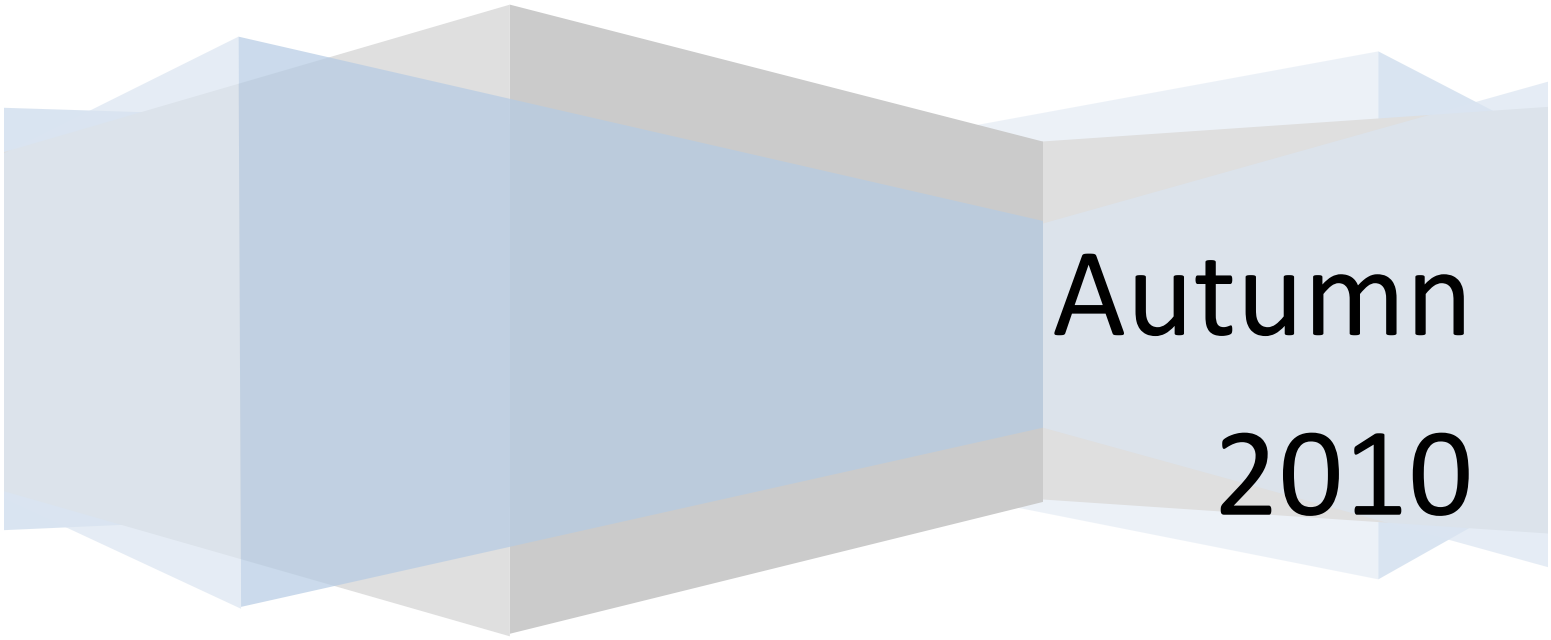
University Of Washington Bothell
Computing Software & Systems

MASS Library

Multi-Process Version

John Emau



Autumn 2010

Table of Contents

Overview

Description

The MASS Multi-Process Library was designed to implement the MASS specification using multiple spawned processes on remote computer nodes through Java's JSCH. Each Node represents a unique process located on a remote machine. An array of Place objects is divided amongst the nodes and thus computational work done by all the Place objects is divided amongst the processes, improving computational performance of any time and memory consuming computational task.

Use

To use the MASS Multi-Process Library import MASS in your program and follow the MASS specification in your code. At compile time and runtime make sure the MASS class files are included in the classpath as well as Jsch class files. Please see the descriptions below for more detailed design notes on each class. Please see API Section for more.

Example code running applications from the command line:

java -Xmx1g -cp ../MASS-working:../jsch-0.1.42.jar <Application> < Application Parameters>

Performance and Development

Performance

Performance of the MASS Library has been tested periodically during development in an informal method using the Wave2D application as a test base. The overall performance of MASS has been scaling negatively with increased processor, this is the opposite of what the library is intended for. The cause is was a combination of caching issues and ExchangeAll implementation.

Handler Implementation

To address the caching problem a new API was created that used primitive data types to represent the place objects. This allowed for better caching on smaller CPU cache machines (1-2MBs)

Multi-Threaded ExchangALL

To address the ExchangALL problem a new implementation was devised to use multi-threads and sockets for inter-process communication. This will take the place of the relayed communication through process 0 in the current implementation.

API

The full MASS API can be found at the following web link:
http://faculty.washington.edu/mfukuda/doc/MassSpec_050710.doc

Please note this is a working version of MASS and the API changes along with development; do not consider the provided API as complete or current unless otherwise specified. This report will focus only on the API currently implemented in the Multi-Process Version, they are as follows:

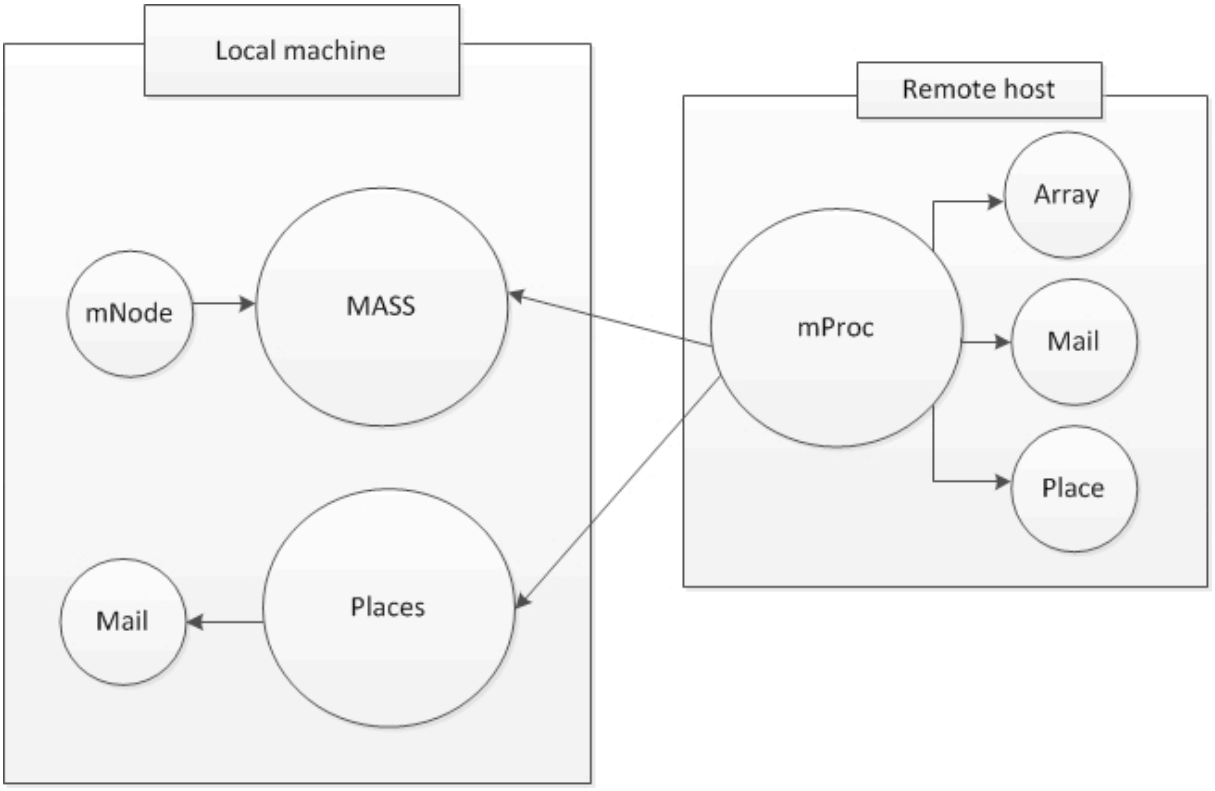
Mass

public static void	init(String[] args) Involves as many processes as requested in the same computation and has each process spawn as many threads as the number of CPU cores.
public static void	finish( ) Finishes computation.

Places

Public	Places(int handle, String className, Object argument, int… size) Instantiates a shared array with “size” from the “className” class as passing an Object argument to the “className” constructor. This array is associated with a user-given handle that must be unique over machines.
public void	callAll(int functionId) Calls the method specified with functionId of all array elements. Done in parallel among multi-processes/threads.
public void	callAll(int functionId, Object argument) Calls the method specified with functionId of all array elements as passing an Object argument to the method. Done in parallel among multi-processes/threads.
public Object[]	callAll(int functionId, Object[] arguments) Calls the method specified with functionId of all array elements as passing arguments[i] to element[i]’s method, and receives a return value from it into Object[i]. Done in parallel among multi-processes/threads. In case of a multi-dimensional array, “i” is considered as the index when the array is flattened to a single dimension.
public void	exchangeAll(int handle, int functionId, Vector<int[]> destinations) Calls from each of all cells to the method specified with functionId of all destination cells, each indexed with a different Vector element. Each vector element, say destination[] is an array of integers where destination[i] includes a relative index (or a distance) on the coordinate i from the current caller to the callee cell. The caller cell’s outMessage, (i.e., an Object) is a set of arguments passed to the callee’s method. The caller’s inMessages[], (i.e., an array of Objects) stores values returned from all callees. More specifically, inMessages[i] maintains a set of return values from the ith callee.

Class Relations Diagram



MASS

init( )

init is called early on in the program using the MASS library, and must be the first method called from the MASS Library because it initialized all the nodes which will later be accessed. init takes in a string array that is meant to have originated from the console, the first three elements must be Username, Password, and Host File respectively. Init( ) will read the Host File and create a mNode object for each host, if a host specified in the Host File cannot be connected to by JSCH the process will end and all already established host will close their connection. Each node is stored is an array of mNodes, this array is accessible to the public and used when other classes need to communicate with a particular or all the nodes.

finish( )

finish( ) contacts each node in mNode and sends a command for the node to suicide, this allows the node to perform any closing operations before the connection is terminated (terminating the connection will kill the remote process). After the node has competed its closing operations it will send a one dimensional byte array to signal its completion and self-terminate. At this point the Nodes underlying stream objects will be closed.

mNode

This sub-class is a wrapper for the JSCH connection established during MASS.init( ). The mNode has several useful data members- channel gives access to the underlying communication channel to the process; in and out are Object streams created from the channel and used for convenient writing of Objects to and reading from the node, we use Objects streams because of the large number of objects we must send and receive between the nodes during communication; Lower and Upper represent the lowest and highest element this node is responsible in the Place array, these values are used for sorting the nodes by Place object. The mail vector is a collection of Mail objects that need to be sent to this node, this is used only during the Places.exchange commands.

Places

Places Constructor

Instantiates a shared array with "size" from the "className" class as passing an Object argument to the "className" constructor. This array is associated with a user-given handle that must be unique over the machine. The array is first partitioned by two private functions, createBoundsArray( ) and createRangesArray( ), the Bounds represent the division of elements per-dimention and the ranges represent the number of elements each node is responsible for. Then each node is called and issued a command to start array creation and assigned a lower and upper bound.

Void callAll( )

Calls the method specified with functionId of all array elements. Done in parallel among multi-processes. This is achieved by in order accessing each node and issuing the callAll command, passing any arguments to the node for the callMethod, then waiting to receive the completion confirmation in the same order.

Object[] callAll(..., Object[] Arguments)

Calls the method specified with functionId of all array elements as passing arguments[i] to element[i]'s method, and receives a return value from it into Object[i]. Done in parallel among multi-processes/threads. In case of a multi-dimensional array, "i" is considered as the index when the array is flattened to a single dimension.

This is done in the same way as in the callAll method described above only now an additional smaller array is created of the arguments assigned to the places in that specific node, this small array is sent over and a small array of the same size is sent back with the respective return values.

exchangeAll( )-- Old implementation replaced in Multi-Threaded version

Calls from each of all cells to the method specified with functionId of all destination cells, each indexed with a different Vector element. Each vector element, say destination[] is an array of integers where destination[i] includes a relative index (or a distance) on the coordinate i from the current caller to the callee cell. The caller cell's outMessage, (i.e., an Object) is a set of arguments passed to the callee's method. The caller's inMessages[], (i.e., an array of Objects) stores values returned from all callees. More specifically, inMessages[i] maintains a set of return values from the i^{th} callee.

This is done in the following order:

1. Send ExchangeALL command to all nodes with functioned and Vector Destinations parameters.
2. Receive call request from all nodes. Each mail message represents a request to perform a callMethod on a remote process to be performed by the remote process on behalf of the caller.
3. Sort call mail by reviver
4. Send call mail to all nodes
5. Receive responds mail from all nodes
6. Responds mail represents the return message from the previous Call mail
7. Clear out nodes mail box for reuse
8. Sort responds mail by receiver
9. Send responds mail to all nodes
10. Loop through all nodes and get confirmation

Mail

This is a wrapper for a message to be sent between nodes during the ExchangeAll command. It contatains the message as an Object and useful information about where the message is going and where it is from.

mProc

This is the remote process that is launched during the JSCH connection creation. This process facilitates all commands issued by the MASS library. After creation and establishing of the Object Input/Output streams mProc sits in an infinite loop waiting to receive a command issued by the MASS library through the standard input. The command must be a string written using the ObjectOutputStream. Once a command is read a sequence of actions is performed. Error logging is written locally and if any exception is thrown it is written to the error log and the process terminates closing the JSCH connection.

Array

This class contains a multi-dimensional array of objects. You can specific the lower and upper bound of the array and it will only create elements within those bounds. The Array class has a Vector if indexes for each element it is responsible for, it also has a second multidimensional array of Integers that represent the element number of any given index in the array.

Credits

John Emau, John Spiger, and Munehiro Fukuda (CSS, UW Bothell)