Distributed MASON

Rajdeep Singh Lather and Lilas Dinh
GEORGE MASON UNIVERSITY

Build from Scratch

This is only needed for setting up the environment and building for the first time

This is a quick checklist on how to build Distributed Mason, which includes a listing of jars and how to install OpenMPI. I assume that you have a UNIX-like system; if this is not the case, please translate anything you see in black highlight in mono to the appropriate commands or variables.

1. Have Java 8 sdk installed

Completely case dependent. Often OSX already has it installed. Linux will often require you to install your preferred JDK.

2. Have OpenMPI installed with Java bindings.

You can't just get it from a package manager, usually, unless you specifically get it compiled with it Java bindings (not impossible but typically not possible). So, check first-- but typically, you'll need to build it.

2.1. Ensure that you have the following tools installed - Automake, Autoconf, Libtool
  2.1.1. In OSX you can do this by doing
    `brew install automake autoconf libtool`
2.2. `git clone https://github.com/open-mpi/ompi`
2.3. `cd ompi`
2.4. `./autogen.pl`
2.5. Choose a configuration option
  2.5.1. Do This:
    `./configure --enable-mpi-java`
  2.5.2. Read this to choose your own configuration command:
    2.5.2.1. https://github.com/open-mpi/ompi/blob/master/README.JAVA.txt
    2.5.2.2. https://github.com/open-mpi/ompi/blob/master/README
2.6. `make all install`
2.7. `cp ompi/mpi/java/java/mpi.jar` *`[wherever for $CLASSPATH]`*

3. Clone and install distributed MASON

3.1. Clone

3.1.1. `cd [real installation dir]`
3.1.2. `git clone https://github.com/eclab/mason`

3.2. Install MASON (Skip this if already installed)

3.2.1. `cd mason`
3.2.2. `mvn clean install`

3.3. Install Distributed

3.3.1. `cd distributed`

3.3.2. `mvn clean install`

4. Workflow Steps/Recompiling:

Once you've done this, you won't have to do it again. To rebuild, you simply need do

`mvn clean install`

If you want to only rebuild a particular module, then

`cd [module]; mvn clean install`

4.1. Suggested tips if working from barebones (no integrated development environment, using *vim*, *emacs*, etc.)

Put all the jars in a directory

Make some script that launches in terminal on start (`.bashrc` or `bash_profile` etc.) which looks in that directory and loads them all into the `$CLASSPATH` variable. Below is an example.

```
# /bin/bash
# File: .bashrc
for i in $(find ~/.m2 -type f -name "*.jar");
do
        $CLASSPATH=$CLASSPATH:$i
done
export CLASSPATH
```

If you need to update the `$CLASSPATH` because of an update to the directory, do:

`source` *file*

Example: `source .bashrc`

4.2. Suggested Workflow for Eclipse

4.2.1. Import in Eclipse

4.2.1.1. Import the Maven project.

File -> Import -> Existing Maven Projects -> Root Directory

4.2.1.2. Build and Install the mason-build project.

In MASON Run as -> Maven install

If needed update dependencies for Maven using right click -> maven -> update project

4.2.1.3. Import the Distributed Project

File -> Import -> Existing Maven Projects -> distributed Directory

4.2.1.4. Build and Install Distributed

In distributed Run as -> Maven install

If needed update dependencies for Maven using right click -> maven -> update project

4.2.2. Setup Classpath for Eclipse

Go to Java Build Path -> Add -> mason

4.2.3. Exporting JAR

Export -> Runnable JAR -> Specify class and destination, then select “package required libraries into Generated JAR”

Running Applications

Available Built-In Demos:

Currently, there are two demos you can run:

- sim.app.dflockers.DFlockers
- sim.app.dheatbugs.DHeatBugs

They are located in sim/app/dflockers and sim/app/dheatbugs respectively (from the distributed/ directory)

Running applications locally

`mpirun -np *n* java *application*`

Eg: `mpirun -np 4 java sim.app.dflockers.DFlockers`

One restriction on the value of n - **It must be a multiple of 4.**

The reason is due to the partitioning scheme (the way that the platform divides tasking amongst machines) underneath uses QuadTrees

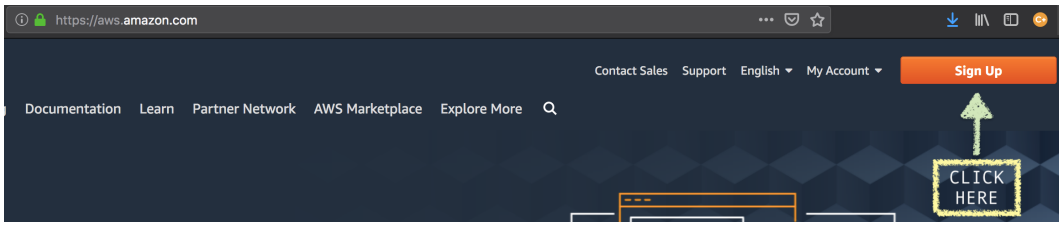
Running applications over AWS

If you wish to take advantage of a cluster, we have instructions for setup over Amazon Web Services. Of course, you can skip any step of which you have already done to go to the next part. The final two steps are the only ones strictly necessary if the appropriate setup is complete.

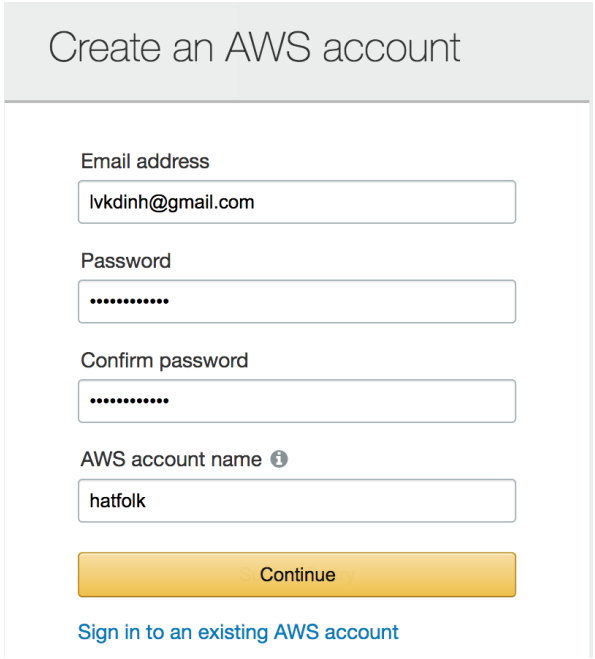
Acquire an AWS account with Root permissions

If not provided already, the way to do this is the following:

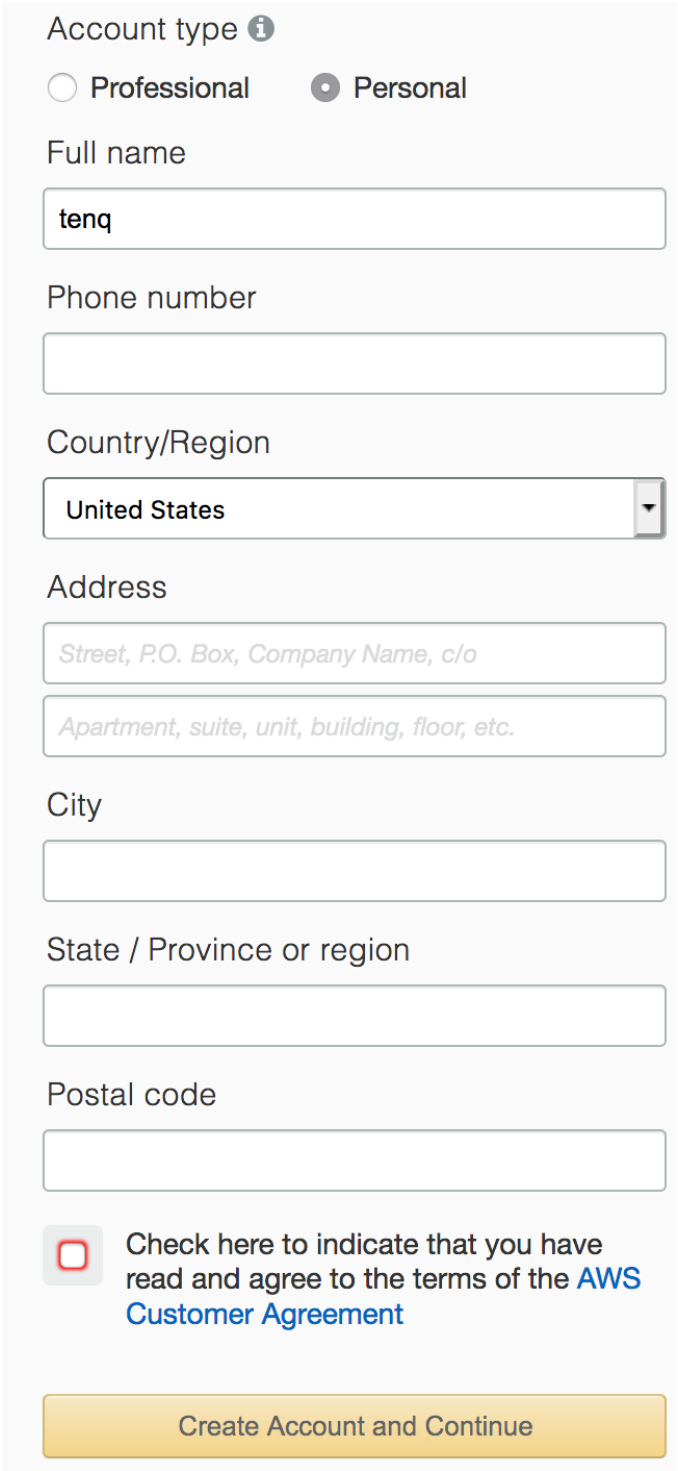
1. Go here and click Sign-Up: https://aws.amazon.com/



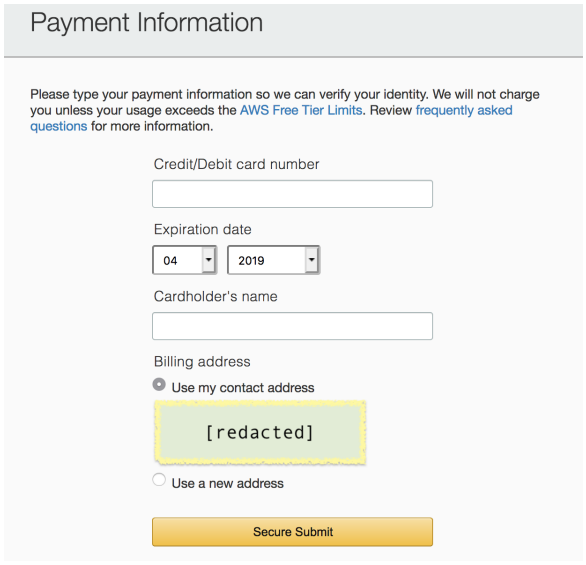
2. Fill in the information you'd like to put here and press Continue.



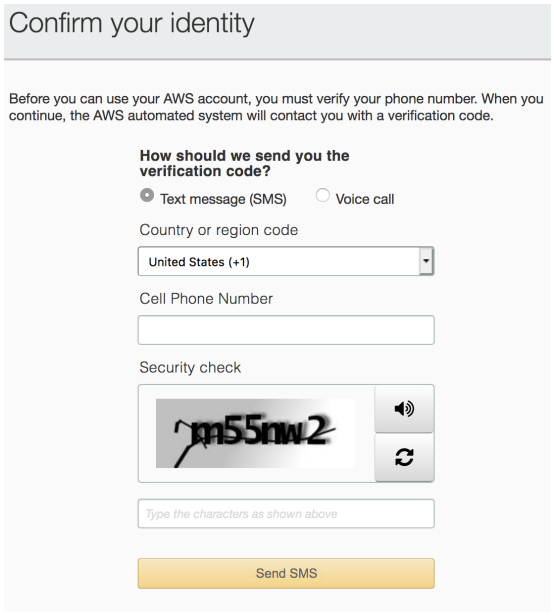
3. Fill in the appropriate information for your account. It should be noted that the address needs to exist. You need to tick the Checkbox for “Check here to indicate that you have read and agree to the terms of the AWS Customer Agreement”. Finally, hit Create Account and Continue. Also, the phone number needs to be callable or textable.



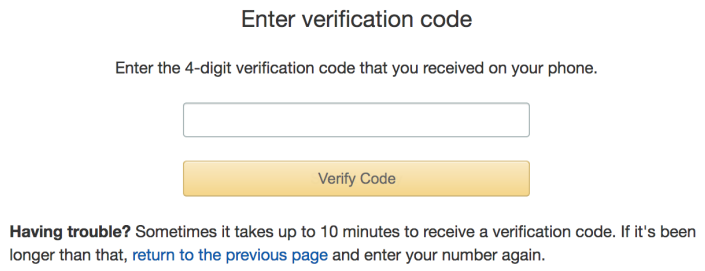
4. You are required to put in some form of payment - Amazon requires this to verify identity.



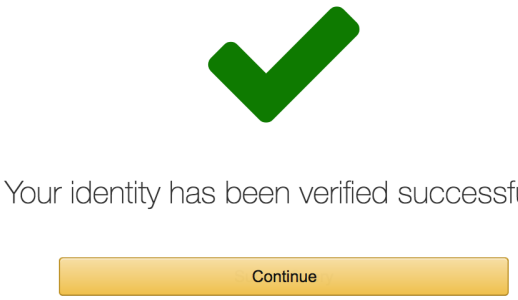
5. You'll have to do a Phone identity challenge. Choose whether you want your code texted or called. Input your phone number. Write your Security Check. The click Send SMS/Contact Me.



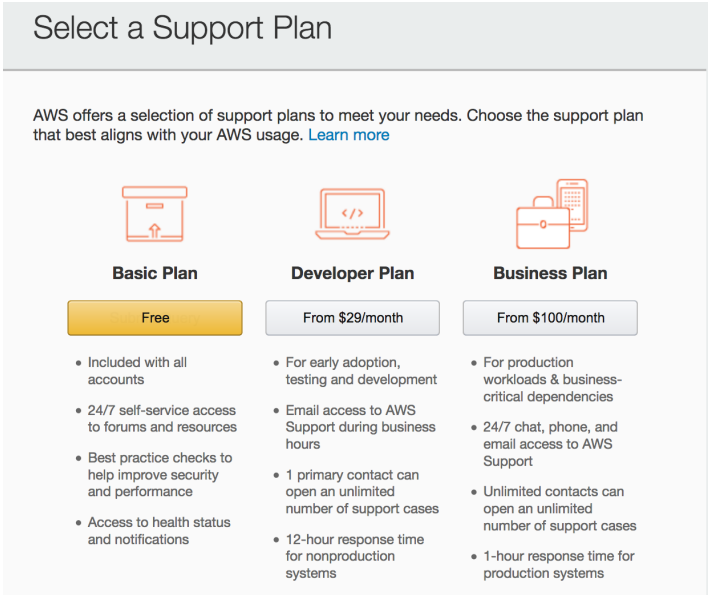
6. Input the code you are given; however you receive it.



Click on Verify Code. And you'll get to click Continue.



7. Choose your tier setting.



8. You'll get emails saying that your account is ready, and you should be able to log in now using the information used above.

Have awscli installed

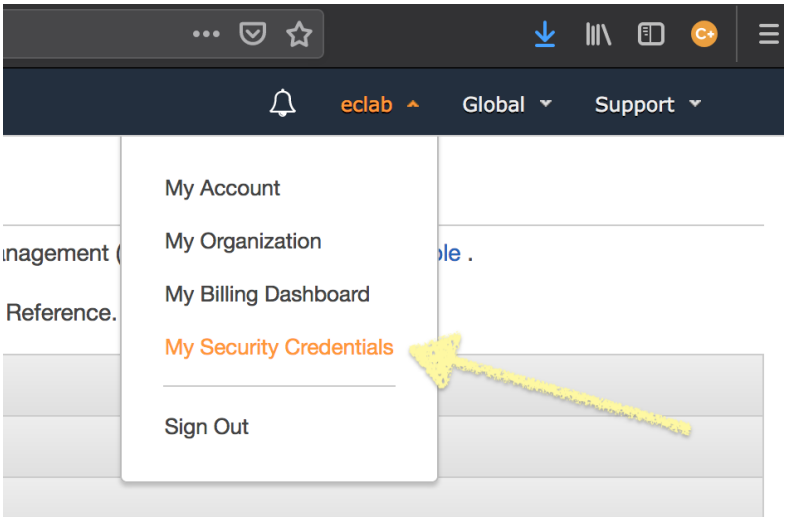
This should be available via package manager, especially via pip. More information here: https://aws.amazon.com/cli/
E.g.: pip install awscli

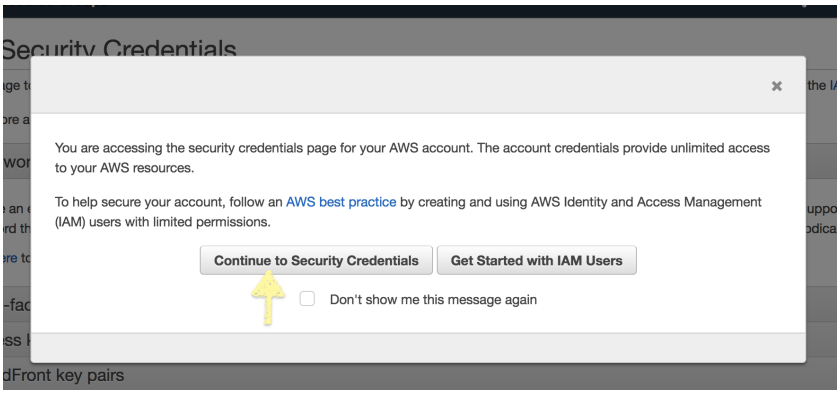
Gather the Following Data (if you don't already have it)

Access Key ID

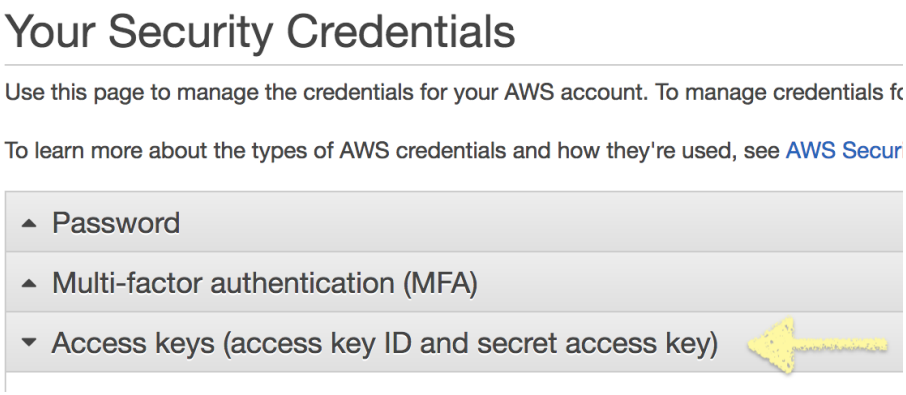
To find this information, in the AWS Management console, hover over your account name.

Click on "My Security Credentials"

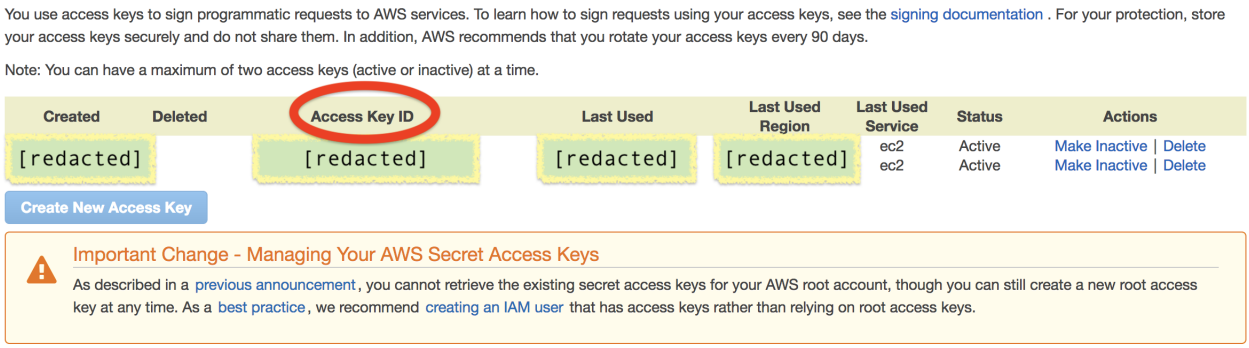




Reveal your Access Keys.

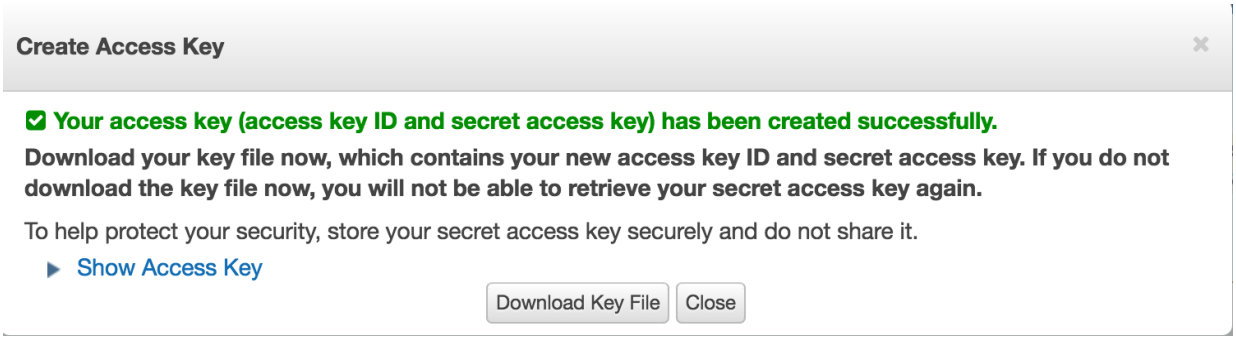


You'll find your Access Key IDs under the circled column below. If you have this, then hopefully you have your Secret Access Key as well. If you don't know your Secret Access Key or you don't have any keys listed, go to the next step.

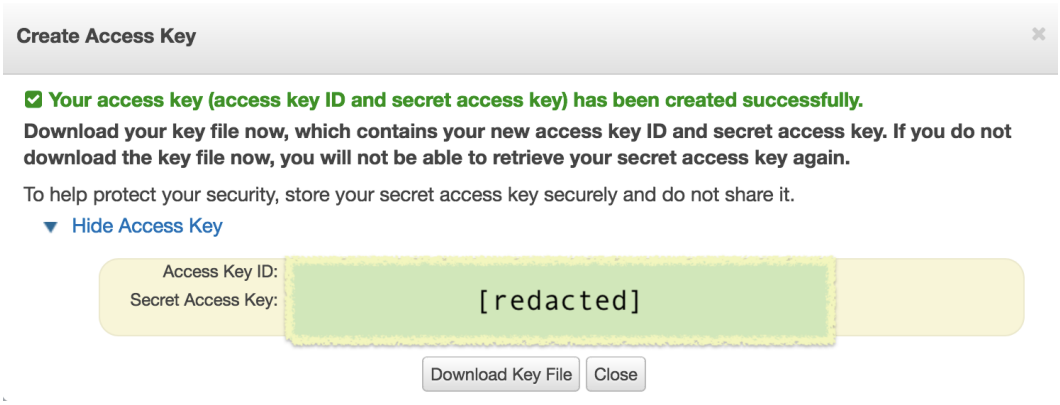


Secret Access Key

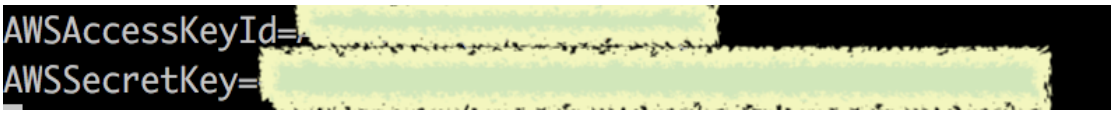
In the same menu as before, click "Create New Access Key".



You should download the Key File and place it somewhere.
Clicking on "Show Access Key" will give you both the Access Key ID and the Secret Access Key.
Make sure to download these in a place you can both find and keep secure. Amazon won't allow you to download it again and this will be the only time this is revealed over your browser.



Downloading the Key File will result in some file called “rootkey.csv” which will look like this:



This, of course, will contain identical information from above.

AWS Region

This has to do with where the machines you’ll want to you will be located. Usually, you’ll want machines that are nearest in geolocation.

For George Mason, the region code that is likely best preferred is `us-east-1`.
For a full list of other regions, here’s the AWS document.

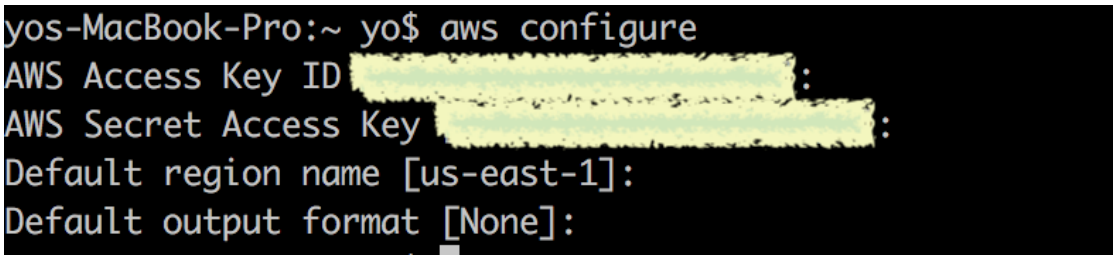
Configure aws

You can pick to do one of two ways.

1. Use awscli to do it

   `aws configure`

   It’ll then ask you to input the Access Key ID, Secret Access Key, and region name. Finally, it’ll ask you for the Output Format, but you can usually leave that blank. To know more about that, here’s the documentation for that. It’ll look more or less like this:

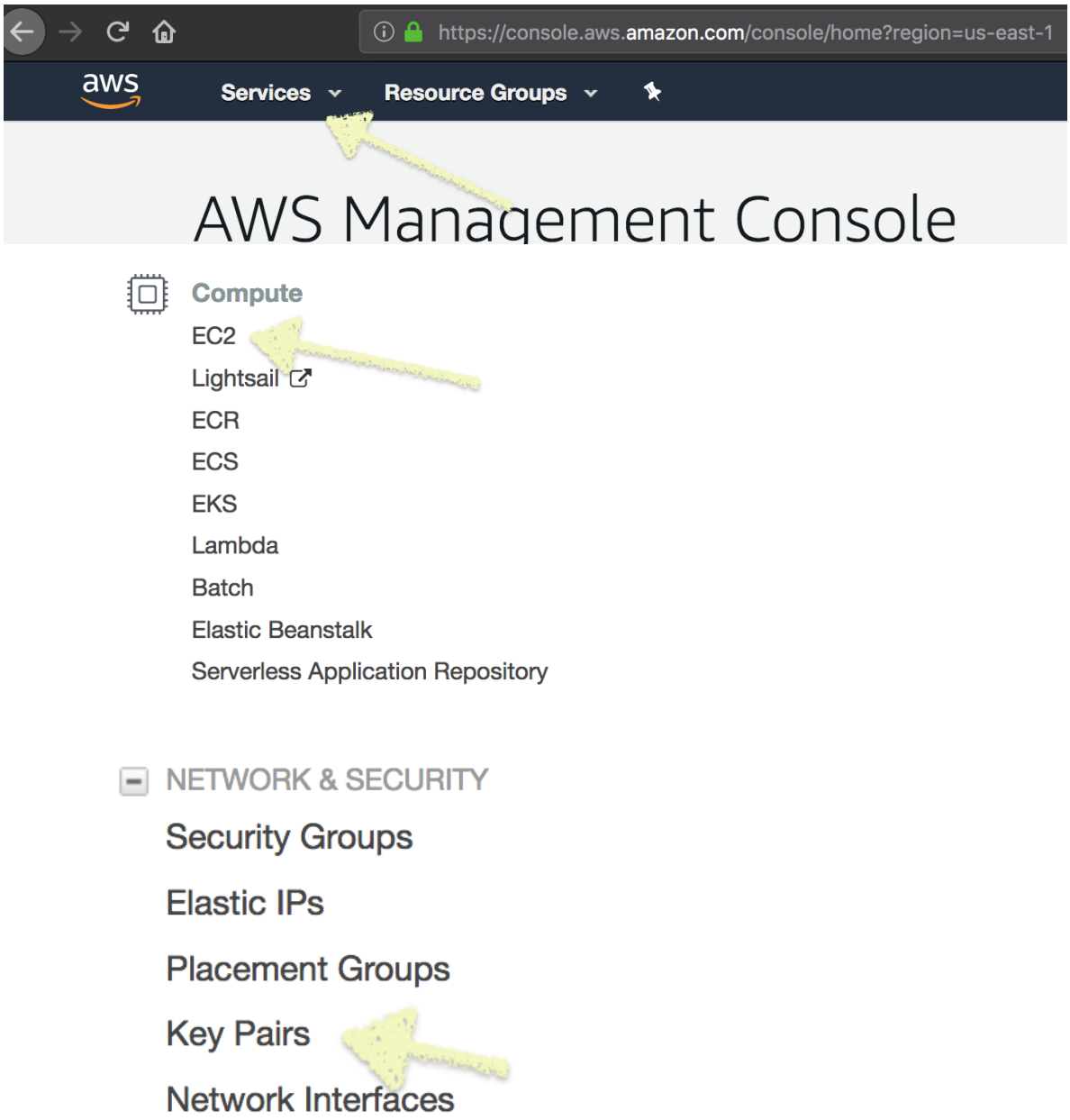
2. Fill in the files manually

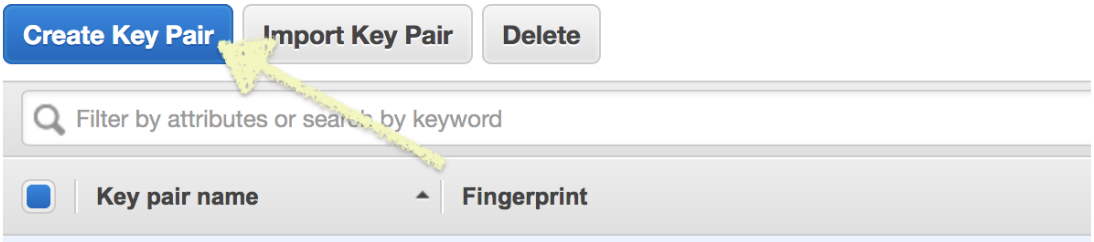
   An alternative to using `aws configure` is just creating the following files with the information filled in

```
#In file ~/.aws/config ; Fill in region as appropriate.
[default]
region = $region_name
```

```
#In file ~/.aws/credentials
[default]
aws_access_key_id = $access_key_id
aws_secret_access_key = $secret_access_key
```

Go get your PEM file.





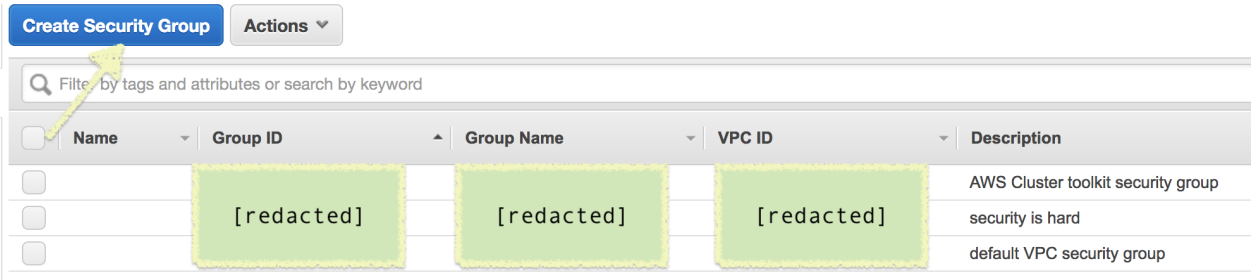
I'd name it mason. And then download the pem file and put it in `~/.ssh/`

You'll need to chmod it. For example, if it's called `$pemfile.pem` and located at the recommended location, `~/.ssh`, then you can chmod it like so.

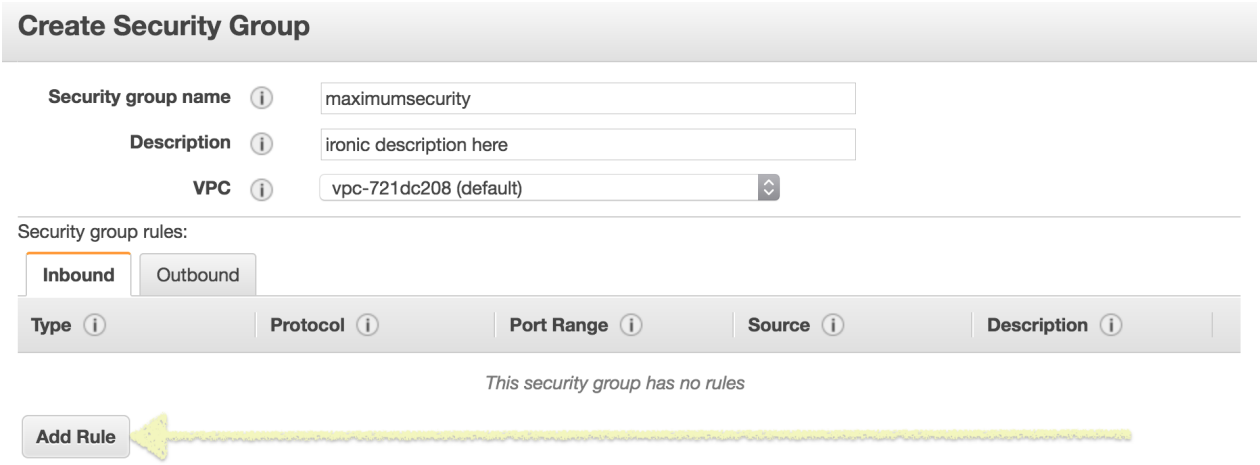
```
chmod g-r ~/.ssh/$pemfile.pem; chmod o-r ~/.ssh/$pemfile.pem
```

Setup Security Groups

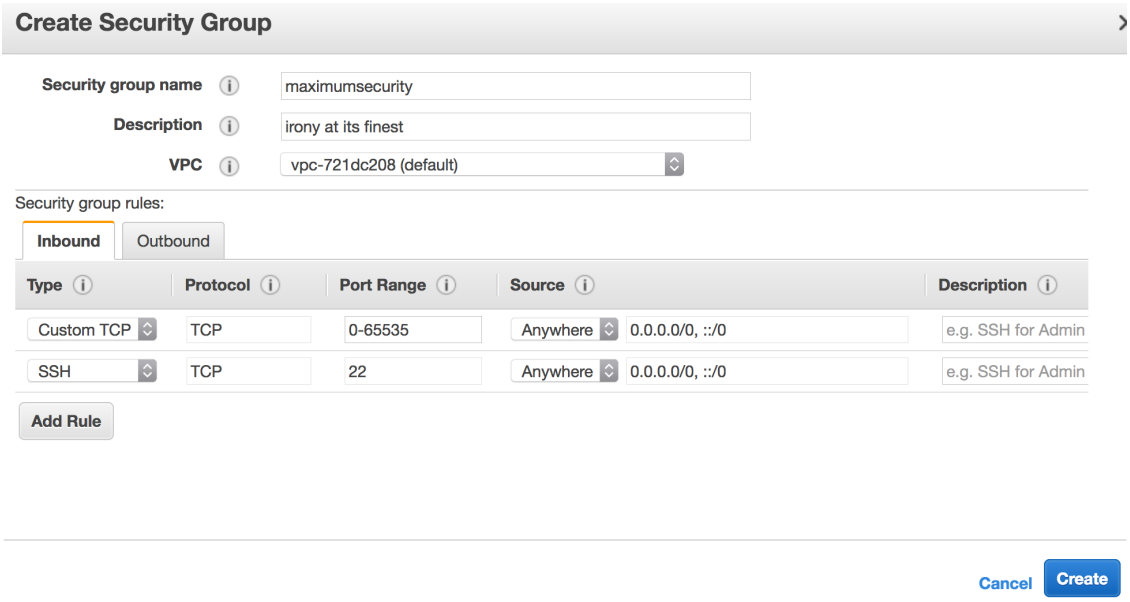
Follow the above until the second to last picture. Instead of selecting "Key Pairs", select "Security Groups"



Select Create Security Group:



Create Two Rules and assign it like so:



Once you hit create, look up the Group ID associated with your Group Name.
Keep the Security Group ID around (referred to as sgid)

Setup the Subnet

ec2 -> Default VPC -> Virtual Private Cloud -> Subnets

Compute

EC2

Lightsail

ECR

ECS

EKS

Lambda

Batch

Elastic Beanstalk

Serverless Application Repository

Account Attributes

Supported Platforms

VPC

Default VPC

vpc-721dc208

Resource ID length management

Console experiments

Virtual Private Cloud

Your VPCs

Subnets

Route Tables

Internet Gateways

Egress Only Internet Gateways

DHCP Options Sets

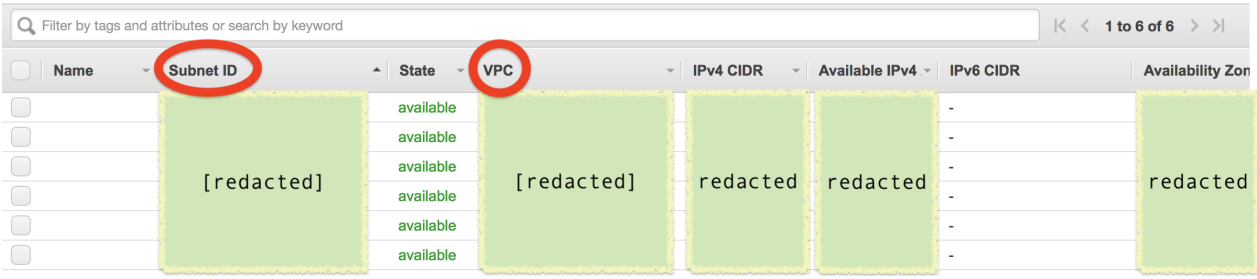
Elastic IPs

Endpoints

Endpoint Services

NAT Gateways

Peering Connections



You'll have a subnet already, so if you don't want a specific subnet for a specific VPC, just keep a subnet id around.
Else, just create the subnet.

Edit aws-config

In `mason/contrib/distributed/scripts/aws-config` change the appropriate variables.
In particular, the key, security_gid, and subnet_id of this section.

```
# AWS Config
key=$MASON_KEY
security_gid=$SECURITY_GID # Set this up via AWS Security Group
subnet_id=$SUBNET_ID # Set this up in AWS VPC -> Subnet
```

Launch aws-setup.sh

The script is located in `distributed/script/aws-setup.sh`

On success, it'll tell you how to log into your node and you can start running experiments from there. This script generates the IP addresses of all the nodes in your cluster.

The way to use it is the following:
`./aws-setup.sh [# of instances to generate] [name of the run]`

Move JARs to the AWS nodes

Once the Setup is done, move the jar that you want to run onto the AWS nodes using SCP command like the following -

```
scp -i ~/.ssh/mason.pem ~/Projects/mason-stable/contrib/distributed/target/DHeatbugs.jar "target-node":~/
```

MPI Run

Next SSH into the root node and run your experiment using mpirun command giving it the mpi_hosfile. For example -

```
ssh ubuntu@ec2-1-1-1-1.compute-1.amazonaws.com -i ~/.ssh/mason.pem
mpirun -hostfile mpi_hostfile -n 4 java -jar DHeatbugs.jar
```

When you're done with experiments: Launch aws-teardown.sh

The script is located in `distributed/script/aws-teardown.sh`

The script will tell AWS to shutdown and destroy your instances. For the sake of your income, I'd really recommend not forgetting to do this.

Developing your own application

A word about assumptions made about Distributed Computing

Distributed Computing is when you use multiple machines to act as a single service, allowing you to fake a computer with supremely high specs. However, realistically, this illusion of a single service is shielded only by what is known as a brittle abstraction. The reason is that it is not feasible for the user to be totally blind to the fact that distribution is occurring-- factors like geolocation and physical limitations of the machine can easily reveal themselves under the circumstances. However, it's generally the case that you would not want to do a total shielding under most circumstances (ie going to a busier server in the US might be better than going to an idle server in Europe if you live in the US-- depending on the user's use case).

Required Reading

- Documentation for MASON

Converting an existing MASON application to Distributed MASON

A Quick Guide to Conversion

Add

import sim.util.Timing, sim.util.MPITest, mpi.*;

Convert

SimState → DSimState
DoubleGrid2D → NDoubleGrid2D
SparseGrid2D → NObjectGrid2D
Continuous2D → NContinuous2D
doLoop → doLoopMPI

In General, there are distributed counter parts for various fields in MASON.

Change

Anything that requires direct access to an array to use Getters and Setters
Be careful not to accidentally overwrite any interfaces.
Add anything that needs to be on separate processes in some form like -

schedule.scheduleRepeating(Schedule.EPOCH, n, new <Instance>(), m);
Or
schedule.scheduleOnce(object, n);

An explanation of the model used

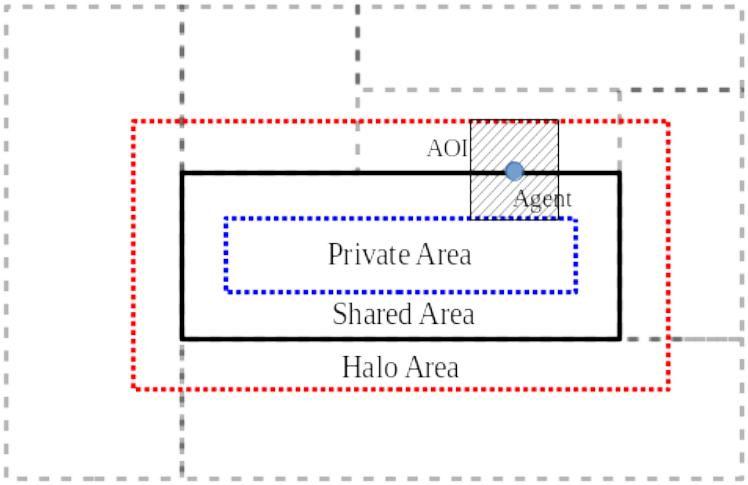
The Logical Model

Designed by ISISLAB-UNISA and George Mason University MASON team, the following model was introduced.

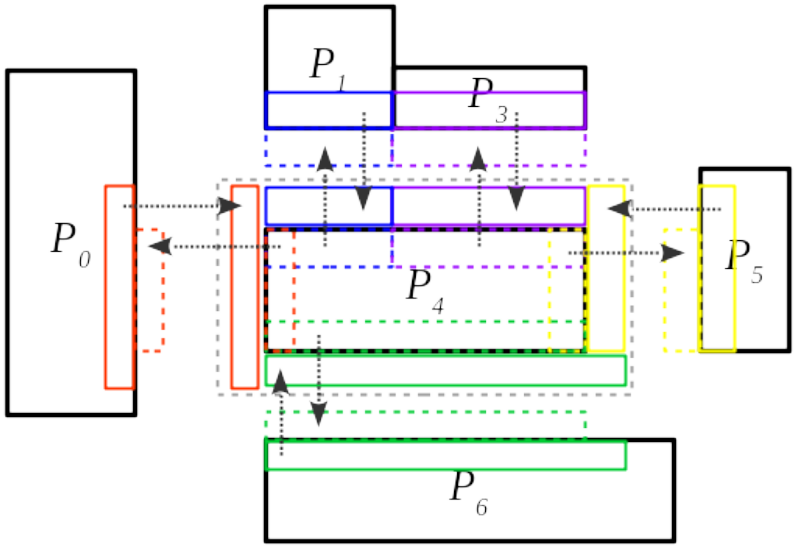
Distributed MASON overhauls the fields and scheduling mechanisms of MASON. Since the scheduling and moving mechanisms must happen together for repeating agents thus, we have tightly coupled the two here. As such, every field must extend the HaloField class and every schedule repeating agent must be registered to iterativeRepeatRegistry within DSimState.

The Field

It divides the field into several regions. Each processor node takes one region, stores all the data and processes all the agents within that region. The distributed MASON is currently optimized for the models that have a “flat” topology where agents interact primarily with nearby agents/data. Specifically, each agent is associated with a location in the field and has a “Area of Interest” (AOI) around it. The agent can read values/status of other agents within in its AOI but can only modify the value at its location and/or change its own internal status/location. When an agent sets its location to somewhere outside its current processor, the agent will be migrated to its target processor and scheduled there.



To accommodate for such a scenario, besides the information within its own region, each node also stores an AOI-wide “Halo” area around its region so that when an agent at the edge of one processor node wants to access the information outside the region, it does not need to go through the network to request the information from the neighbor nodes. Information cached in the Halo area needs to be synchronized once any changes are made in its source node. Such an operation is called “Halo Exchange” and it is performed after each round in distributed MASON.



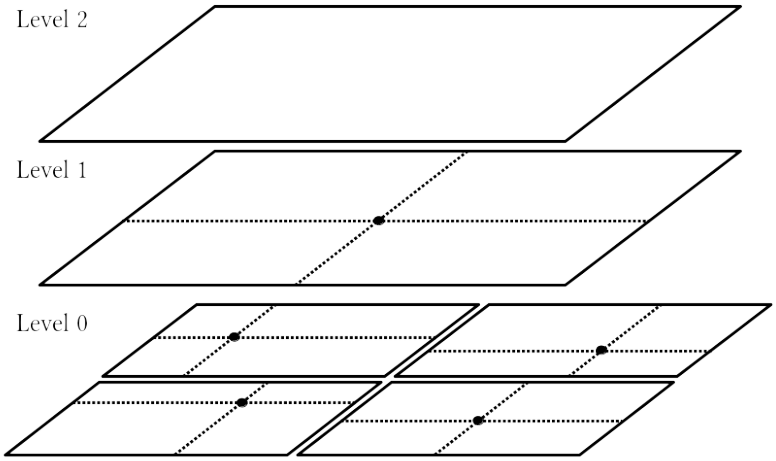
In a more tangled scenario where agents do need to access data outside the AOI, such remote access is also supported but not well-optimized in terms of performance and scalability.

Partitioning

Class sim.field.DQuadTreePartition implementing the interface DPartition oversee dividing the field into regions, identifying the neighboring nodes, creating MPI communication topology between neighbors and adjusting the partition. Currently one processor holds one and only one region. All the processors hold the same copy of the partition scheme and any changes to partition scheme will be synchronized across all nodes.

Here we assume that all the fields follow the same partition scheme. In other words, no matter how many grids/continuous fields an application may have and how the objects are distributed in those fields, they are getting partitioned in the same way. Therefore, the DPartition is expected to be a singleton and once it is initialized, the type of partition QuadTree cannot be changed.

DQuadTreePartition is implemented on top of sim.field.DNonUniformPartition and it partitions the field using a QuadTree. Besides setting up the communication topology with its neighbors, it also sets up group communication at different levels so that the group of immediate siblings (cousins / second cousins/...) can communicate with each other. This is typically used in hierarchical load-balancing.



Transporting and Migrating

Moving across process nodes is called Transporting while moving and scheduling an agent is called Migrating within the code. For most cases it is safer to move it using DRemoteTransporter's transport and migrate methods. However, we also provide direct access through RMI methods within the various fields.

When an agent set its location in a field somewhere outside its current processor's region, the field will find out which neighbor this agent should go and add the agent to that neighbor's queue in the DRemoteTransporter by calling one of the migrate methods. After each round, the sync() method of the migrator will be called to exchange the migrating agents with neighbors. The received agents will be put into a queue and the receiving field will read the agents from the queue and schedule it.

Remote Partition Access

Two classes sim.field.RemoteProxy and sim.field.RemoteField are added to provide a very Preliminary way for one node to access data located in arbitrary node. It is implemented using RMI. In initialization of the RemoteProxy, the master node (pid = 0) will create an RMI registry and broadcast the ip and port number to the rest of the nodes. Then all the nodes will connect to the registry and register the field with a random name. The names from each node will then be broadcast to every other node so that every node knows how to find any other node in the registry. Finally, each node will get a RemoteField handle of every other node. During the simulation, if an agent tries to access the data outside its current region, the data will be obtained by finding the corresponding remote node and calling get on its RemoteField handle.

The HaloField

All fields in distributed MASON must extend this class.

sim.field.HaloField abstract class provides the skeleton of the "Halo Exchange". By extending it and assigning appropriate type of sim.field.storage, it can handle double/int/object grid or continuous field.

The halo exchange is implemented in the sync() function. Also, collect()/distribute() and collectGroup()/distributeGroup() are implemented to enable data collection or distribution to/from one node. It also provides various stabbing queries of determining which area the given point is.

The inner class sim.field.HaloField$Neighbor contains the information about which part of my region should be sent out to which neighbor and where I should put the received data from my neighbors into my halo area.

reload() function is called when the underlying partition scheme is changed. The reload() function will update the internal status as necessary and recreate instances of sim.field.HaloField$Neighbor. It will also rearrange the data in the storage by calling field.reshape(haloPart). Here you may notice that there are no addition data transmissions with other nodes. This is because here we assume that each adjustment of the partition will not go beyond the AOI, which means in case of region expansion, the data needed already exists in the Halo area, avoiding additional data transmissions.

For convenience, HaloField also provides moveAgent and moveRepeatingAgent methods that handle migrating if need be. Thus, the recommended way to work with Agents is to use the moveAgent methods exposed by the various fields. Using the HaloField or extending classes means that the modeler will not have to worry about scheduling, registering or migrating Agents as the HaloField class will handle everything for them.

MPI Abstraction

sim.util.MPITest

Provide a few utilities to help print out debugging information during testing

sim.util.MPIParam

Create an MPI datatype that can be used to extract any N-dimensional rectangular subarray from a 1-d array representing a n-d grid. This is used when pack/unpack data in the Halo Exchange.

sim.util.GroupComm

Create a MPI communicator that includes a subset of all the nodes. This is used in the hierarchical load balancer to collect/distribute data at a certain level

sim.util.MPIUtil

Wrappers for MPI calls. MPI calls usually require that both the sender/receiver know the size of the message. However, in case of sending serialized objects, the receiver is unaware of the exact size. In MPIUtil, we provide wrappers that exchange the exact size of the variable length payload first and then do the actual MPI calls.

JavaDocs for important Classes

sim.field

Class HaloField<T extends java.io.Serializable,P extends NdPoint,S extends GridStorage>

All Implemented Interfaces:

`java.rmi.Remote, DField<T,P>, RemoteField<T,P>`

Direct Known Subclasses:

`NContinuous2D, NDoubleGrid2D, NIntGrid2D, NObjectGrid2D, NObjectsGrid2D`

Method Summary

Modifier and Type	Method and Description
void	**add**(**P** p, **T** t) Adds Object t to location p The location can be remote
void	**addAgent**(**P** p, **T** t) Adds and schedules an agent.
void	**addAgent**(**P** p, **T** t, int ordering, double time) Adds and schedules an agent.
void	**addRepeatingAgent**(**P** p, **T** t, double time, int ordering, double interval) Adds, schedules and registers a repeating agent.
void	**addRepeatingAgent**(**P** p, **T** t, int ordering, double interval) Adds, schedules and registers a repeating agent.
void	**addRMI**(**P** p, **T** t) Used internally for RMI
void	**collect**(int dst, **GridStorage** fullField)
void	**collectGroup**(int level, **GridStorage** groupField)
void	**distribute**(int src, **GridStorage** fullField)
void	**distributeGroup**(int level, **GridStorage** groupField)
int	**getHeight**()

GridStorage	**getStorage**()
int	**getWidth**()
boolean	**inGlobal**(**IntPoint** p)
boolean	**inHalo**(**P** p)
void	**initRemote**()
boolean	**inLocal**(**P** p)
boolean	**inLocalAndHalo**(**P** p)
boolean	**inPrivate**(**P** p)
boolean	**inShared**(**P** p)
void	**move**(**P** fromP, **P** toP, **T** t) The location can be remote
void	**moveAgent**(**P** fromP, **P** toP, **T** t) Moves and schedules an agent.
void	**moveAgent**(**P** fromP, **P** toP, **T** t, int ordering, double time) Moves and schedules an agent.
void	**moveLocal**(**P** fromP, **P** toP, **T** t)
void	**moveRepeatingAgent**(**P** fromP, **P** toP, sim.engine.IterativeRepeat iterativeRepeat) Moves and schedules a repeating agent.
void	**moveRepeatingAgent**(**P** fromP, **P** toP, **T** t) Moves and schedules a repeating agent.
void	**reload**()
void	**remove**(**P** p) Removes all Objects from location p The location can be remote
void	**remove**(**P** p, **T** t) Removes Object t from location p The location can be remote

void	**removeAndStopRepeatingAgent**(**P** p, sim.engine.IterativeRepeat iterativeRepeat) Removes and stops a repeating agent.
void	**removeAndStopRepeatingAgent**(**P** p, **T** t) Removes and stops a repeating agent.
void	**removeRMI**(**P** p) Used internally for RMI
void	**removeRMI**(**P** p, **T** t) Used internally for RMI
double	**stx**(double x)
int	**stx**(int x)
double	**sty**(double y)
int	**sty**(int y)
void	**syncHalo**()
double	**tdx**(double x1, double x2)
double	**tdy**(double y1, double y2)
IntPoint	**toLocalPoint**(**IntPoint** p)
java.lang.String	**toString**()
double	**toToroidal**(double x, int dim)
int	**toToroidal**(int x, int dim)
IntPoint	**toToroidal**(**IntPoint** p)
double	**toToroidalDiff**(double x1, double x2, int dim)

sim.engine

Class DSimState

- java.lang.Object
  - sim.engine.SimState
    - sim.engine.DSimState

Method Summary

Modifier and Type	Method and Description
static void	**doLoopMPI**(java.lang.Class<?> c, java.lang.String[] args)
static void	**doLoopMPI**(java.lang.Class<?> c, java.lang.String[] args, int window)
sim.engine.IterativeRepeat	**getIterativeRepeat**(sim.engine.Steppable steppable)
boolean	**isDistributed**()
void	**preSchedule**()
int	**registerField**(**HaloField**<? extends java.io.Serializable,? extends**NdPoint**,? extends **GridStorage**> haloField) All HaloFields register themselves here. Do not call this method explicitly, it's called in the HaloField constructor
void	**registerIterativeRepeat**(sim.engine.IterativeRepeat iterativeRepeat) Adds the given iterativeRepeat to the Registry
void	**start**()
sim.engine.IterativeRepeat	**stopIterativeRepeat**(sim.engine.IterativeRepeat iterativeRepeat) Removes the given iterativeRepeat from the Registry and calls stop on it
sim.engine.IterativeRepeat	**stopIterativeRepeat**(sim.engine.Steppable steppable) Removes iterativeRepeat corresponding to the given steppable from the Registry and calls stop on it
sim.engine.IterativeRepeat	**unRegisterIterativeRepeat**(sim.engine.IterativeRepeat iterativeRepeat) Removes the given iterativeRepeat from the Registry
sim.engine.IterativeRepeat	**unRegisterIterativeRepeat**(sim.engine.Steppable steppable) Removes iterativeRepeat corresponding to the given steppable from the Registry

Class DRemoteTransporter

Method Summary

Modifier and Type	Method and Description
void	**clear**()
void	**migrateAgent**(sim.engine.AgentWrapper agentWrapper, int dst) Internal method.
void	**migrateAgent**(sim.engine.AgentWrapper agentWrapper, int dst,**NdPoint** loc, int fieldIndex) Transports the Object as well as migrates it
void	**migrateAgent**(int ordering, double time, sim.engine.Steppable agent, int dst) Does not transport the Object, only migrates it
void	**migrateAgent**(int ordering, double time, sim.engine.Steppable agent, int dst, **NdPoint** loc, int fieldIndex) Transports the Object as well as migrates it
void	**migrateAgent**(int ordering, sim.engine.Steppable agent, int dst) Does not transport the Object, only migrates it
void	**migrateAgent**(int ordering, sim.engine.Steppable agent, int dst,**NdPoint** loc, int fieldIndex) Transports the Object as well as migrates it
void	**migrateAgent**(sim.engine.Steppable agent, int dst) Does not transport the Object, only migrates it
void	**migrateAgent**(sim.engine.Steppable agent, int dst, **NdPoint** loc, int fieldIndex) Transports the Object as well as migrates it
void	**migrateRepeatingAgent**(sim.engine.IterativeRepeat iterativeRepeat, int dst) Does not transport the Object, only migrates it
void	**migrateRepeatingAgent**(sim.engine.IterativeRepeat iterativeRepeat, int dst, **NdPoint** loc, int fieldIndex) Transports the Object as well as migrates it.
void	**reload**()

int	**size**()
void	**sync**()
void	**transportObject**(java.io.Serializable obj, int dst, **NdPoint** loc, int fieldIndex) Transports the Object but doesn't schedule it.

sim.field

Class HaloField<T extends java.io.Serializable,P extends NdPoint,S extends GridStorage>

- java.lang.Object
  - sim.field.HaloField<T,P,S>
- **Type Parameters:**

  T - The Class of Object to store in the field

  P - The Type of NdPoint to use

  S - The Type of Storage to use

  All Implemented Interfaces:

  java.rmi.Remote, DField<T,P>, RemoteField<T,P>

  Direct Known Subclasses:

  NContinuous2D, NDoubleGrid2D, NIntGrid2D, NObjectGrid2D, NObjectsGrid2D

All fields in distributed MASON must extend this class.

Method Summary

Modifier and Type	Method and Description
void	**add**(**P** p, **T** t) Adds Object t to location p The location can be remote
void	**addAgent**(**P** p, **T** t) Adds and schedules an agent.
void	**addAgent**(**P** p, **T** t, int ordering, double time) Adds and schedules an agent.
void	**addRepeatingAgent**(**P** p, **T** t, double time, int ordering, double interval) Adds, schedules and registers a repeating agent.

void	**addRepeatingAgent**(**P** p, **T** t, int ordering, double interval) Adds, schedules and registers a repeating agent.
void	**addRMI**(**P** p, **T** t) Used internally for RMI
void	**collect**(int dst, **GridStorage** fullField)
void	**collectGroup**(int level, **GridStorage** groupField)
void	**distribute**(int src, **GridStorage** fullField)
void	**distributeGroup**(int level, **GridStorage** groupField)
int	**getHeight**()
GridStorage	**getStorage**()
int	**getWidth**()
boolean	**inGlobal**(**IntPoint** p)
boolean	**inHalo**(**P** p)
void	**initRemote**()
boolean	**inLocal**(**P** p)
boolean	**inLocalAndHalo**(**P** p)
boolean	**inPrivate**(**P** p)
boolean	**inShared**(**P** p)
void	**move**(**P** fromP, **P** toP, **T** t) The location can be remote
void	**moveAgent**(**P** fromP, **P** toP, **T** t) Moves and schedules an agent.
void	**moveAgent**(**P** fromP, **P** toP, **T** t, int ordering, double time) Moves and schedules an agent.
void	**moveLocal**(**P** fromP, **P** toP, **T** t)

void	**moveRepeatingAgent**(**P** fromP, **P** toP, sim.engine.IterativeRepeat iterativeRepeat) Moves and schedules a repeating agent.
void	**moveRepeatingAgent**(**P** fromP, **P** toP, **T** t) Moves and schedules a repeating agent.
void	**reload**()
void	**remove**(**P** p) Removes all Objects from location p The location can be remote
void	**remove**(**P** p, **T** t) Removes Object t from location p The location can be remote
void	**removeAndStopRepeatingAgent**(**P** p, sim.engine.IterativeRepeat iterativeRepeat) Removes and stops a repeating agent.
void	**removeAndStopRepeatingAgent**(**P** p, **T** t) Removes and stops a repeating agent.
void	**removeRMI**(**P** p) Used internally for RMI
void	**removeRMI**(**P** p, **T** t) Used internally for RMI
double	**stx**(double x)
int	**stx**(int x)
double	**sty**(double y)
int	**sty**(int y)
void	**syncHalo**()
double	**tdx**(double x1, double x2)
double	**tdy**(double y1, double y2)
IntPoint	**toLocalPoint**(**IntPoint** p)
java.lang.String	**toString**()

```
double              toToroidal(double x, int dim)

int                 toToroidal(int x, int dim)

IntPoint            toToroidal(IntPoint p)

double              toToroidalDiff(double x1, double x2, int dim)
```