CSS499 Agent Teamwork
Undergrad Research Assistant

Solomon Lane

June 2, 2007

Table of Contents

Introduction

I have worked as a research assistant on the agent teamwork project for several quarters now. My previous work primarily focused on performance evaluation and as a secondary result I would find, report and help troubleshoot various bugs and verify their fixes. The most recent version of agent teamwork agent aims to address many of the performance and usability issues that were present during my previous evaluation. This quarter my goal was to continue this type of verification and performance evaluation of the latest version of agent Teamwork. This final report outlines my accomplishments toward this goal.

Week 1-5 Using the New AgentTeamwork

The first step was to verify the new version by running some existing test programs. The 2 weeks were spent troubleshooting problems with program injected into the agent teamwork environment. Once this was resolved the next 3 weeks were spent debugging problems that were preventing my injected test program from running. During this timeframe several bugs where discovered, debugged, and fixes verified. This section highlights the issues and their resolutions during this stage of the project.

1. Trouble with the new UWPlace. I was able to start it, but when I submited a job, it would throw the following exception after loading several classes.
   [UWClassLoader] className = MPJ/mpjrun Exception in thread "socketThread_1174866044527" java.lang.NoClassDefFoundError: LAgentUtil;
2. Debugging steps to resolve included:
   a. Checking classpaths and included jars, attempting to resubmit and checking the logs
   b. Notifying professor Fukuda who made changes to UWPlace which I validated had not resolved the issue
3. Met with professor Fukuda at campus and we debugged together, he would modify and compile file and I would run test scripts and examine log output
   a. Tried again and it didn't work but Bug was found. AgentTeamwork.Ateam.UserProgWrapper has been requested instead of AgentTeamwork/Ateam/UserProgWrapper.
   b. Began communicating with Josh Phillips determine how he was able to successfully run a program.
   c. Got some additional instructions to try from Jumpei Miyauchi, a student also working on Agent Teamwork. I eventually tried it against the source code in Jumpei's home dir and the program injection did work. I discovered that his source was different from the source I have been instructed to use.
   d. Copying his source I was able to successfully inject a program, but the program would still not run.
   e. Met with Professor Fukuda again to debug together

i. Professor Fukuda found some bugs in UserProgWrapper.java as we debugged

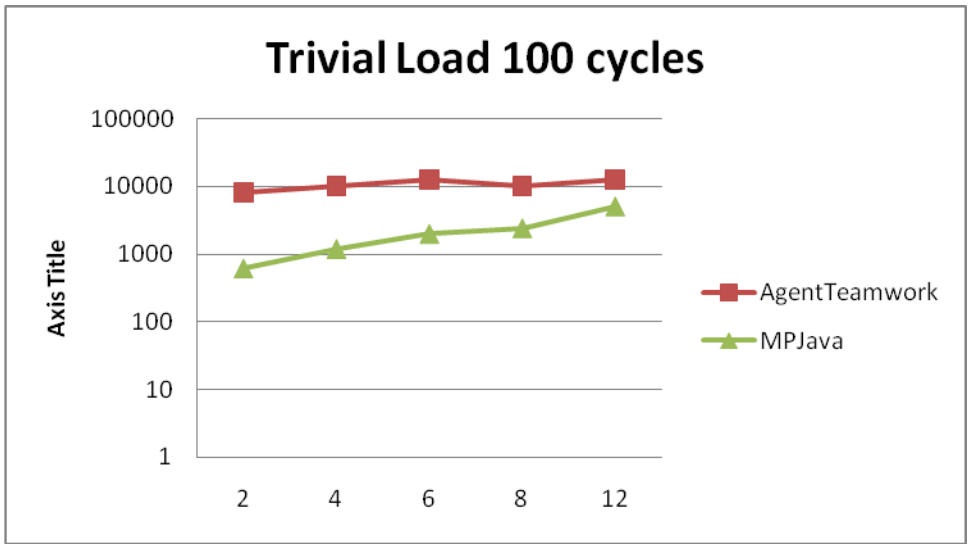
f. Professor Fukuda had me try another version and I found another bug
   i. We iterated through this sequence as neccessary
g. Debugging led Professor Fukuda to also modify SentinelAgent.java and GridTcp.java
h. At this point I could run programs that used the func_n model. But still not those coded with Josh Phillips's model
i. Tried using various example scripts modify classpaths, jars, etc. Recursive diff's of source trees – but at the end of week 5 I could still only run func_n version

Week 6-7 Performance Evaluation & New Programming Model

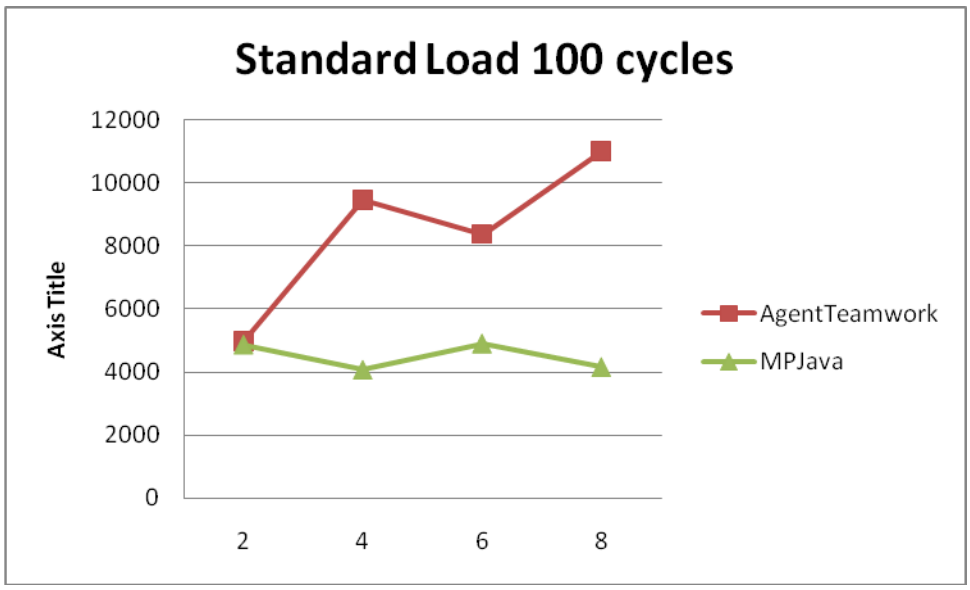
Performance Evaluation

At this stage I could run programs under the func_n model, but not under Josh Phillips's model. Professor Fukuda asked me to stop working on running under Josh's model and move my focus to running performance evaluations. The following outlines the steps I took to accomplish this goal.

1. I wrote a test script to collect the performance information and run the various programs for a number of test cycles.
   a. The text of the script is located in Appendix A
2. Found that Wave2D ran out of memory and spent time running it multiple times and analyzing the logs in an attempt to isolate the tipping point for the out of memory crash. Because of the crashes, I was only able run performance evaluations for a small number of nodes and only for a small number of cycles. At these loads there was no advantage of parallelism, and MPIJava outperformed agent teamwork as indicated in Graph 1 and Graph 2 below.

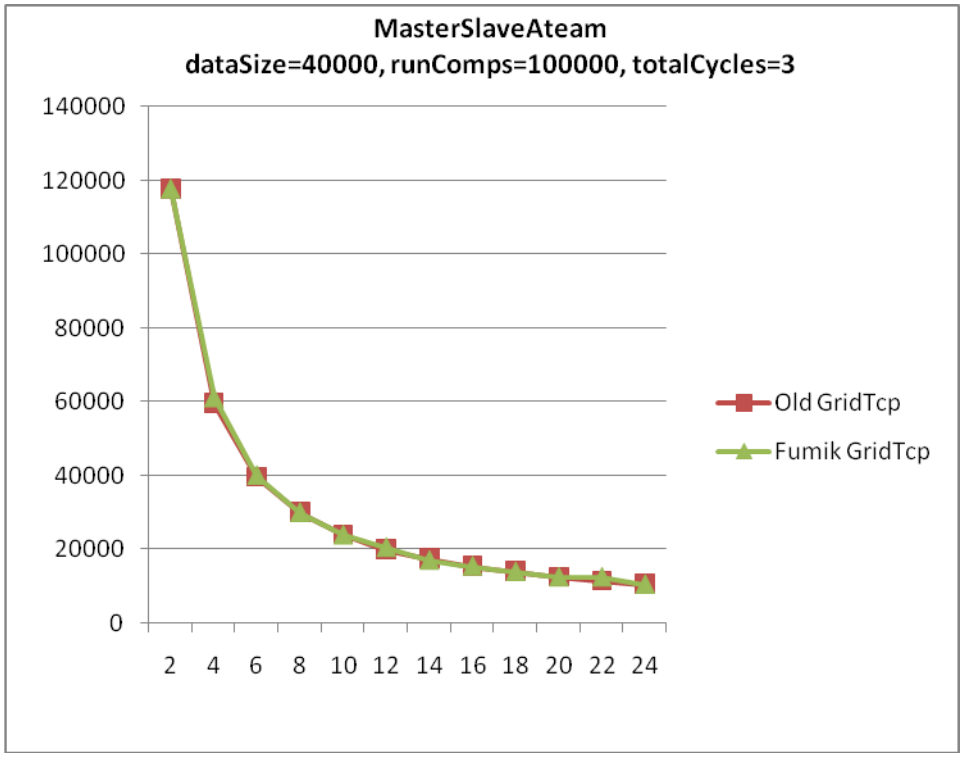


Graph 1: Trivial Load 100 Cycles

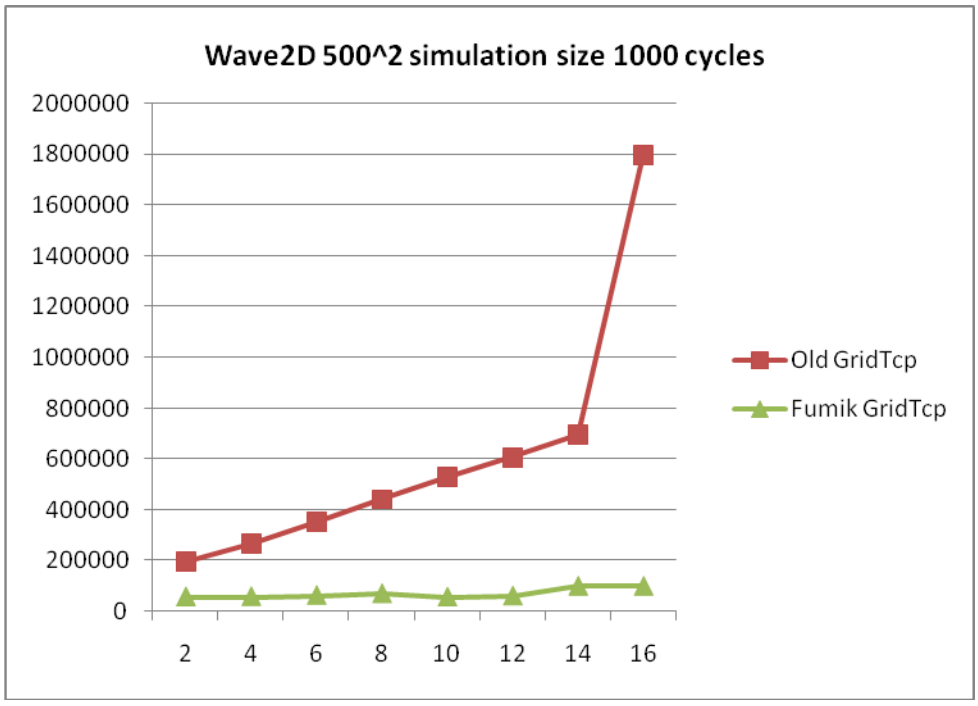


Graph 2: Standard Load 100 cycles

In addition to evaluating Agent teamwork and MPIJava, another of Professor Fukuda's students Fumitka had implemented a higher performance version o f the GridTCP library a fundamental component of Agent teamwork. To evaluate this new version I ran a performance evaluation running the test program with the old version and with the new Fumitka version. As you can see from graphs Four below, there was a dramatic performance improvement in performance with a Wave2D test program in the new Fumitka version. Graph Three shows no difference in performance, however the test program MasterSlave puts very little load in GridTCP hence there is very little difference in performance.



Graph 3: MasterSlaveAteam



Graph 4: Wave2D 500^ simulation size 1000 cycles

Programming in the new model

From the beginning I had hoped to port my existing agent teamwork test programs to a new more usable programming model that had been developed by another of the Professor Fukuda's students, Josh Phillips. Up to this point I had only been able to perform models with the func_n model. While I had been working on performance evaluation Professor Fukuda followed up with Josh Phillips and confirmed that this model requires the following two constructors:

a. public YourProg(Ateam o) { }
   public YourProg( ) { }

This was slightly different from the documentation that I had been referencing which had shown the blank constructor taking an object rather than an Ateam. With this new information, and some example code provided by Professor Fukuda I was now able to port my programs in Josh's programming model.

Now I began porting the test programs to the new model.

Week 8–10 Test Case Automation

My next goal was to implement the following test cases, outlined in the SOA:

repeating the following sequence of test items for each application of InterCluster.java, MD.java, Wave2D.java, and Mandel.java:

1. Deploy an application over a single cluster, kill one sentinel after the first snapshot has been taken, and check if the crashed sentinel can be resumed at another computing node.
2. Deploy an application over two clusters, kill one sentinel (not on a gateway) after the first snapshot has been taken, and check if the crashed sentinel can be resumed at another node within the same cluster.
3. Deploy an application over two clusters, kill one sentinel at a cluster gateway after the first snapshot has been taken, and check if the crashed gateway agent can resume its computation and children within an another cluster.

Create shell scripts, each executing the above test items respectively. Repeat the above test sequence by killing one sentinel before the first snapshot has been taken, so that we can check if a crashed sentinel agent actually restarts an application from its very beginning.

I completed a test script that automated the above test cases and I ported all of the related applications to Josh Phillips model.
To summarize:

1. The test scrip will run all three test specs as outlined in SOA phase3 2.
2. It will run any of the 4 programs outlined in SOA phase3 3.
3. The script source is listed in Appendix B

I ran the test cases and identified the following bugs:

1. MPJ.Init never returns when running any program across two clusters
2. Test spec one fails to recover when running any program on a single cluster with an E_ extra node. After killing a node, instead of migrating to the extra node, rank0 crashes with a connection refused exception.
3. MPJ.COMM_WORLD.Recv crashes with a null pointer Exception at MPJ.Communicator.Recv(Communicator.java:464) when using MPJ.ANY_SOURCE for the source rank.

Conclusion

This quarter I contributed to agent teamwork by find bugs, helping to debug them, and validating their fixes. I also performed further performance evaluations to assist those who are working on performance improvements. Finally I implemented test scripts that significantly reduce the amount of setup and tear down time required to perform tests. This includes starting uwplace on the appropriate nodes, shutting down uwplace when the test is complete, capturing log files, and automatically running complex test scenarios.

Appendix A – Run Multiple Test Cycles

This script runs Wave2D multiple times starting with 2 nodes and then increasing the node count up to MAXNODES. The script collects the runtimes and stores all of the UWPlace logs for the run. The global parameters at the top of the script determine where things will be placed. The script accepts the following arguments:

```
-t type of run, depth or breadth first, passed on to runWave2D.sh
-m max number of nodes
-r the number of run cycle for each test, passed on to runWave2D.sh
-l the test load, which corresponds to sets of parameters to be passed onto runWave2D.sh
    Options are trivial, standard, medium, large

This script depends on runWave2D.sh which is listed in Appendix C.
```

Script Source

```
#!/bin/bash

ROOT_DIR=/home/globus/scratch/FUKUDA-499-SPR
RUN_LOGDIR=$ROOT_DIR/uwplace-logs
TEST_LOGDIR=$ROOT_DIR/test-logs
TIMESTAMP=`date +%F-%k.%M.%S`
TEST_REPORT=$TEST_LOGDIR/test_cycle_results_$TIMESTAMP

touch $TEST_REPORT
if [ $? -ne 0 ]; then
    echo "Fatal: failed to touch $TEST_REPORT"
    exit 1
fi

# defaults that can overriden with arguments
TEST_LOAD="trivial"
TEST_CYCLES=100
MAXNODES=8

function kill_all_uwplace {
    echo "PLEASE WAIT while I shutdown uwplace..."
    # killall uwplace
    for H in `cat $ROOT_DIR/all_machines`; do
        ssh -o BatchMode=yes $H 'killall -g runUWPlace.sh'
    done
    exit
}

trap kill_all_uwplace INT

# process args
for ARG in $*; do
case "$ARG" in
    -t)
        shift
        type=$1
        ;;
    -m)
        shift
        MAXNODES=$1
        ;;
    -r)
        shift
        TEST_CYCLES=$1
        ;;
```

```
    -l)
        shift
        TEST_LOAD=$1
        ;;
     *)
        shift
        ;;
esac
done

case "$TEST_LOAD" in
    trivial)
        TEST_ARGS="-N 100 -xmin 20 -ymin 20 -xmax 30 -ymax 30"
        ;;
    standard)
        TEST_ARGS="-N 500 -xmin 200 -ymin 200 -xmax 300 -ymax 300"
        ;;
    medium)
        TEST_ARGS="-N 800 -xmin 400 -ymin 400 -xmax 600 -ymax 600"
        ;;
    large)
        TEST_ARGS="-N 1000 -xmin 400 -ymin 400 -xmax 600 -ymax 600"
        ;;
esac

function run_test {
    echo "Running $TEST_CMD"
    bash -x $TEST_CMD > $RUN_LOGDIR/runcmd.out 2>&1
    echo "Watching $RUN_LOGDIR/mnode0-UWPlace.log.  Looking for 'Run time for' which indicates the program
is finished"
    count=0
    grep -q 'Run time for' $RUN_LOGDIR/mnode0-UWPlace.log
    while [ $? -ne 0 ];do
        echo -n "."
        sleep 10
        count=$((count+1))
        # timeout after 30 min
        if [ $count -gt 180 ]; then
            RUNDESC=$TEST_LOAD"_"$TEST_CYCLES"_"$NODES"_"$TIMESTAMP
            echo -e "$RUNDESC\tTIMEOUT" >> $TEST_REPORT
            tar -cf $TEST_LOGDIR/$RUNDESC.tar $RUN_LOGDIR
            return
        fi
        grep -q 'Run time for' $RUN_LOGDIR/mnode0-UWPlace.log
    done
    RUNTIME=`grep 'Run time for' $RUN_LOGDIR/mnode0-UWPlace.log`
    RUNDESC=$TEST_LOAD"_"$TEST_CYCLES"_"$NODES"_"$TIMESTAMP
    echo -e "$RUNDESC\t$RUNTIME" >> $TEST_REPORT
    tar -cf $TEST_LOGDIR/$RUNDESC.tar $RUN_LOGDIR
}


NODES=2
while [ $NODES -le $MAXNODES ]; do
    TEST_CMD="runWave2D.sh -t d -n $NODES -r $TEST_CYCLES $TEST_ARGS -i 80 -v"
    run_test
    NODES=$(($NODES+2))
done

kill_all_uwplace
```

Appendix B – Run User Program Test Cases

This shell scripts displays the following instructions when run with –h:

```
usage: FUKUDA-499-SPR/applications/runUserProg.sh -t [1|2|3] -n #nodes -p UserProgram Optional_UPargs
 ex: FUKUDA-499-SPR/applications/runUserProg.sh -t 1 -n 10 -p Wave2D -v

 -t   Test spec to run:
      1 - Deploy an application over a single cluster, kill one sentinel after the first snapshot has been
taken, and
          check if the crashed sentinel can be resumed at another computing node.
      2 - Deploy an application over two clusters, kill one sentinel (not on a gateway) after the first
snapshot has
          been taken, and check if the crashed sentinel can be resumed at another node within the same
cluster.
      3 - Deploy an application over two clusters, kill one sentinel at a cluster gateway after the first
snapshot
          been taken, and check if the crashed gateway agent can resume its computation and children
within an another cluster.
 -p   [Wave2D|MD|InterCluster|Mandellel] The classname of the user program to be run
 -n   the number of nodes to be used for the execution

 Optional_UPargs (user program arguments).  These arguments
 depend on the program being run

=====Wave2D Arguments=====
 -r     the # of computation cycles to run for.  0 == forever
 -N     the simulation size: N*N. default=100*100
 -i     the snampshot interval - the number of computation cycles
         between snapshots
 -v     verbose
 -xmin  cube size limit
 -xmax  cube size limit
 -ymin  cube size limit
 -ymax  cube size limit

Where cube size defaults to xmin=ymin=40, xmax=ymax=60
==========================

=====Molecular Dynamics Arguments=====
 -r   the of computation cycles to run for.  0 == forever
 -N   the number of molecules in the simulation.  Default 100.
 -i     the snampshot interval - the number of computation cycles
         between snapshots
 -v     verbose
 -mold   molecule diameter in pixels.  Default 10
 -boxw   box width in units of the molecular diameter.  Default 40.
 -boxh   box height in units of the molecular diameter. Default 40.
==========================

=====InterCluster Arguments=====
 -r   the of computation cycles to run for. default is 1.
 -N   the message size in bytes to be transered
==============================

=====Mandellel Arguments=====
 -xResolution     resolution height of the graphic to generate
 -yResolution     resolution width of the graphic to generate
 -r               maximum number of iterations to check a point for connectivity to the Mandelbrot set
 -colorMode         Color Modes: BW, ALT-BLACK, ALT-BLUE, ALT-RED, ALT-YELLOW, ALT-GREEN,
                              GRAD-BLACK, GRAD-BLUE, GRAD-RED, GRAD-YELLOW, GRAD-GREEN
 -fileName        The full path the file that will be generated by the program
 -v               verbose
==============================
```

Script Source

```
#!/bin/bash

# breadth first
# while i <= hostcount
# build hoststring, mnode2_
# build hoststring, uw1-320-00_uw1-320-01

. /etc/rc.d/init.d/functions

# global config
AGENT_TEAMWORK_ROOT="/home/globus/scratch/FUKUDA-499-SPR/agentteamwork-dev"
PATH_TO_RUNUWPLACE="$AGENT_TEAMWORK_ROOT/applications"
LOG_DIR="/home/globus/scratch/FUKUDA-499-SPR/uwplace-logs"
COMMANDER="medusa"
CL_medusa="CL_medusa_medusa"
ECL_medusa="E"
CL_priam="CL_priam_priam"
ECL_priam="E"
SENTINEL="perseus"
BOOKKEEPER="phoebe"
PORT=20000
JARS="benchmark/Benchmark.jar,jars/Agents.jar,jars/UWAgent.jar,jars/Ateam.jar,jars/MPJ.jar"
USERPROG_PATH="applications"
USER_CLASSES="_Timer" # addtional user classes, appended to C_$user_prog_name
test_spec=1
UWPLACE_ARRAY=()
MEDUSA_CLUSTER_ARRAY=()
PRIAM_CLUSTER_ARRAY=()
TIMEOUT=30

# user program defaults
runCycles=1000
N=100
snapInterval=10
verbose=""

# wave2D defaults
xmin=40
xmax=60
ymin=40
ymax=60

# molecular dynamics defaults
mold=10
boxw=40
boxh=40

# mandellel defaults
xRes=800
yRes=600
cMode="GRAD-BLUE"
fileName="/home/globus/scratch/FUKUDA-499-SPR/uwplace-logs/mandellel.out"


#-----------------------
# FUNCTIONS
#-----------------------
function usage {
    cat<<HERE

usage: $0 -t [1|2|3] -n #nodes -p UserProgram Optional_UPargs
 ex: $0 -t 1 -n 10 -p Wave2D -v

 -t   Test spec to run:
      1 - Deploy an application over a single cluster, kill one sentinel after the first snapshot has been
taken, and
```

```
          check if the crashed sentinel can be resumed at another computing node.
      2 - Deploy an application over two clusters, kill one sentinel (not on a gateway) after the first
snapshot has
          been taken, and check if the crashed sentinel can be resumed at another node within the same
cluster.
      3 - Deploy an application over two clusters, kill one sentinel at a cluster gateway after the first
snapshot
          been taken, and check if the crashed gateway agent can resume its computation and children
within an another cluster.
 -p   [Wave2D|MD|InterCluster|Mandellel] The classname of the user program to be run
 -n   the number of nodes to be used for the execution

 Optional_UPargs (user program arguments).  These arguments
 depend on the program being run

=====Wave2D Arguments=====
 -r     the # of computation cycles to run for.  0 == forever
 -N     the simulation size: N*N. default=100*100
 -i     the snampshot interval - the number of computation cycles
         between snapshots
 -v     verbose
 -xmin  cube size limit
 -xmax  cube size limit
 -ymin  cube size limit
 -ymax  cube size limit

Where cube size defaults to xmin=ymin=40, xmax=ymax=60
==========================

=====Molecular Dynamics Arguments=====
 -r   the of computation cycles to run for.  0 == forever
 -N   the number of molecules in the simulation.  Default 100.
 -i     the snampshot interval - the number of computation cycles
         between snapshots
 -v     verbose
 -mold   molecule diameter in pixels.  Default 10
 -boxw   box width in units of the molecular diameter.  Default 40.
 -boxh   box height in units of the molecular diameter. Default 40.
==========================

=====InterCluster Arguments=====
 -r   the of computation cycles to run for. default is 1.
 -N   the message size in bytes to be transered
==============================

=====Mandellel Arguments=====
 -xResolution     resolution height of the graphic to generate
 -yResolution     resolution width of the graphic to generate
 -r               maximum number of iterations to check a point for connectivity to the Mandelbrot set
 -colorMode         Color Modes: BW, ALT-BLACK, ALT-BLUE, ALT-RED, ALT-YELLOW, ALT-GREEN,
                              GRAD-BLACK, GRAD-BLUE, GRAD-RED, GRAD-YELLOW, GRAD-GREEN
 -fileName        The full path the file that will be generated by the program
 -v               verbose
==============================
HERE
    exit
}

function shutDownAllUWPlace {
    echo -e  "\nShutting down uwplace on all nodes..."
    for node in ${UWPLACE_ARRAY[*]}; do
        ssh -o BatchMode=yes -o StrictHostKeyChecking=no $node 'killall -g runUWPlace.sh'
    done
    exit
}

function killUWPlace {
     ssh -o BatchMode=yes -o StrictHostKeyChecking=no $1 'killall -9 -g runUWPlace.sh'
}
```

```
function runningUWPlace {
    node=$1
    status=`ssh -o BatchMode=yes -o StrictHostKeyChecking=no $node 'source
/etc/rc.d/init.d/functions;status runUWPlace.sh > /dev/null 2>&1;echo $?'`
    if [ $? -ne 0 ]; then
        return 1
    fi
    if [ $status -ne 0 ]; then
        ssh -o BatchMode=yes -o StrictHostKeyChecking=no $node 'cd '$PATH_TO_RUNUWPLACE'; ./runUWPlace.sh
> '$LOG_DIR/'`hostname`-UWPlace.log 2>&1 &'
    else
        ssh -o BatchMode=yes -o StrictHostKeyChecking=no $node 'killall -g runUWPlace.sh'
        sleep 15
        ssh -o BatchMode=yes -o StrictHostKeyChecking=no $node 'cd '$PATH_TO_RUNUWPLACE'; ./runUWPlace.sh
> '$LOG_DIR/'`hostname`-UWPlace.log 2>&1 &'
    fi
    status=`ssh -o BatchMode=yes -o StrictHostKeyChecking=no $node 'source
/etc/rc.d/init.d/functions;status runUWPlace.sh > /dev/null 2>&1;echo $?'`
    if [ $status -eq 0 ]; then
        UWPLACE_ARRAY=(${UWPLACE_ARRAY[*]} $node)
    fi
    return $status
}

function medusaCluster {
    echo "Initializing Medusa Cluster"
    currentNode=$1
    nodeCount=$2
    extraNodeCount=$3
    i=0
    while [ $currentNode -lt $nodeCount ];do
        if [ $i -gt 31 ]; then
            echo "Failed to run UWPlace on enough nodes in the medusa cluster"
            exit 1
        fi

        runningUWPlace mnode$i
        if [ $? -eq 0 ]; then
            CL_medusa=$CL_medusa"_mnode$i"
            currentNode=$(($currentNode+1))
            MEDUSA_CLUSTER_ARRAY=(${MEDUSA_CLUSTER_ARRAY[*]} mnode$i)
        fi
        i=$(($i+1))
    done

    nodeCount=$(($nodeCount+$extraNodeCount))
    echo "Initializing $extraNodeCount backup nodes on Medusa Cluster"
    while [ $currentNode -lt $nodeCount ];do
        if [ $i -gt 31 ]; then
            echo "Failed to run UWPlace on enough backup nodes in the medusa cluster"
            exit 1
        fi

        runningUWPlace mnode$i
        if [ $? -eq 0 ]; then
            ECL_medusa=$ECL_medusa"_mnode$i"
            currentNode=$(($currentNode+1))
            MEDUSA_CLUSTER_ARRAY=(${MEDUSA_CLUSTER_ARRAY[*]} mnode$i)
        fi
        i=$(($i+1))
    done
}

function priamCluster {
    echo "Initializing Priam Cluster"
    currentNode=$1
    nodeCount=$2
    extraNodeCount=$3
```

```
    i=0

    # init the cluster head
    runningUWPlace priam

    while [ $currentNode -lt $nodeCount ];do
        if [ $i -lt 10 ];then
            node="uw1-320-0$i"
        else
            node="uw1-320-$i"
        fi

        runningUWPlace $node
        if [ $? -eq 0 ]; then
            CL_priam=$CL_priam"_$node"
            currentNode=$(($currentNode+1))
            PRIAM_CLUSTER_ARRAY=(${PRIAM_CLUSTER_ARRAY[*]} $node)
        fi
        i=$(($i+1))

        # exit if we can't find enough nodes runningUWPlace
        if [ $i -gt 31 ]; then
            echo "Failed to run UWPlace on enough nodes in the priam cluster"
            exit 1
        fi
    done

    nodeCount=$(($nodeCount+$extraNodeCount))
    echo "Initializing $extraNodeCount backup nodes on Priam Cluster"
    while [ $currentNode -lt $nodeCount ];do
        if [ $i -gt 31 ]; then
            echo "Failed to run UWPlace on enough backup nodes in the Priam cluster"
            exit 1
        fi

        if [ $i -lt 10 ];then
            node="uw1-320-0$i"
        else
            node="uw1-320-$i"
        fi

        runningUWPlace $node
        if [ $? -eq 0 ]; then
            ECL_priam=$ECL_priam"_$node"
            currentNode=$(($currentNode+1))
            PRIAM_CLUSTER_ARRAY=(${PRIAM_CLUSTER_ARRAY[*]} $node)
        fi
        i=$(($i+1))
    done
}

function breadthFirst {
    count=`expr $1 / 2`
    extraNodes=$2
    if [ `expr $1 % 2` -eq 0 ]; then
        medusaCluster 0 $count $extraNodes
        priamCluster 0 $count $extraNodes
    else
        medusaCluster 0 `expr $count + 1` $extraNodes
        priamCluster 0 $count $extraNodes
    fi
}

function ateamInit {
    # initialize the AgentTeamwork team
    echo "Initializing the Commander: $COMMANDER"
    runningUWPlace $COMMANDER
    if [ $? -ne 0 ];then
      echo "Failed to start UWPlace on Commander: $COMMANDER"
```

```
      exit 1;
    fi

    echo "Initializing the Bookkeeper: $BOOKKEEPER"
    runningUWPlace $BOOKKEEPER
    if [ $? -ne 0 ];then
      echo "Failed to start UWPlace on Bookkeeper: $BOOKKEEPER"
      exit 1;
    fi

    echo "Initializing the Sentinel: $SENTINEL"
    runningUWPlace $SENTINEL
    if [ $? -ne 0 ];then
      echo "Failed to start UWPlace on Sentinel: $SENTINEL"
      exit 1;
    fi
}

function printInfo {

cat<<EOM
======================================================================
$user_prog_name has been injected with the following configuration:
COMMANDER=$COMMANDER
SENTINEL#2=$SENTINEL
BOOKKEEPER=$BOOKKEEPER
PORT=$PORT
NODECOUNT=$nodes

EOM

    if [ ${#MEDUSA_CLUSTER_ARRAY[*]} -gt 0 ]; then
        echo "Medusa Cluster"
        index=1
        for NODE in ${MEDUSA_CLUSTER_ARRAY[*]}; do
            echo "$index. $NODE"
            index=$(($index+1))
        done
    fi
    echo ""

    if [ ${#PRIAM_CLUSTER_ARRAY[*]} -gt 0 ]; then
        echo "Priam Cluster"
        index=1
        for NODE in ${PRIAM_CLUSTER_ARRAY[*]}; do
            echo "$index. $NODE"
            index=$(($index+1))
        done
    fi
    echo ""
}

function lookForMarker {
    marker=$1
    log=$2
    timeout=$3
    count=0
    grep -q "$marker" $log
    while [ $? -ne 0 ];do
        echo -n "."
        sleep 10
        count=$((count+1))
        # check timeout
        if [ $count -gt $timeout ]; then
            return 1
        fi
         grep -q "$marker" $log
    done
}
```

```
#--------------------
# traps
#------------------
trap shutDownAllUWPlace INT


#----------------------
# MAIN
#----------------------

if [ $# -lt 4 ];then
    usage
fi

# process args
for ARG in $*; do
case "$ARG" in
    -p)
        shift
        user_prog_name=$1
        ;;
    -t)
        shift
        test_spec=$1
        ;;
    -n)
        shift
        nodes=$1
        ;;
    -r)
        shift
        runCycles=$1
        ;;
    -i)
        shift
        snapInterval=$1
        ;;
    -N)
        shift
        N=$1
        ;;
    -v)
        shift
        verbose="_verbose"
        ;;
    -xmin)
        shift
        xmin=$1
        ;;
    -xmax)
        shift
        xmax=$1
        ;;
    -ymin)
        shift
        ymin=$1
        ;;
    -ymax)
        shift
        ymax=$1
        ;;
    -mold)
        shift
        mold=$1
        ;;
    -boxw)
        shift
        boxw=$1
```

```
        ;;
    -boxh)
        shift
        boxh=$1
        ;;
    -xResolution)
        shift
        $xRes=$1
        ;;
    -yResolution)
        shift
        $yRes=$1
        ;;
     -colorMode)
        shift
        $cMode=$1
        ;;
     -fileName)
        shift
        $fileName=$1
        ;;
     *)
        shift
        ;;
esac
done

case "$user_prog_name" in
     Wave2D)

UPARGS="runCycles_"$runCycles"_simSize_"$N"_xmin_"$xmin"_xmax_"$xmax"_ymin_"$ymin"_ymax_"$ymax"_snapInterv
al_"$snapInterval$verbose
        ;;
     MD)

UPARGS="verbose_modulo_"$snapInterval"_runCycles_"$runCycles"_Nmol_"$N"_pixelDiameter_"$mold"_boxWidth_"$b
oxw"_boxHeight_"$boxh$verbose
        USER_CLASSES=$USER_CLASSES"_Molecule"
        ;;
     InterCluster)
        UPARGS="runCycles_"$runCycles"_msgSize_"$N
        ;;
     Mandellel)

UPARGS="snapInterval_"$snapInterval"_maxIterations_"$runCycles"_xResolution_"$xRes"_yResolution_"$yRes"_co
lorMode_"$cMode"_fileName_"$fileName$verbose
        USER_CLASSES=$USER_CLASSES"_WorkContract"
        ;;
     *)
        echo "\"$user_prog_name\" is not a supported user program"
        exit 1
        ;;
esac

# prepare run commands
inject="java -Xmx512M UWAgent.UWInject -m 4 -j $JARS -p $PORT $COMMANDER
AgentTeamwork.Agents.CommanderAgent B_$BOOKKEEPER S_$SENTINEL -u AgentTeamwork/Agents/"
userprog=" -up $USERPROG_PATH/$user_prog_name/ U_"$user_prog_name"_dummy_"$PORT"_-np_"$nodes"_"$UPARGS"
C_"$user_prog_name"$USER_CLASSES AP_60001"

cd $AGENT_TEAMWORK_ROOT
clear
echo "current directory: `pwd`"
ateamInit

case "$test_spec" in
    1)
        medusaCluster 0 $nodes 1
        echo -e "Injecting Program:\n$inject $CL_medusa $userprog"
```

```
        $inject $CL_medusa $userprog > $LOG_DIR/command.out 2>&1
        printInfo
        echo "Running Test Spec 1"
        TEST_CASE_VICTIM=${MEDUSA_CLUSTER_ARRAY[1]}
        ;;
    2)
        breadthFirst $nodes 1
        echo -e "Injecting Program:\n$inject $CL_priam $CL_medusa $userprog"
        $inject $CL_priam $CL_medusa $userprog > $LOG_DIR/command.out 2>&1
        printInfo
        echo "Running Test Spec 2"
        TEST_CASE_VICTIM=${MEDUSA_CLUSTER_ARRAY[1]}
        ;;
    3)
        breadthFirst $nodes 1
        echo -e "Injecting Program:\n$inject $CL_priam $CL_medusa $userprog"
        $inject $CL_priam $CL_medusa $userprog > $LOG_DIR/command.out 2>&1
        printInfo
        echo "Running Test Spec 3"
        TEST_CASE_VICTIM=$SENTINEL
        ;;
esac

rank0log="$LOG_DIR/"${MEDUSA_CLUSTER_ARRAY[0]}"-UWPlace.log"
TEST_MARKER="Taking a snapshot"
echo "Waiting for marker '$TEST_MARKER' in $rank0log"
lookForMarker "$TEST_MARKER" "$rank0log" "$TIMEOUT"
if [ $? -eq 0 ];then
    echo "found Marker, killing $TEST_CASE_VICTIM"
    # kill the node
#    killUWPlace $TEST_CASE_VICTIM
    # now wait for the program to finish
    TEST_MARKER='Run time for'
    echo "Waiting for final marker '$TEST_MARKER' in $rank0log"
    lookForMarker "$TEST_MARKER" "$rank0log" "$TIMEOUT"
    if [ $? -ne 0 ];then
        echo "Timed out waiting for final marker.  The program probably didn't recover."
    else
        echo "found: " `grep "$TEST_MARKER" $rank0log`
    fi
else
    echo "Timed out waiting for marker"
fi
```

Appendix C – runWave2D.sh

This script displays the following instructions when run with the –h option:

```
usage: FUKUDA-499-SPR/applications/Wave2D/runWave2D.sh -t [b|d|u] -n #nodes Optional_UPargs
 ex: FUKUDA-499-SPR/applications/Wave2D/runWave2D.sh b 10

 -t   type of distribution:
      b(breath first) uses medusa and uw1-320 equally
      d(depth first) uses up medusa first thereafter uw1-320
      u(depth first) uses up uw1-320 first thereafter medusa

 -n   the number of nodes to be used for the execution

 Optional_UPargs (user program arguments)
 -r     the of computation cycles to run for.  0 == forever
 -N     the simulation size: N*N. default=100*100
 -i     the snampshot interval - the number of computation cycles
         between snapshots
 -v     verbose
 -xmin  cube size limit
 -xmax  cube size limit
 -ymin  cube size limit
 -ymax  cube size limit

Where cube size defaults to xmin=ymin=40, xmax=ymax=60
```

Script Source

```
#!/bin/bash

# breadth first
# while i <= hostcount
# build hoststring, mnode2_
# build hoststring, uw1-320-00_uw1-320-01

. /etc/rc.d/init.d/functions

# globals
AGENT_TEAMWORK_ROOT=/home/globus/scratch/FUKUDA-499-SPR/agentteamwork-dev
COMMANDER="medusa"
CL_medusa="S_perseus"
CL_priam="CL_priam"
SENTINEL="perseus"
BOOKKEEPER="phoebe"

# MD default params
runCycles=1000    # the of computation cycles to run for.  0 == forever
N=100             # simulation_size^-2
snapInterval=10
verbose=""
xmin=40
xmax=60
ymin=40
ymax=60

#-----------------------
# FUNCTIONS
#-----------------------
function usage {
    cat<<HERE

usage: $0 -t [b|d|u] -n #nodes Optional_UPargs
```

```
 ex: $0 b 10

 -t   type of distribution:
      b(breath first) uses medusa and uw1-320 equally
      d(depth first) uses up medusa first thereafter uw1-320
      u(depth first) uses up uw1-320 first thereafter medusa

 -n   the number of nodes to be used for the execution

 Optional_UPargs (user program arguments)
 -r     the of computation cycles to run for.  0 == forever
 -N     the simulation size: N*N. default=100*100
 -i     the snampshot interval - the number of computation cycles
         between snapshots
 -v     verbose
 -xmin  cube size limit
 -xmax  cube size limit
 -ymin  cube size limit
 -ymax  cube size limit

Where cube size defaults to xmin=ymin=40, xmax=ymax=60

HERE
    exit
}

function runningUWPlace {
    status=`ssh -o BatchMode=yes -o StrictHostKeyChecking=no $1 'source /etc/rc.d/init.d/functions;status
runUWPlace.sh > /dev/null 2>&1;echo $?'`
    if [ $? -ne 0 ]; then
        return 1
    fi
    if [ $status -ne 0 ]; then
        ssh -o BatchMode=yes -o StrictHostKeyChecking=no $1 'cd /home/globus/scratch/FUKUDA-499-
SPR/agentteamwork-dev/applications ; ./runUWPlace.sh > /home/globus/scratch/FUKUDA-499-SPR/uwplace-
logs/`hostname`-UWPlace.log 2>&1 &'
    else
        ssh -o BatchMode=yes -o StrictHostKeyChecking=no $1 'killall -g runUWPlace.sh'
        sleep 15
        ssh -o BatchMode=yes -o StrictHostKeyChecking=no $1 'cd /home/globus/scratch/FUKUDA-499-
SPR/agentteamwork-dev/applications ; ./runUWPlace.sh > /home/globus/scratch/FUKUDA-499-SPR/uwplace-
logs/`hostname`-UWPlace.log 2>&1 &'
    fi
    status=`ssh -o BatchMode=yes -o StrictHostKeyChecking=no $1 'source /etc/rc.d/init.d/functions;status
runUWPlace.sh > /dev/null 2>&1;echo $?'`
    return $status
}

function medusaCluster {
    echo "Initializing Medusa Cluster"
    currentNode=$1
    nodeCount=$2
    i=0
    while [ $currentNode -lt $nodeCount ];do
        if [ $i -gt 31 ]; then
            echo "Failed to run UWPlace on enough nodes in the medusa cluster"
            exit 1
        fi

        runningUWPlace mnode$i
        if [ $? -eq 0 ]; then
            CL_medusa=$CL_medusa"_mnode$i"
            currentNode=$(($currentNode+1))
        fi
        i=$(($i+1))
    done
}

function priamCluster {
```

```
    echo "Initializing Priam Cluster"
    currentNode=$1
    nodeCount=$2
    i=0

    # init the cluster head
    runningUWPlace priam

    while [ $currentNode -lt $nodeCount ];do
        if [ $i -lt 10 ];then
            node="uw1-320-0$i"
        else
            node="uw1-320-$i"
        fi

        runningUWPlace $node
        if [ $? -eq 0 ]; then
            CL_priam=$CL_priam"_$node"
            currentNode=$(($currentNode+1))
        fi
        i=$(($i+1))

        # exit if we can't find enough nodes runningUWPlace
        if [ $i -gt 31 ]; then
            echo "Failed to run UWPlace on enough nodes in the priam cluster"
            exit 1
        fi
    done
}

function breadthFirst {
    # $1 == node count
    count=`expr $1 / 2`
    if [ `expr $1 % 2` -eq 0 ]; then
        medusaCluster 0 $count
        priamCluster 0 $count
    else
        medusaCluster 0 `expr $count + 1`
        priamCluster 0 $count
    fi
}

#-----------------------
# MAIN
#----------------------

if [ $# -lt 4 ];then
    usage
fi

# process args
for ARG in $*; do
case "$ARG" in
    -t)
        shift
        type=$1
        ;;
    -n)
        shift
        nodes=$1
        ;;
    -r)
        shift
        runCycles=$1
        ;;
    -i)
        shift
        snapInterval=$1
        ;;
```

```
    -N)
        shift
        N=$1
        ;;
    -v)
        verbose="_verbose"
        ;;
    -xmin)
        shift
        xmin=$1
        ;;
    -xmax)
        shift
        xmax=$1
        ;;
    -ymin)
        shift
        ymin=$1
        ;;
    -ymax)
        shift
        ymax=$1
        ;;
     *)
        shift
        ;;
esac
done

echo "type = " $type " #nodes = " $nodes


# initialize the AgentTeamwork team
echo "Initializing the Commander: $COMMANDER"
runningUWPlace $COMMANDER
if [ $? -ne 0 ];then
  echo "Failed to start UWPlace on Commander: $COMMANDER"
  exit 1;
fi

echo "Initializing the Bookkeeper: $BOOKKEEPER"
runningUWPlace $BOOKKEEPER
if [ $? -ne 0 ];then
  echo "Failed to start UWPlace on Bookkeeper: $BOOKKEEPER"
  exit 1;
fi

echo "Initializing the Sentinel: $SENTINEL"
runningUWPlace $SENTINEL
if [ $? -ne 0 ];then
  echo "Failed to start UWPlace on Sentinel: $SENTINEL"
  exit 1;
fi

cd $AGENT_TEAMWORK_ROOT

# prepare run commands
UPARGS="runCycles_"$runCycles"_simSize_"$N"_xmin_"$xmin"_xmax_"$xmax"_ymin_"$ymin"_ymax_"$ymax$"_snapInter
val_"$snapInterval$verbose
#inject="java -cp UWAgent.jar:GridTcp.jar:MPJ.jar:. UWInject -p 20000 $COMMANDER CommanderAgent
S_$SENTINEL"
inject="java -Xmx512M UWAgent.UWInject -p 20000 $COMMANDER AgentTeamwork.Agents.CommanderAgent
B_$BOOKKEEPER"
userprog="-m 4 -j benchmark/Benchmark.jar,jars/Agents.jar,jars/UWAgent.jar,jars/Ateam.jar,jars/MPJ.jar -u
AgentTeamwork/Agents/ -up applications/Wave2D/ U_Wave2D_dummy_2000_-np_"$nodes"_"$UPARGS" C_Wave2D_Timer
AP_60001"
#userprog="-m 4 -j benchmark/Benchmark.jar,jars/Agents.jar,jars/UWAgent.jar,jars/Ateam.jar,jars/MPJ.jar -u
AgentTeamwork/Agents/ -up applications/Wave2D/ U_Wave2D_10000_20000_"$nodes"_"$UPARGS" C_Wave2D_Timer
AP_60001"
```

```
# execute the appropriate cluster configuration
case "$type" in
    b)
        echo "type: breadth first"
        breadthFirst $nodes
        pwd
        echo "Injecting Program"
        $inject $CL_medusa $CL_priam $userprog
        ;;
    d)
        echo "type: depth first (medusa)"
        if [ $nodes -lt 33 ]; then
            medusaCluster 0 $nodes
            echo "Injecting Program"
            $inject $CL_medusa $userprog
        else
            medusaCluster 0 32
            priamCluster 32 $nodes
            echo "Injecting Program"
            $inject $CL_medusa $CL_priam $userprog
        fi
        ;;
    u)
        echo "type: depth first (uw1-320)"
        if [ $nodes -lt 33 ]; then
            priamCluster 0 $nodes
            echo "Injecting Program"
            $inject $CL_priam $userprog
        else
            priamCluster 0 32
            medusaCluster 32 $nodes
            echo "Injecting Program"
            $inject $CL_priam $CL_medusa $userprog
        fi
        ;;
     *)
        usage
        ;;
esac
```