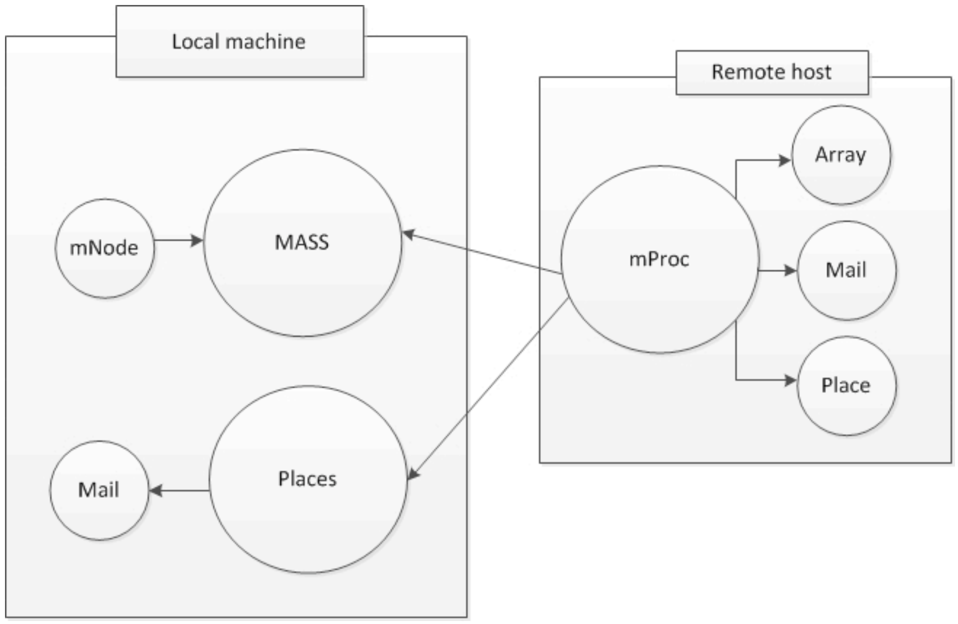
Implementation

MASS & MProc Implementation

Running a MASS application involves three main steps (1) Dynamic initialization of the cluster, (2) Sending some commands to the cluster, (3) and deconstructing the cluster. The two important classes are MASS and MProc, they manage the inter-process communication and program flow. The MASS class only exists on Node 0, or Process 0, or the Master Node. MProc then exists on all remote machines and represents a single node in a MASS Cluster (it is possible to run multiple MProcs on a single machine).

Class Relations

The following describes the relationship between classes and machines in the MASS Cluster.



Initialization

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.

Deconstruction

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.

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.

Places Implementation

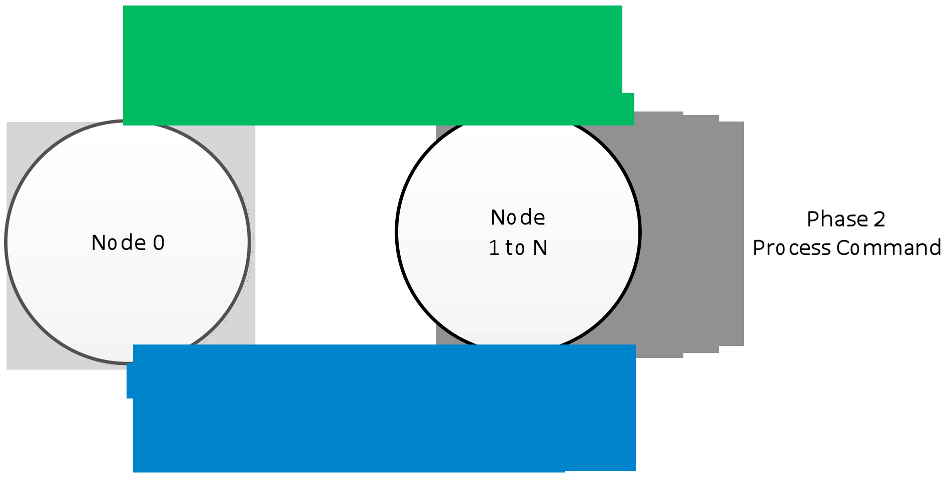
Instantiates a shared array with “size” from the “className” class as passing an Object argument to the “className” constructor. This array is associated with a user-given handle that must be unique over the machine. The array is first partitioned by two private functions, createBoundsArray( ) and createRangesArray( ), 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.

Primitive vs. Object

Call Method

Calls the method specified with function Identifier of all array elements. This is 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 the MASS process waits to receive the completion confirmation in the same order. Alternatively a node may send data as a return value as confirmation. The following Figure provides a visual of the process.

Figure 2. Call Method



Exchange Data (Old Implementation)

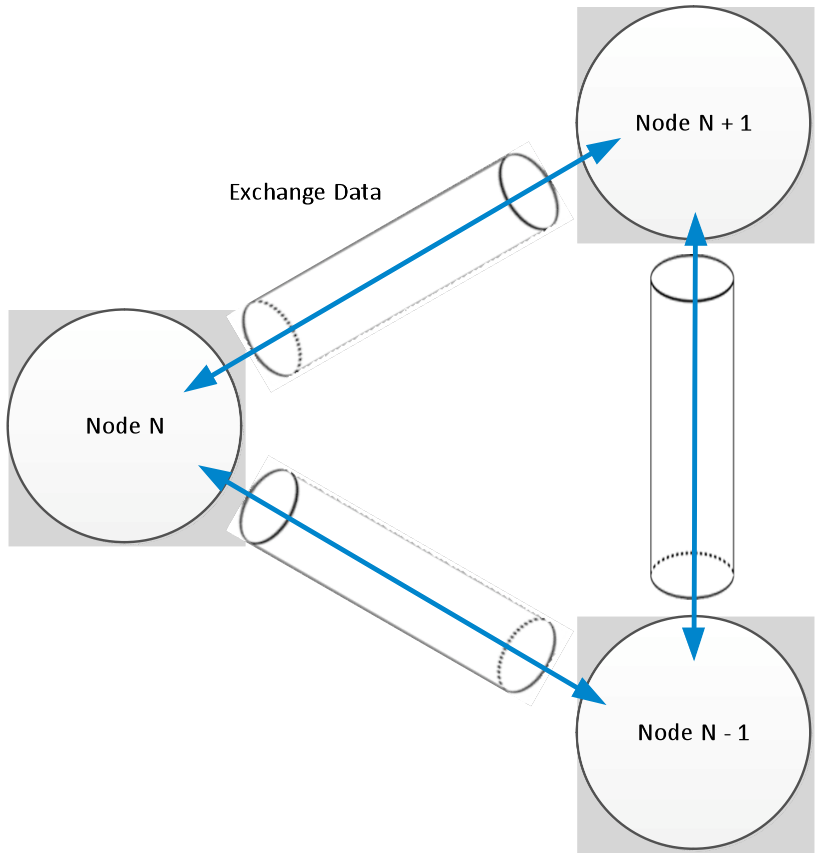
Calls from each of all cells to the method specified with functionId of all destination cells, each indexed with a different Vector element. Each vector element, say destination[] is an array of integers where destination[i] includes a relative index (or a distance) on the coordinate i from the current 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

The following figure 3 is a visual representation of the new implementation of Exchange All.

Figure. 3 Exchange Data



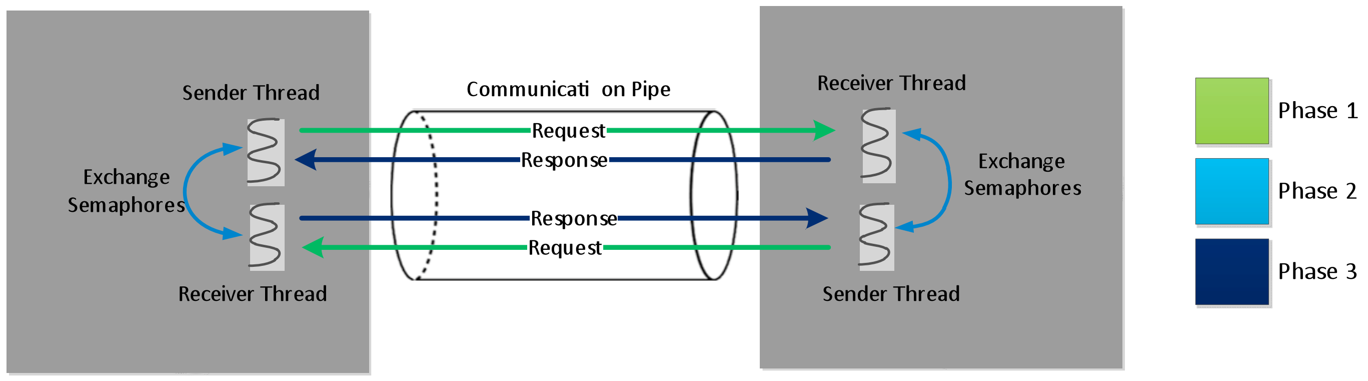
This simplistic view of Exchange Data gives the idea what is achieved by the function. All communications is done near-asynchronously using threads and a single socket per-connection pipe. The reason it is only near-asynchronously is due to hardware limitations such as a single network card/connection, possibility of a single core node and various algorithm limitation.

The algorithm limitations are interesting and can be examined with more detail by analyzing the following table. The following stages are required for Exchange Data.

Sender Thread	**Receiver Thread**
1. Send Request to Remote Process	1. Receiver Request from Remote Process
2. Wait for Responses	2. Process Request / Generate Responses
3. Receive Responses	3. Send responses to Remote Process

Although two threads are not required for this process it is clear that it is advantageous to support two threads to reduce CPU idling and take use the full-duplex connection available in the Java Sockets (basis for connection). However because we have multiple threads reading for a socket we must be careful that the correct thread gets the correct message. For example if the Receiver Thread on the remote process was faster than the send thread on the remote process we might receive the responses before receiving the requests. To we will implement a semaphore to prevent this as you can see in Figure 4 below.

Figure 4. Exchange Data Closer Look



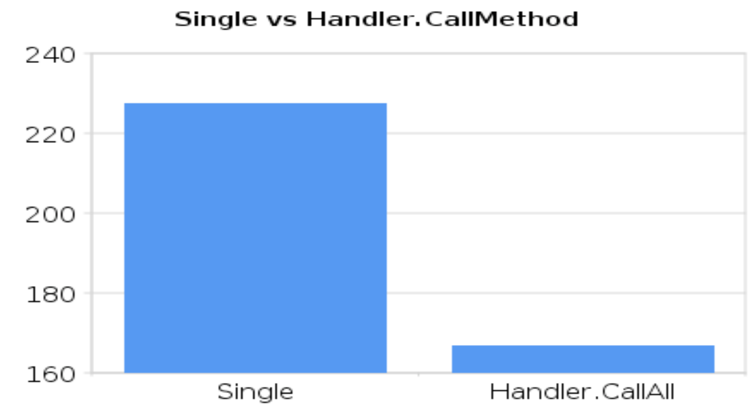
This figure demonstrates the process in each communication pipe shown in Figure 3. There are three phases in the communication. Phase 1 request are made by both processes asynchronously through the full-duplex connection. At this point two Semaphores must be changed (Phase 2), this is because we want to ensure that the request have been received before we begin listening on the socket for responses – this will prevent a message from going to the wrong thread if one of the nodes on one end of the connection is starved (possible case if Node A has a dual-core while Node B has a single core).

Performance

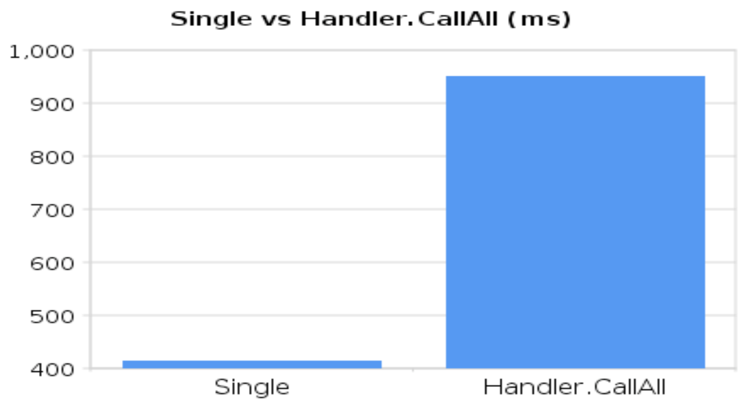
Discussion TBA

CPU Cache Impact

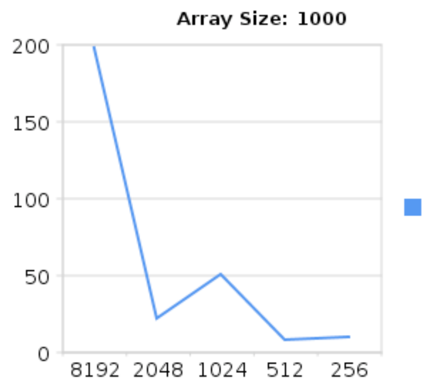
Note: Test preformend on mnode31 with 1mb cache



Note: Test preformend on hercules with 8mb cache



Performance over Cache decrease


Array Size: 1000
200
150
100
50
0
8192
2048
1024
512
256

Call Method

TBA

Exchange All

TBA

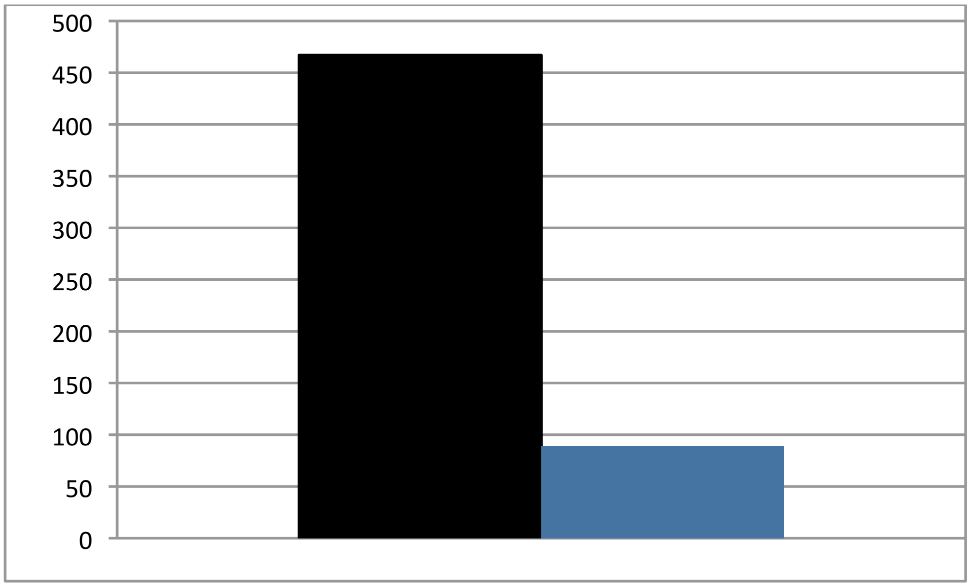
Mass Overall

Discussion TBA

Mass Library		
10,000 Elements (100 x 100)		
Trial	**Total Time**	**Single Time**
1	467	
2	505	
3	452	
4	457	
5	454	
Total:	467	0

Wave2D Single		
10,000 Elements (100 x 100)		
Trial	**Total Time**	**Single Time**
1	79	
2	102	
3	97	
4	71	
5	97	
Total:	89.2	0

10,000 Elements - Mass Library vs Wave2D


500
450
400
350
300
250
200
150
100
50
0