Distributed Mega-Scale Agent Management in a parallel-programming simulation and analysis environment: injection, diffusion, guarded migration, merger and distributed termination

CSS 700 – Winter 2014 – Term Report
Cherie Wasous

Table of Contents
Work Summary ...........................................................................................................................................2
Detailed Specification of Work ..................................................................................................................2
 Current MASS-2013 – “brute force” implementation of enhanced Agents ...........................................2
 Enhanced Agents MASS-2014 ................................................................................................................3
 Controlled injection and diffusion .........................................................................................................3
 Central injection and diffusion ...............................................................................................................4
 Directions where to find and how to run the current climate program ..................................................6
 Snapshot of current Source Code – at user application level ..................................................................10
 ClimateAnalysisMass.java - user main ..................................................................................................10
 ClimateData.java – extends Place .........................................................................................................18
 MaxFinder.java - extends Agent ...........................................................................................................21
 MaxFinderEnh.java - extends AgentEnh (for MASS-2014 enhanced agents) ........................................26
 MaxClimateData.java - data structure ..................................................................................................29
 Snapshot of current Source Code – of MASS-2014 key files & code sniplets ........................................30
 AgentEnh.java extends Agent .................................................................................................................30
 Agents.java – code snipt of enhanced agents constructor .....................................................................31
 MProcess.java – code snipt of new action message for enhanced agents constructor ............................36
 Constants.java – code snipt of new action message for enhanced agents constructor ..........................37
 Message.java – code snipt for enhanced agents constructor ...................................................................37
 Appendix A. Details of MASS-2014: 2-D Controlled injection and diffusion ........................................39
 Appendix B. Details of MASS-2014: 2-D Central injection and diffusion ...............................................44
Work Summary

This quarter I focused my efforts in 2 main areas:

1. Continue implementation of Climate application attempting to use MASS-2013 to do features “brute force” that will be streamlined in the agent management enhancements of my thesis work. Further understanding of agents capabilities and limitations in current MASS-2013.
2. Begin implementation of Agent enhancements in MASS-2014. Initial focus on controlled injection and diffusion, and agent collection/merge/termination.

Detailed Specification of Work

Current MASS-2013 – “brute force” implementation of enhanced Agents

Continued work began in Autumn 2013 (refer to previous term report), adding more “runModes” to traverse agents across the climate application, hunting and gathering a maximum data value. This platform will be used as a comparison for the final MASS-2014 Enhanced Agents solution. Both user level application programming “effort” and performance will be compared between MASS-2013 and MASS-2014.

Refer to slides 4 through 11 of my power point presentation of 2/6/2014 (Wasous_Winter2014_0206.pptx), which includes a summary of the user application classes and a performance measurement snapshot.
Enhanced Agents MASS-2014

I have implemented and began testing/debugging the new Enhanced Agents constructor, which includes a new action message to the other computing nodes. There are some known problems in the MASS-2013 which are currently preventing a thorough exercising of this area. These will be worked on during Spring Break week, in a week long pair-programming session with Prof. Fukuda.

Controlled injection and diffusion

A controlled-injection and diffusion technique has been studied and is currently under implementation. Below is a sketch and summary of the detailed process for both the two and three dimensional techniques. The agents will stay local on each computing node, avoiding inter-node communications for agent migration between nodes.

Two-dimensional Controlled injection and diffusion

![Diagram](image)

Instantiate agents all along mid-point X axis and mid-point Y axis, then:
- Agents on +X axis will migrate North (0, +1)
- Agents on –X axis will migrate South (0, -1)
- Agents on +Y axis will migrate West (-1, 0)
- Agents on –Y axis will migrate East (+1, 0)

More detailed description is provided in appendix A: “Details of MASS-2014 2-D Controlled injection and diffusion”.

Three-dimensional Controlled injection and diffusion

Instantiate agents onto two planes:
- One plane is along mid-point X axis and extends into all Z locations above and below
- The other plane is along mid-point Y axis and extends into all Z locations above and below

The agents then migrate as follows:
- The Z location of the agent never changes.
- Agents on +X axis plane will migrate “North” (0, +1, 0)
- Agents on –X axis plane will migrate “South” (0, -1, 0)
- Agents on +Y axis plane will migrate “West” (-1, 0, 0)
- Agents on –Y axis plane will migrate “East” (+1, 0, 0)
Central injection and diffusion
A central-injection and diffusion technique has been studied and will be implemented. Below is a sketch and summary of the detailed process for both the two and three dimensional techniques.

Two-dimensional Central injection and diffusion

Instantiate one agent in middle of each node, then spawn one agent in each of the directions of N,S,E,W. Then:

Traveling North (in +Y direction)

- Migrate (0, +1)
- Spawn (-1, 0) - this will then become a Traveling West

Traveling South (in -Y direction)

- Migrate (0, -1)
- Spawn (+1, 0) – this will then become a Traveling East

Traveling East (in +X direction)

- Migrate (+1, 0)
- Spawn (0, +1) – this will then become a Traveling North

Traveling West (in -X direction)

- Migrate (-1, 0)
- Spawn (0, -1) – this will then become a Traveling South

More detailed description is provided in appendix B: “Details of MASS-2014 2-D Central injection and diffusion”.
Three-dimensional Central injection and diffusion

Instantiate one agent in middle of each node, then spawn one agent in each of the directions of N,S,E,W,up,down.

Then:

Traveling in +X direction
- Migrate ( +1, 0, 0 )
- Spawn ( 0, +1, 0 ) – this will then become a Traveling in +Y direction
- Spawn ( 0, 0, +1 ) – this will then become a Traveling in +Z direction

Traveling in -X direction
- Migrate ( -1, 0, 0 )
- Spawn ( 0, -1, 0 ) – this will then become a Traveling in -Y direction
- Spawn ( 0, 0, -1 ) – this will then become a Traveling in -Z direction

Traveling in +Y direction
- Migrate ( 0, +1, 0 )
- Spawn ( -1, 0, 0 ) – this will then become a Traveling in -X direction
- Spawn ( 0, 0, +1 ) – this will then become a Traveling in +Z direction

Traveling in -Y direction
- Migrate ( 0, -1, 0 )
- Spawn ( +1, 0, 0 ) – this will then become a Traveling in +X direction
- Spawn ( 0, 0, -1 ) – this will then become a Traveling in -Z direction

Traveling in +Z direction
- Migrate ( 0, 0, +1 )
- Spawn ( -1, 0, 0 ) – this will then become a Traveling in -X direction
- Spawn ( 0, +1, 0 ) – this will then become a Traveling in +Y direction

Traveling in -Z direction
- Migrate ( 0, 0, -1 )
- Spawn ( +1, 0, 0 ) – this will then become a Traveling in +X direction
- Spawn ( 0, -1, 0 ) – this will then become a Traveling in -Y direction (** differs
Directions where to find and how to run the current climate program

The MASS source files are found at:

And the Climate application files are found at:
This is how to compile and run the program. Step 1 is to compile the new MASS.jar.

Step 2 is to compile the Climate application.

Then run the Climate application (various run files / modes are shown):
This is the current output (some debug messages):

[dslab@hercules testrun]$ ./runClimate101a.sh
+ port=32781
+ numProc=1
+ numThrd=1
+ numAgents=2
+ numTimeSlots=2
+ numDays=2
+ runMode=101
+ java -Xms1g -Xmx2g -cp ./DLB.jar:./MASS.jar:/jsch-0.1.44.jar:. ClimateAnalysisMass dslab dslab-302 32781 2 1 1 2 2 101
set port to 32781
# of mNodes: 0 SystemSize: 1
Initialized threads - # 1

t--> Time spent thru MASS init: 22 ms
DLB.properties: values are :
History Based Algo: false
Window Based Algo: false
Slope Based Algo: false
DLB step count : 4

------------------------Beginning Initialization sequence for place handle 1------------------------
I'm place element[ 0, 0 ] and I'm instantiating maxSeen Vector
------------------------Complete Initialization for myPid = 0 with place handle 1------------------------
---Initialization variables for myPid = 0 Places(holder) length: 4 chunksize: 4 total length: 4 remainder: 0
t--> Time spent instantiating Places climateData: 22 ms
t--> Time spent call all Places 'compute_': 20 ms
==> just BEFORE instantiate enhanced agents
Inside Agent Init Enhancements function of myPid = 0
Agents injection: Original
I'm a newly created agent with agentId = 0.
I'm a newly created agent with agentId = 1.
Inside MaxFinder map (myRunMode = 1) for current places location [ 0, 0 ]
Agents.constructor on myPid = 0: added agent(0) to place[0, 0]
Inside MaxFinder map (myRunMode = 1) for current places location [ 0, 1 ]
Inside MaxFinder map (myRunMode = 1) for current places location [ 1, 0 ]
Agents.constructor on myPid = 0: added agent(1) to place[1, 0]
At end of Agent Init Enhancements function of myPid = 0, where bag.size() = 2
==> just AFTER instantiate enhanced agents
total number of agents = 2
==> 2 agents left
*** Just set footprint true for place[ 0, 0 ]
*** Just set footprint true for place[ 1, 0 ]
==> 2 agents left
*** Just set footprint true for place[0, 1]
*** Just set footprint true for place[1, 1]

$t$-- Time spent in loop of agents searching (until totalAgents=0): 4 ms

Inside ClimateData.collect at place 0,0.  maxSeen is size: 2

*** Inside collect, iteration done, thisNodeMax is flux= 488.644, dir= 122.161, day=1, time=1

$t$-- Time spent to collect max climateData from Places: 2 ms

Max value found at day=1, and time=1
  flux=488.644, direction=122.161
  x=122, y=161

$t$-- Time spent in user MAIN of searching for max collected: 1 ms
Finishing MASS...
$t$-- Time for MASS.finish: 1 ms

$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$
[dslab@hercules testrun]$
import java.util.Date;
import java.util.Iterator;
import java.util.Vector;

import MASS.Agents;
import MASS.MASS; // Library for Multi-Agent Spatial Simulation
import MASS.Places;

public class ClimateAnalysisMass {

    // runMode
    // <100 : uses original MASS Agents constructor
    // 100s : uses new enhanced MASS Agents constructor, but in "original" mode
    // 200s : uses new enhanced MASS Agents constructor, with enhanced injection

    // = 0: only use places to do the job (so this main finds max)
    // = 1: put one agent at each timeslot for day 0,
    // then migrate these agents thru each day, staying in same timeslot.
    // = 2: put one agent at each day at timeslot 0,
    // then migrate these agents thru each timeslot, staying in same day.
    // = 3: (not yet working fully!) one agent at each node's 0,0
    // = x: (yet to be implemented) put two agents at each thread:
    // one at start (migrate forward) & one at end (migrate backwards).
    // If they meet at same place, then forward "wins".

    // = 101: Enhanced constructor, but behavior same as runMode=1
    // = 102: Enhanced constructor, but behavior same as runMode=2

    // = 201: Enhanced constructor, inject "controlled" (center axis, NSEW
    // divided march)

    private static final int USE_NO_AGENTS = 0;
    private static final int START_AGENTS_AT_EVERY_X_ZERO = 1;
    private static final int START_AGENTS_AT_EVERY_Y_ZERO = 2;

    private static final int USE_ENHANCED_AGENTS_AT_EVERY_X_ZERO = 101;
    private static final int USE_ENHANCED_AGENTS_AT_EVERY_Y_ZERO = 102;

    public static void main(String[] args) throws Exception {
        Date startMassInitTime = new Date();
        Date stopMassInitTime, stopInitPlacesTime, stopPlacesComputeTime,
        startAgentsLoopTime, stopPlacesCollectTime, stopUserSearchTime, startMassFinishTime,
        stopMassFinishTime, stopPlacesMyMaxGatherTime;

        // Verify number of arguments
        if (args.length < 9) {
            System.err
                .println("\nUsage:
" + "java ClimateAnalysisMass login "
                + "pass port nAgents nProc nThrds nTimeSlots nDays runMode");
            System.exit(-1);
        }
    }
}
// Set variables with the user input data
String login = args[0];
String pass = args[1];
String port = args[2];
int nAgents = Integer.parseInt(args[3]);
int nProcesses = Integer.parseInt(args[4]);
int nThreads = Integer.parseInt(args[5]);

// usually 4 timeslots (midnite, 6am, noon, 6pm) --> this is X dimension
int nTimeSlots = Integer.parseInt(args[6]);
// usually 30 or 364 --> this is Y dimension
int nDays = Integer.parseInt(args[7]);

// runMode
// = 0: only use places to do the job (so main finds max)
// = 1: put one agent at each timeslot for day 0,
// then migrate these agents thru each day, staying in same timeslot.
// = 2: put one agent at each day at timeslot 0,
// then migrate these agents thru each timeslot, staying in same day.
// = 3: needs described
// = 101: uses Enhanced Agents, now just for printing debug messages
// yet to be implemented:
// = x: put two agents at each thread:
// one at start (migrate forward) & one at end (migrate backwards).
// If they meet at same place, then forward "wins".
int runMode = Integer.parseInt(args[8]);

// prepare MASS arguments
String[] massArgs = new String[4];
massArgs[0] = login; // user login
massArgs[1] = pass; // user password
massArgs[2] = "machinefile.txt"; // machine file
massArgs[3] = port; // port

// Start the MASS library
MASS.init(massArgs, nProcesses, nThreads);

long stopMassInitTime = new Date();
long timeMassInit = (stopMassInitTime.getTime() - startMassInitTime.getTime());
System.out.println("\n\nt-- Time spent thru MASS init: " + timeMassInit + " ms");

// Create the ClimateData Places array: nTimeSlots x nDays
// typically 4 times per day (midnight, 6am, noon, 6pm), but
// programmable
// Typically a month of data (30 days), but programmable
// Each place element contains a grid for the Pacific NW, which consists
// of 123 x 162 locations. So at each location in this grid is the
// climate information that that particular timeSlot and day.
int chunk = nTimeSlots / nProcesses;
Places climateData = new Places(1, "ClimateData", chunk, nTimeSlots, nDays);

long stopInitPlacesTime = new Date();
long timeInitPlaces = (stopInitPlacesTime.getTime() - stopMassInitTime.getTime());
System.out.println("\n\nt-- Time spent instantiating Places climateData: " + timeInitPlaces + " ms");
+ timeInitPlaces + " ms";

    // Each climateData element "reads its data", then computes values
    climateData.callAll(ClimateData.compute_, null);

    stopPlacesComputeTime = new Date();
    long timePlacesCompute = (stopPlacesComputeTime.getTime() -
    stopInitPlacesTime
    .getTime());
    System.out.println("t--> Time spent call all Places 'compute_': " + timePlacesCompute + " ms");

    switch (runMode) {
    case USE_NO_AGENTS: // do all work without using any agents
        // gather the max values from each place element
        Object[] tempArgs = new Object[nDays * nTimeSlots];
        Object[] temp = climateData.callAll(ClimateData.myMax_, tempArgs);
        stopPlacesMyMaxGatherTime = new Date();
        long timePlacesMyMaxGather = 0;
        timePlacesMyMaxGather += (stopPlacesMyMaxGatherTime.getTime() -
        stopPlacesComputeTime
        .getTime());
        System.out.println("\nt--> Time spent to gather myMax climateData from each Place: " +
        timePlacesMyMaxGather + " ms");

        // look thru returned values and find the max
        MaxClimateData overallMax = new MaxClimateData();
        overallMax.mcdFlux = 0.0; // set to very low value
        for (int i = 0; i < temp.length; i++) {
            MaxClimateData nextValue = (MaxClimateData) temp[i];
            if (overallMax.mcdFlux < nextValue.mcdFlux) {
                overallMax = nextValue;
            }
        }

        System.out.println("\nMax value found at day=" + overallMax.mcdDay + ", and time=" + overallMax.mcdTime); 
        System.out.println("  flux=" + overallMax.mcdFlux + ", direction=" + overallMax.mcdDir);
        System.out.println("  x=" + overallMax.mcdX + ", y=" + overallMax.mcdY);

        stopUserSearchTime = new Date();
        long timeUserSearch = stopUserSearchTime.getTime() - stopPlacesMyMaxGatherTime.getTime();
        System.out.println("\nt--> Time spent in user MAIN of searching for max collected: " +
        timeUserSearch + " ms");
        break;
    case START_AGENTS_AT_EVERY_X_ZERO:
    case START_AGENTS_AT_EVERY_Y_ZERO:
    case 3: // use agents (start
// dayZero &
// march
// thru
// days) to
// do the
// job

// Create the MaxFinder agents, which will be distributed
// over the climateData elements.
// Currently (source.new3/4) will put one agent at each place until
// run out of agents
// So, for now, ask for # agents = # places

Object[] maxFinderArgs = new Object[8];
maxFinderArgs[0] = runMode;
maxFinderArgs[1] = nTimeSlots;
maxFinderArgs[2] = nDays;
maxFinderArgs[3] = nProcesses;
maxFinderArgs[4] = nThreads;
maxFinderArgs[5] = 0; // justSpawned flag
maxFinderArgs[6] = 0; // only meaningful when justSpawned flag = 1
                      // (spawnNewX)
maxFinderArgs[7] = 0; // only meaningful when justSpawned flag = 1
                      // (spawnNewY)

Agents agents = new Agents(2, "MaxFinder", maxFinderArgs,
                          climateData, nAgents);

startAgentsLoopTime = new Date();

while (agents.totalAgents() > 0) {
    // System.out.println(" ==> " + agents.totalAgents() +
    // " agents left");
    agents.callAll(MaxFinder.find_, (Object) null);
    // System.out.println(" ==> All agents have just executed 'find'
    
    routine ");
    agents.manageAll();
}

Date stopAgentsLoopTime = new Date();

long timeAgentsLoop = (stopAgentsLoopTime.getTime() -
                          startAgentsLoopTime
                          .getTime());
System.out
      .println("t--> Time spent in loop of agents searching
      (until totalAgents=0): ",
              + timeAgentsLoop + " ms");

// gather the max values found at each proc/node
// Double, Double, Integer, Integer, Integer, Integer, Integer
Vector<MaxClimateData> results = new Vector<MaxClimateData>();

// get the max data seen at each process/node and put into results
// how collect the values returned????
// ?? I know only using one thread and one process, for Now!!
// ?? how to determine them ?
// Can't use callSome as there is only code for void callSome (not
// Object[] callSome)

tempArgs = new Object[nDays * nTimeSlots];
// Object[] tempArgs = new Object[nDays * nTimeSlots];
// Object[] tempArgs = new Object[120];

temp = climateData.callAll(ClimateData.collect_, tempArgs);
// Object[] temp = climateData.callAll(ClimateData.collect_,
// tempArgs);

for (int i = 0; i < temp.length; i++) {
    if (temp[i] != null) {
        // System.out.println("*** found non-null value in

        // Object[] temp, at i=

        // + i);

        MaxClimateData temp3 = (MaxClimateData) temp[i];
        results.add(temp3);
        /*
        * System.out.println(
        * "\n** After return from climateData.callAll(collect_),
        see non-null values of :

        *
        */
        System.out.println(" Max value found at day=" +
        * temp3.mcdDay + ", time=" + temp3.mcdTime);
        System.out.println(" flux=" + temp3.mcdFlux +
        * ", direction=" + temp3.mcdDir);
        System.out.println(" x=" + temp3.mcdX + ", y=" +
        * temp3.mcdY);
        */
    }
}

stopPlacesCollectTime = new Date();

long timePlacesCollect = 0;
timePlacesCollect += (stopPlacesCollectTime.getTime() -
stopAgentsLoopTime
    .getTime());
System.out
    .println("\nt---> Time spent to collect max climateData
from Places: "+ timePlacesCollect + " ms");

// now go thru max results from each proc/node & find the real max
// and print to console
Iterator<MaxClimateData> iter = results.iterator();
overallMax = new MaxClimateData();
// MaxClimateData overallMax = new MaxClimateData();
overallMax.mcdFlux = 0.0; // set to very low value; TODO: take 1st
    // value from Vector

instead.
MaxClimateData tempCD = new MaxClimateData();

while (iter.hasNext()) {
    tempCD = iter.next();
    if (overallMax.mcdFlux < tempCD.mcdFlux) { // TODO: What if more
        // than 1 have same
        // maximum value
        // ??!!
        overallMax = tempCD;
    }
}
System.out.println("\nMax value found at day=" + overallMax.mcdDay + ", and time=" + overallMax.mcdTime);
System.out.println("    flux=" + overallMax.mcdFlux + ", direction=" + overallMax.mcdDir);
System.out.println("    x=" + overallMax.mcdX + ", y=" + overallMax.mcdY);

stopUserSearchTime = new Date();
timeUserSearch = stopUserSearchTime.getTime() - stopPlacesCollectTime.getTime();
// long timeUserSearch = stopUserSearchTime.getTime();
// - stopPlacesCollectTime.getTime();
System.out.println("\n--- Time spent in user MAIN of searching for max collected: " + timeUserSearch + " ms");
break;

case USE_ENHANCED_AGENTS_AT_EVERY_X_ZERO:
    // Create the MaxFinder agents, using the enhanced mode
    maxFinderArgs = new Object[8];
    // Object[] maxFinderArgs = new Object[8];
    maxFinderArgs[0] = 1; // TODO: Cherie 3-12-2014 override runmode to be 1 for now!
    // TODO: Cherie 3-12-2014 override runmode to be 1 for now!
    maxFinderArgs[1] = nTimeSlots;
    maxFinderArgs[2] = nDays;
    maxFinderArgs[3] = nProcesses;
    maxFinderArgs[4] = nThreads;
    maxFinderArgs[5] = 0; // justSpawned flag
    maxFinderArgs[6] = 0; // only meaningful when justSpawned flag = 1
    // (spawnNewX)
    maxFinderArgs[7] = 0; // only meaningful when justSpawned flag = 1
    // (spawnNewY)

    int injectionMode = 0;
    int guardedMigrationMode = 2;

    System.out.println("==> just BEFORE instantiate enhanced agents");
    agents = new Agents(injectionMode, guardedMigrationMode, 2,
        "MaxFinder", maxFinderArgs, climateData, nAgents);
    // Agents agents = new Agents(injectionMode, guardedMigrationMode,
    // 2,
    // "MaxFinder", maxFinderArgs, climateData, nAgents);
    System.out.println("==> just AFTER instantiate enhanced agents");

    agents.manageAll(); // clean up value of number of agents
    System.out.println("total number of agents = " + agents.totalAgents());

    startAgentsLoopTime = new Date();

    while (agents.totalAgents() > 0) {
        System.out.println(" ==> " + agents.totalAgents() + ", agents left");
        agents.callAll(MaxFinder.find_, (Object) null);
        // System.out.println(" ==> All agents have just executed 'find"
        routine ");
        agents.manageAll();
    }
```java
stopAgentsLoopTime = new Date();
// Date stopAgentsLoopTime = new Date();

timeAgentsLoop = (stopAgentsLoopTime.getTime() - startAgentsLoopTime.getTime());
// long timeAgentsLoop = (stopAgentsLoopTime.getTime() - startAgentsLoopTime.getTime());
System.out.println("t--> Time spent in loop of agents searching (until totalAgents=0): " + timeAgentsLoop + " ms");

// gather the max values found at each proc/node
// Double, Double, Integer, Integer, Integer, Integer
results = new Vector<MaxClimateData>();
// Vector<MaxClimateData> results = new Vector<MaxClimateData>();

// get the max data seen at each process/node and put into results
// how collect the values returned????
// ?? I know only using one thread and one process, for Now!!
// ?? how to determine them ??
// Can't use callSome as there is only code for void callSome (not
// Object[] callSome)

tempArgs = new Object[nDays * nTimeSlots];
// Object[] tempArgs = new Object[120];

temp = climateData.callAll(ClimateData.collect_, tempArgs);
for (int i = 0; i < temp.length; i++) {
    if (temp[i] != null) {
        // System.out.println("*** found non-null value in Object[] temp, at i=" + i);
        MaxClimateData temp3 = (MaxClimateData) temp[i];
        results.add(temp3);
        /* System.out.println("\n** After return from climateData.callAll(collect_),
see non-null values of :"
* ); System.out.println(" Max value found at day=" +
* temp3.mcdDay + ", and time=" + temp3.mcdTime);
* System.out.println(" flux=" + temp3.mcdFlux +
* ", direction=" + temp3.mcdDir);
* System.out.println(" x=" + temp3.mcdX + ", y=" +
* temp3.mcdY);
* /
    }
}

stopPlacesCollectTime = new Date();
timePlacesCollect = 0;
timePlacesCollect += (stopPlacesCollectTime.getTime() - stopAgentsLoopTime.getTime());
System.out.println("\nt--> Time spent to collect max climateData
```
from Places: 

+ timePlacesCollect + " ms");

// now go thru max results from each proc/node & find the real max
// and print to console
iter = results.iterator();
// Iterator<MaxClimateData> iter = results.iterator();
overallMax = new MaxClimateData();
// MaxClimateData overallMax = new MaxClimateData();
overallMax.mcdFlux = 0.0; // set to very low value; TODO: take 1st
// value from Vector

instead.

tempCD = new MaxClimateData();
// MaxClimateData tempCD = new MaxClimateData();

while (iter.hasNext()) {
tempCD = iter.next();
if (overallMax.mcdFlux < tempCD.mcdFlux) { // TODO: What if more
// than 1 have same
// maximum value
// ??!!
overallMax = tempCD;
}
}

System.out.println("\nMax value found at day=" + overallMax.mcdDay
+ ", and time=" + overallMax.mcdTime);
System.out.println(" flux=" + overallMax.mcdFlux
+ ", direction=" + overallMax.mcdDir);
System.out.println(" x=" + overallMax.mcdX + ", y="
+ overallMax.mcdY);

stopUserSearchTime = new Date();

timeUserSearch = stopUserSearchTime.getTime()
   - stopPlacesCollectTime.getTime();
// long timeUserSearch = stopUserSearchTime.getTime()
// - stopPlacesCollectTime.getTime();
System.out
.println("\n--- Time spent in user MAIN of searching for
max collected: "+ timeUserSearch + " ms");

break;
default:
   System.out.println("\n$$$ that runMode=" + runMode
   + " not yet supported, come back later !! ");
   break;
}

startMassFinishTime = new Date();

// Gracefully shut-down MASS
MASS.finish();

stopMassFinishTime = new Date();

long timeMassFinish = (stopMassFinishTime.getTime() - startMassFinishTime
 .getTime());
System.out.println("t---> Time for MASS.finish: "+timeMassFinish+
" ms");
System.out
.println("n$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$
$");

// Terminate the JVM
System.exit(0);
public ClimateData(Object arg) { // constructor
chunk = (int) arg;
footprint = false;
time = index[0];
day = index[1];
moisture_flux = new double[XRANGE][YRANGE];
direction = new double[XRANGE][YRANGE];
maxFlux = 0.0;

// following is hard coded for now (want to do this only at each
// "local 0,0"
// ??
if ((time == (MASS.MASS.getPid() * chunk)) && (day == 0)) {
    MASS.MASS.log("I'm place element[ " + time + ", " + day
    + " ] and I'm instantiating maxSeen Vector");
    maxSeen = new Vector<MaxClimateData>();
}
}

public Object compute(Object arg) {

    // System.out.println("Got inside compute function of ClimateData time="
    // + time + ", day=" + day);
    // just spending time on some dummy calculations & assignments
    for (int x = 0; x < XRANGE; x++) {
        for (int y = 0; y < YRANGE; y++) {
            moisture_flux[x][y] = (time + 1) * (day + 1)
            * (x + (y / 1000.0));
        }
    }
    for (int x = 0; x < XRANGE; x++) {
        for (int y = 0; y < YRANGE; y++) {
            direction[x][y] = x + (y / 1000.0);
        }
    }
    for (int x = 0; x < XRANGE; x++) {
        for (int y = 0; y < YRANGE; y++) {
            if (maxFlux < moisture_flux[x][y]) {
                // TODO: What if more than
                // 1
                // have same maximum
                maxFlux = moisture_flux[x][y];
                maxX = x;
                maxY = y;
            }
        }
    }

    /*
    * eventually, the previous "dummy" code will be replaced with real
    * code, something like this: NetcdfFile input = NetcdfFile.open(
    * getFileName( time, day ) ); iterX = ( input.findVariable( XWIND )
    * ).read().getIndexIterate( ); iterY = ( input.findVariable( YWIND )
    */
* ) .read( ) .getIterator( ); iterQ = ( input.findVariable( "MOISTURE" ) ) .read( ) .getIterator( );
* 
* while ( iterX.hasNext( ) ) { iterX.next( ); iterY.next( ); iterQ.next( );
* xW = ( double ) iterX.getFloatCurrent( ); xY = ( double ) iterY.getFloatCurrent( );
* xQ = ( double ) iterQ.getFloatCurrent( );
* moisture_flux[x][y] = q * Math.sqrt( Math.pow( xW, 2.0 ) + Math.pow( yW, 2.0 ) );
* direction[x][y] = Math.atan2( xW, xY );
* 
* if ( maxFlux < moisture_flux[x][y] ) { maxFlux = moisture_flux[x][y];
* maxX = x; maxY = y; } }
*/

return null;
}

// just for an investigation/debugging
public Object dummyRoutine(Object arg) {
    return null;
}

// public MaxClimateData collect( Object arg ) {
public Object collect(Object arg) {

    // only one place at each node will be called to collect (main will use
    // callSome (eventually!))
    // (but then user app must know the particular ones to call (not good,
    // complicated for user))

    if (index[0] == 0 && index[1] == 0) {

        // all place elements on this node have put their max into maxSeen,
        // so let place 0, 0 just look thru & find the max for this node &
        // return it
        MaxClimateData thisNodeMax = new MaxClimateData();

        // iterate over contents of maxSeen and find the maximum flux entry
        // and return it
        Iterator<MaxClimateData> iter = maxSeen.iterator();
        System.out.
        .println("
Inside ClimateData.collect at place 0,0. maxSeen is size: 
        + maxSeen.size());
        while (iter.hasNext()) {
            MaxClimateData temp = iter.next();
            if (thisNodeMax.mcdFlux < temp.mcdFlux) {
                /* System.out.println("New max found in collect: " +
                    * thisNodeMax.mcdFlux + " < " + temp.mcdFlux + ", dir=" +
                    * temp.mcdDir + ", day=" + temp.mcdDay + ", time=" +
                    * temp.mcdTime );
                */
                thisNodeMax = temp;
            }
        }

        System.out
        .println("*** Inside collect, iteration done, thisNodeMax is flux= 
        + thisNodeMax.mcdFlux"
MaxFinder.java - extends Agent

// MaxFinder.java  -- Cherie Wasous 3.5.2014

import MASS.Agent; // Library for Multi-Agent Spatial Simulation
// for Vector

public class MaxFinder extends Agent {
    private double maxFlux;
    private double maxDir;
    private int maxTime, maxDay;
    private int maxX, maxY;
    private int myRunMode;
    private int nTimeSlots;
    private int nDays;
    private int nProcs;
    private int nThds;
    private int justSpawned; // if =1 then the next 2 variables have real
    // meaning
    private int spawnNewX;
    private int spawnNewY;

    // Constructors
    // ------------

    public MaxFinder () {
        // return (Object) thisNodeMax;
        return thisNodeMax;
    }

    public Object myMax (Object arg) {

        MaxClimateData myMaxCD = new MaxClimateData();
        myMaxCD.mcdFlux = moisture_flux[maxX][maxY];
        myMaxCD.mcdDir = direction[maxX][maxY];
        myMaxCD.mcdDay = day;
        myMaxCD.mcdTime = time;
        myMaxCD.mcdX = maxX;
        myMaxCD.mcdY = maxY;

        return myMaxCD;
    }
}

...
super();

public MaxFinder(Object arg) {
    super();
    /*
     * myRunMode = (int) object;
     * System.out.println("I'm a newly created agent with runMode = " +
     * myRunMode);
     */
    Object[] args = (Object[]) arg;
    myRunMode = (int) args[0];
    nTimeSlots = (int) args[1];
    nDays = (int) args[2];
    nProcs = (int) args[3];
    nThds = (int) args[4];
    justSpawned = (int) args[5];
    spawnNewX = (int) args[6]; // only has meaning when justSpawned = 1
    spawnNewY = (int) args[7]; // only has meaning when justSpawned = 1

    System.out.println("I'm a newly created agent with agentId = "
        + this.agentId + ":");
}

/**
 * This method replaces the default map method provided in Agent.java.
 * *
 */
@Override
public int map(int maxAgents, int[] size, int[] coordinates) {
    int sizeX = size[0], sizeY = size[1];
    int currX = coordinates[0], currY = coordinates[1];

    System.out.println(" Inside MaxFinder map (myRunMode = " + myRunMode
        + ") for current places location [ " + currX + ", " + currY
        + "]");
    // MASS.MASS.log(" Inside MaxFinder map (myRunMode = " + myRunMode
    // + ") for current places location [ " + currX + ", " + currY
    // + "]");
    if ((myRunMode == 1) || (myRunMode == 101)) {
        // place agent at each timeslot for the first day
        if (currY == 0)
            return 1;
        else
            return 0;
    }

    if ((myRunMode == 2) || (myRunMode == 102)) {
        // place agent at the first timeslot for each day
        if (currX == 0)
            return 1;
        else
            return 0;
    }

    if (myRunMode == 3) {
        // (this runMode not yet fully debugged/tested)
        // place one agent at each node "0,0" location
        // TODO - this code ASSUMES Places divides exactly evenly between
        // nodes
// Meaning: nDays/nProcs = whole number with no remainder!
// This is just test code for now.
// TODO - has bug with very small number of Places elements (4 proc,
// 16 ts, 4 days)
if ((sizeY % nProcs) == 0) {
    int chunkY = sizeY / nProcs;
    if ((currX == 0) && (currY % chunkY == 0))
        return 1; // create agent at this node "0,0"
} else {
    System.out
        .println("####### Error in MAP for agent (code is
simple: wants nDays/nProcs with no remainder)");
    return 0;
}

if (myRunMode == 4) {
    // (this runMode not yet fully debugged/tested)
    // place two agents per thread (start=Forward, end=Backward)
    System.out
        .println("####### Error in MAP for agent (code for runMode
of 4 not yet written)");
    return 0;
}

// the following instruction should never execute:
System.out
    .println("####### Error in MAP for agent (probably bad runMode
value!)");
    return 0;
}

// function identifiers
public static final int find_ = 0;

// this is called from callAll( ) and forwards this call
// to the appropriate function, based on funcId.
// -------------------------------
@Override
public Object callMethod(int funcId, Object args) {
    switch (funcId) {
    case find_:
        return find(args);
    }
    return null;
}

// -------------------------------
// public Object find(Object arg) {

/*
 System.out.println("Got inside find function of MaxFinder maxTime=" +
 this.maxTime + ", maxDay=" + this.maxDay + ", agentID=" + agentId +
 ", footprin" + ((ClimateData)place).footprint");
 */

/*
 System.out.println("This agent located at: [ " +
 ((ClimateData)place).time + " ] [ " + ((ClimateData)place).day +
 " ]");
 */
if (justSpawned == 1) { // if this agent was just spawned, then migrate
    justSpawned = 0;
    migrate(spawnNewX, spawnNewY);
    // System.out.println("@@ migrate (" + spawnNewX + ", " +
    // spawnNewY + ")");
    return null;
}

if (((ClimateData) place).footprint) { // an agent has already visited
    // this place.
    // this should
    not happen for
    // runMode of 1
    // or 2 or 3 or 101
    System.out
    .println("ERROR: an agent has already visited this place, oops !!");
    return null;
} else if (this.maxFlux < ((ClimateData) place).maxFlux) // TODO: what
    // if more
    // than one
    // place
    // element
    // has the
    // same max
    // value ??
{
    // agent not yet visited and this location has larger values
    this.maxFlux = ((ClimateData) place).maxFlux;
    this.maxX = ((ClimateData) place).maxX;
    this.maxY = ((ClimateData) place).maxY;
    this.maxDir = ((ClimateData) place).direction[maxX][maxY];
    this.maxTime = ((ClimateData) place).time;
    this.maxDay = ((ClimateData) place).day;
}

    // indicate agent has visited this climateData element
    // ?? how know another agent is not trying to modify value also? - OK,
    // right?
    ((ClimateData) place).footprint = true;

    // get the X,Y of this agent's current place
    int currX = ((ClimateData) place).index[0], currY = ((ClimateData) place).index[1]; // curr
    // index
    System.out.println("*** Just set footprint true for place[" + currX
    + ", " + currY + "]");

    if ((myRunMode == 3) && (currY % (nDays / nProcs) == 0) && (currX == 0)) {
    // if an originally created agent, then spawn more for this node at
// each timeSlot zero
// spawn agent at each X timeslot 0 for rest of Days on this node
// spawn( int numAgents, Object[] arguments )
int nSpawn = 0;
Object[] totalSpawnArgs = new Object[(nDays / nProcs) - 1];
for (int y = currY + 1; y < (currY + (nDays / nProcs)); y++) {
    Object[] eachSpawnArgs = new Object[8];
eachSpawnArgs[0] = myRunMode;
eachSpawnArgs[1] = nTimeSlots;
eachSpawnArgs[2] = nDays;
eachSpawnArgs[3] = nProcs;
eachSpawnArgs[4] = nThds;
eachSpawnArgs[5] = 1; // justSpawned flag
eachSpawnArgs[6] = 0; // only meaningful when justSpawned flag = // 1 (spawnNewX)
eachSpawnArgs[7] = y; // only meaningful when justSpawned flag = // 1 (spawnNewY)
totalSpawnArgs[nSpawn] = eachSpawnArgs;
    nSpawn++;
}
    // System.out.println("@@ spawn( nSpawn, totalSpawnArgs ); where
    nSpawn="
    // + nSpawn +
    // ", totalSpawnArgs.length=" + totalSpawnArgs.length);
    spawn(nSpawn, totalSpawnArgs);
}
    if (((myRunMode == 1) || (myRunMode == 101)) && (currY == nDays - 1))|| (((myRunMode == 2) || (myRunMode == 102) || (myRunMode == 3)) && (currX == nTimeSlots - 1)) } {
// this is last day for this time sample, then dump this agent's
// max values collected into the Vector for this process
// note: Vector maxSeen is thread-safe
MaxClimateData temp = new MaxClimateData(this.maxFlux, this.maxDir,
this.maxTime, this.maxDay, this.maxX, this.maxY);

// can only do the following statement when running as one process,
// on one node
// currently maxSeen is only instantiated at place element 0,0
// TODO: expand code to create maxSeen at each node's "0,0" place
((ClimateData) place).maxSeen.add(temp);
    kill(); // and then set-up to kill this agent on next manageAll
    } else {
// calculate new coordinates to move to for this agent
    int newX = 0, newY = 0;

    if ((myRunMode == 1) || (myRunMode == 101)) { // agent is moving to
        // next day(Y), but

        // staying
            // in same timeslot(X)
    newX = currX;
    newY = currY + 1;
    }
if (myRunMode == 2 || myRunMode == 102 || myRunMode == 3) { // agent
    // is
    // moving
    // to
    // next
    // timeslot(X), but staying
    // in same day(Y)
    newX = currX + 1;
    newY = currY;
}

// if ( myRunMode == 4 ) { // place two agents per thread
// (start=Forward, end=Backward)
    migrate(newX, newY); // update this agent's index so it will
    // continue to
    // examine climate data locations.
    // How know that all place locations will be visited ?
    // Also, it knows if kill has been called but not yet manageAll ??
    return null;
}

MaxFinderEnh.java - extends AgentEnh (for MASS-2014 enhanced agents)

// MaxFinderEnh.java    -- Cherie Wasous 3.5.2014
// for enhanced agent, injection controlled
import java.util.Iterator;
import java.util.Vector;
import MASS.AgentEnh;
// Library for Multi-Agent Spatial Simulation
// for Vector
public class MaxFinderEnh extends AgentEnh {

    public static Vector<MaxClimateData> agentMaxSeen;

    private double maxFlux;
    private double maxDir;
    private int maxTime, maxDay;
    private int maxX, maxY;

    private int myRunMode;
    private int nTimeSlots;
    private int nDays;
    private int nProcs;
    private int nThds;

    // Constructors
    // ************
    public MaxFinderEnh() {
        super();
    }
public MaxFinderEnh(Object arg) {
    super();

    Object[] args = (Object[]) arg;
    myRunMode = (int) args[0];
    nTimeSlots = (int) args[1];
    nDays = (int) args[2];
    nProcs = (int) args[3];
    nThds = (int) args[4];

    System.out.println("I'm a newly created agent with agentId = "
            + this.agentId + ".");

    if (iAmCollector) {
        MASS.MASS
            .log("I'm Collector agent and I'm instantiating agentMaxSeen Vector");
        agentMaxSeen = new Vector<MaxClimateData>();
    }
}

// function identifiers
public static final int find_ = 0;
// this is called from callAll( ) and forwards this call
// to the appropriate function, based on funcId.

@Override
public Object callMethod(int funcId, Object args) {
    switch (funcId) {
        case find_:  
            return find(args);
    }
    return null;
}

//
// -----------------------------

public Object find(Object arg) {
    
    /*
     * System.out.println("Got inside find function of MaxFinderI maxTime="
     * + this.maxTime + ", maxDay=" + this.maxDay + ", agentID=" + agentId + 
     * ", footprint=" + ((ClimateData)place).footprint );
     */
    
    /*
     * System.out.println("This agent located at: [ " +
     * ((ClimateData)place).time + " ] [ " + ((ClimateData)place).day +
     * " ]");
     */
    if (iAmCollector) {
        if (((ClimateData) place).footprint) {
            // If any new entries into Vector then iterate and only keep
            // desired solutions.
            // User could decide to keep multiple entries (case of all
            // data above/below certain threshold, or tie entries).
            // This sample code it is always known there is only one
            // max across each node & across whole simulation
if (agentMaxSeen.size() > 1) {
    // need to iterate thru & only keep max
    MaxClimateData currentMax = new MaxClimateData();
    Iterator<MaxClimateData> iter = agentMaxSeen.iterator();
    currentMax = iter.next();
    while (iter.hasNext()) {
        MaxClimateData temp = iter.next();
        if (currentMax.mcdFlux < temp.mcdFlux) {
            /*
            * System.out.println("New max found in
            * + thisNodeMax.mcdFlux + "$ + thisNodeMax.mcdDir + "$ +
            * + temp.mcdDir + "$ +
            */
            currentMax = temp;
        }
        agentMaxSeen.add(currentMax); // save the current maximum
    }
} else {
    // indicate agent has visited this climateData element
    ((ClimateData) place).footprint = true;
    // get the X,Y of this agent's current place
    int currX = ((ClimateData) place).index[0];
    int currY = ((ClimateData) place).index[1];
    System.out.println("*** Just set footprint true for place[ "$ + currX + ", "$ + currY + "]");
    // add this place's data into Vector
    this.maxFlux = ((ClimateData) place).maxFlux;
    this.maxX = ((ClimateData) place).maxX;
    this.maxY = ((ClimateData) place).maxY;
    this.maxDir = ((ClimateData) place).direction[this.maxX][this.maxY];
    this.maxTime = ((ClimateData) place).time;
    this.maxDay = ((ClimateData) place).day;
    MaxClimateData temp = new MaxClimateData(this.maxFlux,
        this.maxDir, this.maxTime, this.maxDay, this.maxX,
        this.maxY);
    agentMaxSeen.add(temp);
    // TODO: when very small Places, make sure this agentMaxSeen
    // gets all other agent's values and sorts thru them before
    // it is killed/collected by user
}
} else {
    if (((ClimateData) place).footprint) {
        // Cherie: debug code
        // should user code be able to use MASS logs ???
        MASS.MASS
            .log("ERROR: an agent has already visited this
place, oops !!");
    } else {
        // indicate agent has visited this climateData element
        ((ClimateData) place).footprint = true;
    }
} else {
    // indicate agent has visited this climateData element
    ((ClimateData) place).footprint = true;
}

// Cherie: debug code
// get the X,Y of this agent's current place
int currX = ((ClimateData) place).index[0];
int currY = ((ClimateData) place).index[1];
System.out.println("*** Just set footprint true for place[ "+ currX + ", " + currY + "]");

// collect data if this place has better values
// (user will need to handle cases of tie conditions?)
// (or user may collect everything above/below certain value)
if (this.maxFlux < ((ClimateData) place).maxFlux) {
    this.maxFlux = ((ClimateData) place).maxFlux;
    this.maxX = ((ClimateData) place).maxX;
    this.maxY = ((ClimateData) place).maxY;
    this.maxDir = ((ClimateData) place).direction[maxX][maxY];
    this.maxTime = ((ClimateData) place).time;
    this.maxDay = ((ClimateData) place).day;
}

if (atLastLocation) {
    // if this agent is now at edge of node:
    // add this agent's collection into the Vector for node
    MaxClimateData temp = new MaxClimateData(this.maxFlux,
        this.maxDir, this.maxTime, this.maxDay,
        this.maxX, this.maxY);
    agentMaxSeen.add(temp);
    // and prepare to kill this agent
    kill();
} else {
    byte injectionType = (byte) 2; // controlled
    migrate(injectionType);
}

return null;
}

MaxClimateData.java - data structure

```
import java.io.Serializable;

// MaxClimateData.java -- Cherie Wasous 3.5.2014

// public class MaxClimateData { // OK if no slave nodes
public class MaxClimateData implements Serializable {

    public double mcdFlux;
    public double mcdDir;
    public int mcdTime, mcdDay;
    public int mcdX, mcdY;

    // Constructors
    // ---------
```
public MaxClimateData() {
    super();
}

public MaxClimateData(double flux, double dir, int time, int day, int x,
    int y) {
    super();
    mcdFlux = flux;
    mcdDir = dir;
    mcdTime = time;
    mcdDay = day;
    mcdX = x;
    mcdY = y;
}

Snapshot of current Source Code – of MASS-2014 key files & code sniplets

AgentEnh.java extends Agent

// AgentEnh.java -- Cherie Wasous 3.5.2014
// for enhanced agent, injection controlled
package MASS;

// Library for Multi-Agent Spatial Simulation
// for Vector

public class AgentEnh extends Agent {

    // Cherie 3-18-2014: new Flags for Enhanced Controlled Injection
    // .values are assigned/updated by MASS
    // .user code only reads these values
    // ?? use protected instead of public ??
    public boolean iAmCollector;
    public boolean atLastLocation;
    public int[] directionToMove;

    // Constructors
    // -------

    public AgentEnh() {
        super();
    }

    public AgentEnh(Object arg) {
        super();
        directionToMove = new int[3];
        Object[] args = (Object[]) arg;
        iAmCollector = (boolean) args[0];
        atLastLocation = (boolean) args[1];
directionToMove[0] = (int) args[2];
directionToMove[1] = (int) args[3];
directionToMove[2] = (int) args[4];

//
// ---------------------------------------------------------

// TODO: the following needs updated for the new migrate technique
/**
 * Initiates an agent migration upon a next call to MASS.Agents.manageAll(
 * ). More specifically, migrate( ) updates the calling agent's index[].
 * @param migrationMethod
 * TODO: for now add directionToMove to current place and then
 * check for boundary, etc. Also, update flag to user:
 * atLastLocation (if now migrating to the edge of this node)
 * @return true if a migration was scheduled in success, false if error
 */
public boolean migrate(byte migrationMethod) {
    boolean retVal = false;
    /*
     * the following is the old code for original agent migrate // int[]
     * size = place.size; if (index.length == size.length) { // make sure
     * destination exists boolean alreadyMigrating = false; if
     * (!alreadyMigrating) { boolean validDestination = true; boolean
     * identicalIndex = true; for (int i = 0; i < index.length &&
     * validDestination; i++) { if (index[i] < 0 || index[i] >= size[i]) {
     * validDestination = false; } if (index[i] != this.index[i]) {
     * identicalIndex = false; } } if (validDestination && !identicalIndex)
     * { this.index = index.clone(); // assign the new index retVal = true; }
     * }
     */
    return retVal; // default, i.e., an error
}

Agents.java – code sniplet of enhanced agents constructor

... /**
 * add by Cherie 2-25-2014: Agents Enhancements Constructor
 * *
 * Is the constructor that instantiates "className" objects as a collection
 * of multi-agents with characteristics of "inject" mode and
 * "guardedMigration" mode.
 * *
 * @throws Exception
 * @param inject
a user-given non-negative number to uniquely identify the injection technique to use = 1 means "controlled"

@param guardedMigration
* a user-given non-negative number to uniquely identify the guarded migration technique to use = 1 means "greedy" (only one agent per place element at a time)

@param handle
* a user-given non-negative number to uniquely identify this collection of distributed multi-agents over the system.

@param className
* the name of the class from which each agent is instantiated.

@param argument
* an argument passed to each agent.

@param places
* a distributed array where new agents are instantiated.

@param initPopulation
* the total number of agents to be created.

public Agents(int inject, int guardedMigration, int handle, String className, Object argument, Places places, int initPopulation) throws Exception {
    if (handle < 0) {
        throw new Exception("The handle must be an integer of zero or more.");
    } else if (className == null || className.trim().length() == 0) {
        throw new Exception("The class name must be a valid class name");
    } else if (places == null) {
        throw new Exception("The MASS.Places object cannot be null.");
    } else if (initPopulation < 0) {
        throw new Exception("The starting population must be zero or more.");
    } else if (((inject == 0) || (inject == 1) || (inject == 2))) {
        throw new Exception("The injection mode must be either 0, 1, or 2.");
    } else if (((guardedMigration == 0) || (guardedMigration == 1) || (guardedMigration == 2))) {
        throw new Exception("The guarded migration mode must be either 0, 1, or 2.");
    }

    setTotalAgents(initPopulation); // added by Fukuda on 11-22-13

* place handle ?!?! Also, this.handle has not yet been assigned
* (see about 20 lines below (this.handle = handle;)) So changed
* this code: MASS.log(
* "------------------Beginning Agent Init Enhanced sequence for place handle "
* + this.handle + "------------------");
*/

// Cherie changed 3-12-2014 next line to indicate agent handle (not
// place handle)
MASS.log("------------------Beginning Agent Init Enhanced sequence for agent handle "+ handle + "------------------");

Message m = new Message();
m.createAgentInitEnhancedMessage(inject, guardedMigration, handle,
    className, argument, places.getHandle(), initPopulation);
    // m.createAgentInitializationMessage(handle, className, argument,
    // places.getHandle(), initPopulation);

for (MNode node : MASS.mNodes) {
    node.sendMessage(m);
}

System.err.
    .println("Agent Information sent! Awaiting Acknowledgement... ");

/
* investigating: if the following code is moved to bottom of this
* constructor so that master agent constructor runs in parallel
* with slave agent constructors then there are problems !!! Not
* sure what is going one. Is it just a timing thing ? Slave
* machines must do contructors before do "this." assignments
* (Cherie 3-12-2014)
*/
for (MNode node : MASS.mNodes) {
    node.sendMessage();
}
System.err.println("Received all Acknowledgement... ");
}

this.inject = inject;
thisguardedMigration = guardedMigration;
this.handle = handle;
this.places = places;
this.agentArgument = argument;

    // Cherie: for now just print debug messages
System.out
    .println("Inside Agent Init Enhancements function of myPid = "
          + MASS.myPid);

    // if use the following instead of the system.err.println, then get 2
    // copies to screen
// System.err
// .println("Inside Agent Init Enhancements function of myPid = "
// + MASS.myPid);

// if use the following instead of the system.err.println, then get 2
// copies to screen
// MASS.log("Inside Agent Init Enhancements function of myPid = "
// + MASS.myPid);

File curDir = new File(MASS.CUR_DIR);
// URLClassLoader classLoader = URLClassLoader.newInstance(new URL[] {
// curDir.toURI().toURL() });
klas = Class.forName(className, true, Places.loader);
ctor = klas.getConstructor(Object.class);
// TODO possibly check to see if places is in the MASS.MASS environment

switch (inject) {
    case Constants.AGENTS_INJECT_ORIGINAL: // initialization
        MASS.log("Agents injection: Original");

        // ----- agents added to MASS.MASS here
        if (MASS.addAgents(this)) {
            // make all the agents first
            this.bag = new Vector<Agent>();
            Vector<Agent> tempBag = new Vector<Agent>();
            for (int i = 0; i < initPopulation; i++) {
                // bag.add(createAgent(argument, -1)); // -1 for parentId to
                // be
                // same as agentId, root agent
                createAgent(argument, -1);
            }
            tempBag.addAll(bag);
            bag.clear();
        }
        tempBag.addAll(bag);
        bag.clear();

        // get ready to distribute agent objs among place objs
        Places.Iterator placesIter = places.iterator();
        java.util.Iterator<Agent> agentsIter = tempBag.iterator();
        int[] placesSize = places.size();
        Place currentPlace = null;
        Agent currentAgent = null;
        int placeIndex = -1; // incremented at start so will be 0
        int colonistsNum = -1;
        boolean needNewAgent = true;

        // do the distributing, iterate through each MASS.Place
        // every step thru iteration get a new place, but not always a
        // new agent
        while (placesIter.hasNext() && (agentsIter.hasNext() || !needNewAgent)) {
            placeIndex++;
            currentPlace = placesIter.next();
            // will stick with the same agent if previous place had 0
// colonists
// need to always have an agent to work with first to do
// agent.map()
if (needNewAgent) {
    currentAgent = agentsIter.next();
    needNewAgent = false;
}

colonistsNum = currentAgent.map(initPopulation, placesSize,
    currentPlace.index);
// fill place with each agent colonist
while (colonistsNum > 0
    && (!needNewAgent || agentsIter.hasNext())) {
    if (needNewAgent) {
        // happens whenever > 1 colonists
        currentAgent = agentsIter.next();
        needNewAgent = false;
    }
currentAgent.index = currentPlace.index.clone();
currentAgent.place = currentPlace;
    // agent added to place
    currentPlace.agents.add(currentAgent);

    // Cherie: for now print debug messages
    MASS.log("Agents.constructor on myPid = " + MASS.myPid
        + ": added agent(" + currentAgent.agentId
        + ") to place[" + currentPlace.index[0] + ", 
        + currentPlace.index[1] + "]");

    // the 1 place in MASS.Agents where
    // an agent is added to the bag
    bag.add(currentAgent);
    needNewAgent = true;
    colonistsNum--;
}
}

// Cherie: for now just print debug messages
System.err
    .println("At end of Agent Init Enhancements function of myPid = "
        + MASS.myPid
        + ", where bag.size() = "
        + bag.size());
} else {
    throw new Exception("That handle is already in use.");
}
break;
case Constants.AGENTS_INJECT_CENTEROUTWARDS:
    // MASS.log("Agents injection: Center Point Outwards");
    break;
case Constants.AGENTS_INJECT_CONTROLLED:
    // MASS.log("Agents injection: Controlled");

    break;

case Constants.AGENTS_INJECT_ONEWAYSCAN:
    // MASS.log("Agents injection: One Way Scan");

    break;

case Constants.AGENTS_INJECT_CHAOTIC:
    // MASS.log("Agents injection: Chaotic");

    break;

case Constants.AGENTS_INJECT_RANDOM:
    // MASS.log("Agents injection: Random");

    break;

default:
    MASS.log("Agents injection: Received unknown agents injection command ");

    break;
}

/ *
* problems when moved here by Cherie 3-12-2014 master rank waits for
* all other nodes to send agent init acknowledgement (completed?) if
* (MASS.myPid == 0 & & MASS.systemSize > 1) {
* 
* for (MNode node : MASS.mNodes) { node.receiveMessage(); } 
* System.err.println("Received all Acknowledgement... "); 
* / 
*
}

MProcess.java – code sniplet of new action message for enhanced agents constructor

... in start() ...

case Constants.AGENTS_INIT_ENHANCED:
    log("============= Agent Initialization Params: Action: "
    + m.getAction()
    + " Inject: "
    + m.getInject()
    + " GuardedMigration: "
    + m.getGuardedMigration()
    + " Handle: ")
+ m.getHandle() + " ClassName: " + m.getClassName() + "===================");

AGENTS = new Agents(m.getInject(),
                      m.getGuardedMigration(), m.getHandle(),
                      m.getClassName(), m.getArgument(),
                      MASS.getPlaces(m.getPlacesHandle()),
                      m.getAgentInitPopulation());

log("Initialization Complete.. sending ack");
sendAck();
break;
...

**Constants.java – code snippet of new action message for enhanced agents constructor**

```java
// agent enhancements Cherie added 2-25-2014
public static final int AGENTS_INIT_ENHANCED = 17;

// agent enhancements Cherie added 3-11-2014
public static final int AGENTS_INJECT_ORIGINAL = 0;
public static final int AGENTS_INJECT_CENTEROUTWARDS = 1;
public static final int AGENTS_INJECT_CONTROLLED = 2;
public static final int AGENTS_INJECT_ONEWAYSCAN = 3;
public static final int AGENTS_INJECT_CHAOTIC = 4;
public static final int AGENTS_INJECT_RANDOM = 5;
```

**Message.java – code snippet for enhanced agents constructor**

```java
// Agent enhancements variables
// added - by Cherie 2/25/2014
private int AGENT_INJECT;
private int AGENT_GUARDED_MIGRATION;
...

// Create Agent Initialization Enhancements Message
// added - by Cherie 2/25/2014
public void createAgentInitEnhancedMessage(int injectionMode, int guardedMigrationMode, int handle, String className, Object argument, int placesHandle, int initPopulation) {
    ACTION = Constants.AGENTS_INIT_ENHANCED;
    AGENT_INJECT = injectionMode;
    AGENT_GUARDED_MIGRATION = guardedMigrationMode;
    this.HANDLE = handle;
    ...
```
CLASS_NAME = className;
ARGUMENT = argument;
PLACES_HANDLE = placesHandle;
AGENT_INIT_POPULATION = initPopulation;
}

...

// added by Cherie: 2-25-2014
// Agent Enhancements new parameters
public int getInject() {
    return AGENT_INJECT;
}

public int getGuardedMigration() {
    return AGENT_GUARDED_MIGRATION;
}

...
Appendix A. Details of MASS-2014: 2-D Controlled injection and diffusion

- user calls Agents instantiation with inject = 1
- mass-internals creates all initial agents: o, n, s, e, w
- only the “o” agent (original) will instantiate Vector for collecting results from other agents

at t0:
All o, n, s, e, w agents are created at proper place on each node.

2,15 is original agent and will not migrate nor kill self. This original agent will be the one that collects the data from other agents on this node, as well as the data from its own 2,15 location.

---

X: timeslot
Y: day

<table>
<thead>
<tr>
<th></th>
<th>0</th>
<th>1</th>
<th>2</th>
<th>3</th>
<th>4</th>
<th>5</th>
<th>6</th>
<th>7</th>
<th>8</th>
<th>9</th>
<th>10</th>
<th>11</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>1</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>2</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>3</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>4</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>5</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>6</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>7</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>8</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>9</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>10</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>11</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>12</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>13</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>14</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>15</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>16</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>17</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>18</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>19</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>20</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>21</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>22</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>23</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>24</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>25</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>26</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>27</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>28</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>29</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Node 0  Node 1
- user calls Agent callAll with function that collects data at current location
- and issues migrate (byte of 1) (meaning controlled)
- then manageAll

This is repeated as show in following drawings, until the number of agents at each node = 1

at t1: All n, s, e, w agents migrate and gather data.
- when agent hits the edge of node space, then they dump their data into the “o” agent, and call kill()
- the “o” agent sorts thru the new data, keeping only the appropriate data

- This is repeated as show in following drawings, until the number of agents at each node = 1

Short line with an X indicates an agent hits edge of node and dumps its data into original agent at 2,15 and kills-self.
at t14:
Continue
### Node 0

<table>
<thead>
<tr>
<th>Y (day)</th>
<th>0</th>
<th>1</th>
<th>2</th>
<th>3</th>
<th>4</th>
<th>5</th>
<th>6</th>
<th>7</th>
<th>8</th>
<th>9</th>
<th>10</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>○</td>
<td>○</td>
<td>○</td>
<td>○</td>
<td>○</td>
<td>○</td>
<td>○</td>
<td>○</td>
<td>○</td>
<td>○</td>
<td>○</td>
</tr>
<tr>
<td>1</td>
<td>○</td>
<td>○</td>
<td>○</td>
<td>○</td>
<td>○</td>
<td>○</td>
<td>○</td>
<td>○</td>
<td>○</td>
<td>○</td>
<td>○</td>
</tr>
<tr>
<td>2</td>
<td>○</td>
<td>○</td>
<td>○</td>
<td>○</td>
<td>○</td>
<td>○</td>
<td>○</td>
<td>○</td>
<td>○</td>
<td>○</td>
<td>○</td>
</tr>
<tr>
<td>3</td>
<td>○</td>
<td>○</td>
<td>○</td>
<td>○</td>
<td>○</td>
<td>○</td>
<td>○</td>
<td>○</td>
<td>○</td>
<td>○</td>
<td>○</td>
</tr>
<tr>
<td>4</td>
<td>○</td>
<td>○</td>
<td>○</td>
<td>○</td>
<td>○</td>
<td>○</td>
<td>○</td>
<td>○</td>
<td>○</td>
<td>○</td>
<td>○</td>
</tr>
<tr>
<td>5</td>
<td>○</td>
<td>○</td>
<td>○</td>
<td>○</td>
<td>○</td>
<td>○</td>
<td>○</td>
<td>○</td>
<td>○</td>
<td>○</td>
<td>○</td>
</tr>
<tr>
<td>6</td>
<td>○</td>
<td>○</td>
<td>○</td>
<td>○</td>
<td>○</td>
<td>○</td>
<td>○</td>
<td>○</td>
<td>○</td>
<td>○</td>
<td>○</td>
</tr>
<tr>
<td>7</td>
<td>○</td>
<td>○</td>
<td>○</td>
<td>○</td>
<td>○</td>
<td>○</td>
<td>○</td>
<td>○</td>
<td>○</td>
<td>○</td>
<td>○</td>
</tr>
<tr>
<td>8</td>
<td>○</td>
<td>○</td>
<td>○</td>
<td>○</td>
<td>○</td>
<td>○</td>
<td>○</td>
<td>○</td>
<td>○</td>
<td>○</td>
<td>○</td>
</tr>
<tr>
<td>9</td>
<td>○</td>
<td>○</td>
<td>○</td>
<td>○</td>
<td>○</td>
<td>○</td>
<td>○</td>
<td>○</td>
<td>○</td>
<td>○</td>
<td>○</td>
</tr>
<tr>
<td>10</td>
<td>○</td>
<td>○</td>
<td>○</td>
<td>○</td>
<td>○</td>
<td>○</td>
<td>○</td>
<td>○</td>
<td>○</td>
<td>○</td>
<td>○</td>
</tr>
</tbody>
</table>

### Node 1

<table>
<thead>
<tr>
<th>Y (day)</th>
<th>0</th>
<th>1</th>
<th>2</th>
<th>3</th>
<th>4</th>
<th>5</th>
<th>6</th>
<th>7</th>
<th>8</th>
<th>9</th>
<th>10</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>○</td>
<td>○</td>
<td>○</td>
<td>○</td>
<td>○</td>
<td>○</td>
<td>○</td>
<td>○</td>
<td>○</td>
<td>○</td>
<td>○</td>
</tr>
<tr>
<td>1</td>
<td>○</td>
<td>○</td>
<td>○</td>
<td>○</td>
<td>○</td>
<td>○</td>
<td>○</td>
<td>○</td>
<td>○</td>
<td>○</td>
<td>○</td>
</tr>
<tr>
<td>2</td>
<td>○</td>
<td>○</td>
<td>○</td>
<td>○</td>
<td>○</td>
<td>○</td>
<td>○</td>
<td>○</td>
<td>○</td>
<td>○</td>
<td>○</td>
</tr>
<tr>
<td>3</td>
<td>○</td>
<td>○</td>
<td>○</td>
<td>○</td>
<td>○</td>
<td>○</td>
<td>○</td>
<td>○</td>
<td>○</td>
<td>○</td>
<td>○</td>
</tr>
<tr>
<td>4</td>
<td>○</td>
<td>○</td>
<td>○</td>
<td>○</td>
<td>○</td>
<td>○</td>
<td>○</td>
<td>○</td>
<td>○</td>
<td>○</td>
<td>○</td>
</tr>
<tr>
<td>5</td>
<td>○</td>
<td>○</td>
<td>○</td>
<td>○</td>
<td>○</td>
<td>○</td>
<td>○</td>
<td>○</td>
<td>○</td>
<td>○</td>
<td>○</td>
</tr>
<tr>
<td>6</td>
<td>○</td>
<td>○</td>
<td>○</td>
<td>○</td>
<td>○</td>
<td>○</td>
<td>○</td>
<td>○</td>
<td>○</td>
<td>○</td>
<td>○</td>
</tr>
<tr>
<td>7</td>
<td>○</td>
<td>○</td>
<td>○</td>
<td>○</td>
<td>○</td>
<td>○</td>
<td>○</td>
<td>○</td>
<td>○</td>
<td>○</td>
<td>○</td>
</tr>
<tr>
<td>8</td>
<td>○</td>
<td>○</td>
<td>○</td>
<td>○</td>
<td>○</td>
<td>○</td>
<td>○</td>
<td>○</td>
<td>○</td>
<td>○</td>
<td>○</td>
</tr>
<tr>
<td>9</td>
<td>○</td>
<td>○</td>
<td>○</td>
<td>○</td>
<td>○</td>
<td>○</td>
<td>○</td>
<td>○</td>
<td>○</td>
<td>○</td>
<td>○</td>
</tr>
<tr>
<td>10</td>
<td>○</td>
<td>○</td>
<td>○</td>
<td>○</td>
<td>○</td>
<td>○</td>
<td>○</td>
<td>○</td>
<td>○</td>
<td>○</td>
<td>○</td>
</tr>
</tbody>
</table>

#### at t15:

Last of agents have hit the edge and no just the one Original agent remains on each node. So user application knows to gather data from this agent and then kill it. 

---

Appendix B. Details of MASS-2014: 2-D Central injection and diffusion

At t0:

At t1:
At t2:

At t3:
At t4: