**The STEM Hadoop Workflow (HWF):**

**Azure Deployment and Operation**

Nicholas E. Bruns and Daniel Fink

10.26.2015

**Section 0: Organization Overview**

The general goal of this project is to port the STEM workflow to Hadoop. This workflow needs to generate STEM species distribution models for one species at time, across the North Of The Equator (NOTE) spatial extent for single years or averaging across multiple years of training data. Each STEM analysis job requires

* Model training and predictions made for 200 data folds or subsamples,
* Generating 52 weekly predictions across the 3km SRD locations with visualizations, and
* Generating weekly predictive performance assessment data and plots.

The goal of this document is to provide instructions and documentation for the deployment and operation of the STEM Hadoop WorkFlow (or HWF) on Microsoft’s Azure cloud platform’s Hadoop product, HDInsight. It is organized sequentially into two sections,

1. Cluster deployment and data preparation followed by
2. Workflow operation – the MapReduce steps - for a single STEM job.

Section A on Cluster deployment and data preparation is broken into the following subsections:

A1: Azure Initializations

A2: Local Initializations: RSA key setup

A3: Prepare eBird ERD & SRD data

A4: Azure Cluster Deployment

A5: Finish RSA Key Configuration Connecting Local Machine and Azure Cluster

A6: Check R Installation on Azure Cluster

A7: Transfer Files to Cluster Head Node via SCP

A8: Transfer Data Files to Storage Account – Via Transfer Cluster

The organization of Section B follows directly from the computational workflow:

B1: MR1 Modeling Step: training and prediction of base models

B2: MR2: Summarization Step: summarize predictions across folds

B3: Package output / results from MR2

B4: Data Prep For MR3

B5: MR3 Threshold selection and occurrence predictive performance assessment

B6: Package MR3 Results

Each step is a subsection in this document and includes demonstration scripts and documented code. Currently, the documentation varies in scope and detail, often reflecting the importance or complexity of the task! Ideally, each section will address applicable topics including: inputs, outputs, computational strategy, profiling and timing, extensions and next steps.More detailed descriptions of computational architecture, data structures, etc. will be organized in additional sections. Other information that we have found useful has been gathered into a series of appendices, listed here:

* Profiling
* “By-Hand” Installation of R and R packages on Azure HDI Cluster
* Adding R Packages to Cluster
* Ambari Cluster Monitoring (advanced)
* Azure Storage Account Theory
* File transfers: FTP Client Cyberduck
* File transfer: Azure Command-line Interface (ACI)
* AZURE Copying files across storage accounts

**Section A: Deployment Overview**

* The cluster deployment steps occur on different environments: the home cluster and on Azure.
* Pay attention to changes in environment (local machine, internet, back to local, ssh'd into the cluster)
* When using this text, do a (carefuly) search and replace for these two values, which you'll set at cluster deployment time:
  + - * -<cluster\_name>
      * -<cluster\_password>

The first initialization steps are only necessary when starting from “scratch”. These steps will persist across jobs.

**Section A1: Azure Initializations (Azure one time only)**

These steps are needed to set up a new project or move between Azure regions.

1. Sign in at portal.azure.com
2. Create "Resource Group"

You will need to select one when creating storage account. The resource group persists, so it only needs to be created initially or when changing regions.

1. Create Storage Account

In theory this only needs to be done once since the BLOB storage will persist (See Appendix on Azure Storage for more information about BLOB storage)

portal.azure.com menu path is: New > Data + Storage > Storage Account

\*URL = Name of the storage BLOB (e.g. "ebirddata" or "stemdata" or ”sd2”)

\*Location - must match HDIsight cluster locations

\*Replication - Locally Redundant

After the storage account is set up, its a good idea to get a copy of the primary access key to make it easier to interact with the BLOB.

1. Create “scripts” and “runs” container

I believe this can be done in the portal UI. Check this! Note, it may be *necessary* to create an initial container on Web UI. My current experience is that without any containers, the CLI and CyberDuck can not interact with the storage account - i.e. can not create containers or upload information. I need to verify this.

1. Upload the "ebird-action-script.sh" to the “scripts” container. This action script will be used to install R and the R packages on the cluster. It was stable but in mid October 2015, during the penning of this document, these failed. A fix is being developed.

**Section A1.5: Prepare R libraries for Azure Script Installation**

Begin by running the Azure script when spinning up a small cluster. It is not completely necessary to run the script when spinning up the cluster, but it’s a starting point! The point is that the libraries need to be compiled on the same machines as the large cluster.

Get onto a small cluster to Prepare the R Libraries

mkdir R\_Libs

Start R

install.packages(

pkgs=c('mgcv', 'mboost', 'pryr','PresenceAbsence', 'verification', 'gbm', 'scam', 'fields'),

lib = "~/R\_Libs/",

repos="http://cran.revolutionanalytics.com",

INSTALL\_opts=c('--byte-compile') )

Check Installation

.libPaths("~/R\_Libs/")

library(gbm)

Copy Compiled Libraries to Destination

sudo cp -r ~/R\_Libs/\* /usr/local/lib/R/site-library/

Change Permissions to "drwxrwxr-x"

chmod u=rw id\_rsa\_clusters.pub

chmod g=rwx /usr/local/lib/R/site-library/\*

Create Tarball

cd R\_Libs/

tar -cvf eBird-r-libraries.tar \*

Move tarball to Storage Account, in the “scripts” container.

I did this via CyberDuck. Note: Change permissions for public access. This is an important step because the container defaults to private permissions. I changed the permissions for the “scripts” container on the “SD2” account to be “blob” public. Note that permissions will persist with the storage account so this step will not need to be repeated each time. This link describes permissions, but not how to modify them! <https://azure.microsoft.com/en-us/documentation/articles/storage-manage-access-to-resources/>

Edit the script: eBird-r-install-launcher-v01.sh. Make sure that these lines point to the right script and tarball.

# Download packages.

download\_file https://sd2.blob.core.windows.net/scripts/eBird-r-libraries.tar /tmp/eBird-r-libraries.tar

untar\_file /tmp/eBird-r-libraries.tar /usr/local/lib/R/site-library/

# Remove temporary files.

rm -f /tmp/eBird-r-libraries.tar

**Section A2: Local Initializations: RSA key setup (local one time only)**

To conveniently transfer files between machines (e.g. between local machines & Azure storage or between head and worker nodes on a cluster) we use ssh key authentication To do this you need an RSA key you feel comfortable copying the private portion onto your cluster. I have created a separate key for Azure use, that I call id\_rsa\_clusters. Paste the contents of the id\_rsa\_cluster.pub file into the web UI when instantiating a cluster. The private portion is scp'd up to the cluster head node once it exists (post deployment). The demo script shows the steps needed to create this key, which only needs to be done once on your local machine and can then be used on all instantiated clusters. Note, Ambari, the cluster monitoring software, unfortunately requires the use of ssh tunneling, which is a process more involved than this simple key sharing and is discussed in a weedy appendix at the end of the document.

#-------------------------------------------------------------------------------

# 0, RSA key settup, only need once

# --local machine--

#-------------------------------------------------------------------------------

## important! in copying over rsa's, Im using not my default id\_rsa, but a custom one I made, which is called:

# ~/.ssh/id\_rsa\_clusters

# Creating this is not straightforward

cd ~/.ssh/

# make a file, into which the ssh generation will write

touch id\_rsa\_clusters

ssh-keygen

##you've be prompted with:

### Enter file in which to save the key (/Users/nicholasbruns/.ssh/id\_rsa):

# enter, with complete and not relative path:

# id\_rsa\_clusters

## skip using a passphrase, ie just hit enter when prompted when asked

##be sure the permissions are correct, else it won't work on the cluster

# -rw-r--r-- for id\_rsa\_clusters.pub

# -rw------- for id\_rsa\_clusters

## Here are the unix commands to change permissions

chmod u=rw id\_rsa\_clusters

chmod g= id\_rsa\_clusters

chmod o= id\_rsa\_cluster

chmod u=rw id\_rsa\_clusters.pub

chmod g=r id\_rsa\_clusters.pub

chmod o=r id\_rsa\_clusters.pub

**Section A3: Prepare eBird ERD & SRD data**

Task Overview:

1. On ATLAS:
   1. -initalize a stem\_v3 run with the desired spatial/temporal extent
   2. -process and repackage stem\_v3 ingredients, including fusing into produce

Task Inputs:

A run name -for this species/extent analysis (lower case, and dashes only, no underscores or periods)

Erd/srd processing environment (for now, the ATLAS cluster), with:

-stem\_v3 code base

-stem\_hwf codebase, namely:

source/prep\_date.R

-bigmem files for ERD and SRD

Task Output:

ebird.abund\_<job\_name>\_all.random.merge.txt

fold\_offsets.csv

Task Instructions: Create ERD & SRD for Streaming Hadoop WF

# get on Atlas!

ssh <netid>@atlas.cac.cornell.edu

# Open an interactive session (be kind, dont do work on the head node!)

qsub -I -q inter -l nodes=1:ppn=12

#-------------------------------------------------------------------------------

# Prep a stem\_v3 run

#-------------------------------------------------------------------------------

# set up run parameters

$ cd ~/stem\_hwf/wf.control/

$ vim template\_params.R (or open with another text editor)

# use species names formatting (ie underscores, captilization and spelling) as found in taxonomy.txt file, shipped with erd!

# fields of interest, with settings at time of writing:

# -lines 25 and 26:

# species.common.name <- "Northern\_Pintail"

# species.sci.name <- "Anas\_acuta"

# -line 28

# extent.string <- "nth\_amer"

# -line 47 and 48

# erd.date.begin <- 2004

# erd.date.end <- 2014

# when ready, run extraction routine. This script creates a working space for the run, room for data and results along with a copy of the runs unique paramter file. Then it populates the data subdirectory. We'll take those products and massage them into a Hadoop ready form

#----------------------------------------------------------------------------

cd ~/stem\_hwf/wf.control/

$ R

> source('init\_run.R')

##you will be prompted to supply a name for this run

# supply a run name: [ Species common name is a good option]

#hit return, and in ~ 3 mins, the run will be ready.

> q()

#inspect handiwork if you'd like

ls ../runs/

#-------------------------------------------------------------------------------

# caste stem\_v3 inputs into one Hadoop ready text file

#-------------------------------------------------------------------------------

cd ~/stem\_hwf/source

$vim prep\_data.R

### be sure all the hard coding is ok for your job names!

## namely, check out:

# Line 13 This is parent directory for HWF code & runs

# ebird\_abund\_dir <- paste(work.station\_root,"~/stem\_hwf/",sep="")

# line 15 area:

# run.name <- "TRES\_nth.amer" "Long-billed\_Cerlew"

# Note - run.name is entered on command line in "init\_run.R"

# Line 34-35

# REP\_N <- 50

# SAMPLE\_N <- 4

# line 42 area:

# This is where the V3 data files live, in the runs directory

# stem\_v3\_munge\_string <- "stem\_abund/runs/TRES\_nth.amer\_smooth.method.run.full/data/nth\_amer\_Tree\_Swallow"

$ R

> source('prep\_data.R')

> q()

## randomly interleave train, test, and srd data.

## this randomizing should make the map step more even across all splits, as SRD points are 52 times more work!

cd ../runs/<run.name>/data

sort -R ebird.abund\_TRES\_nth.amer\_\* -o ebird.abund\_TRES\_nth.amer\_all.random.merge.txt

## you can also gather up a debug sized file, if you'd like it for testing later, though this can be produced on the Hadoop cluster as well if you'd like

sort -R debug.size\_ebird.abund\_\* -o ebird.abund\_TRES\_nth.amer\_all.random.merge\_debug.size.txt

**Section A4: Azure Cluster Deployment**

Create cluster with portal.azure.com UI

Sign in at <https://portal.azure.com>

Create HDI Cluster:

* portal.azure.com menu path is: "New" > "Data + Analytics" > "HDInsight"
* Cluster Name: large-cluster or transport-cluster
* Cluster Type: Hadoop
* Cluster Operating System: Ubuntu 12.04LTS for operating system
* Subscription: Our CU enterprise account is called "CLO eBird subscription"
* Resource Group: Select existing resource group. This will determine the Data Source and the Region. NOTE: your storage account and HDI cluster regions must match!
* Credentials:
  + Cluster login username: This defaults to "admin". Look out for confusing parts down the line where a username is asked for that is not this name, "hdiuser", but rather "admin" (most likely the "HTTP user name which is used for Ambari").
  + "Cluster Login Password": Nick has been using <cluster\_password> In theory, we will not need to use this often because of the SSH setup.
  + "SSH Username": I have been using the name "hdiuser", for consistency in scripts.
  + "SSH Authentication type" switch SSH Authentication type to Public key-- then supply said Public key. Get a copy of the public key. Easy way, copy and paste in this box the output (on local machine) of:

$ cat ~/.ssh/id\_rsa\_clusters.pub

* + "Node Pricing Tiers" for picking nodes types
    - Test or Transfer Cluster Size: 2 head nodes and 4 worker nodes using A3, the smallest available on HDI with 4 cores and 7GB ram. These are not MS's recommended, but they are the cheapest!
    - Large Cluster Size: 2 x D13 head nodes and up to 250 x D4 worker nodes.
    - *Important: clusters with more than 35 nodes require head nodes with at least 14 GB RAM!*
  + Optional Configuration:
  + "Script Actions"
    - Name: eBird-R-installation
    - Script URI: <https://sd2.blob.core.windows.net/scripts/eBird-r-install-launcher-v01.sh>
    - NOTE this script needs: <https://sd2.blob.core.windows.net/scripts/eBird-r-libraries.tar>
    - Check Head and Worker, you can also check Zookeeper

See section XXX to upload scripts

Then you wait ~20 minutes for a small cluster and more like 45 min for a large cluster with R and R packages installed.

**Section A5: Finish RSA Key Configuration Connecting Local Machine and Azure Cluster**

On local machine, create "~/.ssh/config" Note that bitbucket has a nice explanation for setting up this config file.

touch config

Permissions for config should be (I think) "-rw-r--r--"

Go into config and add this line to use your non-default private key for this server:

Host large-cluster-ssh.azurehdinsight.net

User <user name, "hdiuser" as specified in gui>

IdentityFile ~/.ssh/id\_rsa\_clusters

Copy to the cluster the public key.

Doing this allows the head node to conveniently talk to all worker nodes.

NOTE: To reuse the same cluster name, you need to delete the corresponding entry in the ~/.ssh/known\_host file.

scp ~/.ssh/id\_rsa\_clusters/ hdiuser@stem-cluster-ssh.azurehdinsight.net:~/.ssh/id\_rsa

scp ~/.ssh/id\_rsa\_clusters/ hdiuser@small-cluster-ssh.azurehdinsight.net:~/.ssh/id\_rsa

scp ~/.ssh/id\_rsa\_clusters/ hdiuser@large-cluster-ssh.azurehdinsight.net:~/.ssh/id\_rsa

**Section A6: Check R Installation on Azure Cluster**

Check that it is has the HWF files and R installed on the head nodes:

ssh hdiuser@stem-cluster-ssh.azurehdinsight.net

ssh hdiuser@small-cluster-ssh.azurehdinsight.net

ssh hdiuser@large-cluster-ssh.azurehdinsight.net

ls -lah

R

You can test to make sure that R has been installed on various worker nodes too.

# The simplest way is to use Ambari call from the head node.

# This command will generate a list ithat inlcudes identification for the type of node

# and the IP, though embedded within the XML.

# (See below for more information about how to generate a list of node IPs.)

curl -u admin:1Northamer! http://localhost:8080/api/v1/hosts?fields=Hosts/ip

# Choose some worker nodes, ssh into them and see if they have R.

ssh 10.0.0.7

**Section A7: Transfer Files to Cluster Head Node via SCP**

Copy up source code from local machine

cd ../source

Now move up to cluster next step

scp mapper.1.R reducer.1.R reducer.1.modeling.library.abundance.R mapper.2.R reducer.2.R [hdiuser@large-cluster-ssh.azurehdinsight.net:~/](mailto:hdiuser@large-cluster-ssh.azurehdinsight.net:~/)

Transfer fold\_offsets.csv to the head node of the cluster

cd ~/stem\_hwf/runs/Willet/data

scp fold\_offsets.csv [hdiuser@large-cluster-ssh.azurehdinsight.net:~/](mailto:hdiuser@large-cluster-ssh.azurehdinsight.net:~/)

Here is what the directory on head node should look like:

-rw-r--r-- 1 hdiuser hdiuser 11274 Oct 14 17:11 fold\_offsets.csv

-rwxr-xr-x 1 hdiuser hdiuser 5142 Oct 14 17:06 mapper.1.R

-rwxr-xr-x 1 hdiuser hdiuser 1192 Oct 14 17:06 mapper.2.R

-rw-r--r-- 1 hdiuser hdiuser 675 Apr 3 2012 .profile

-rw-r--r-- 1 hdiuser hdiuser 20134 Oct 14 17:06 reducer.1.modeling.library.abundance.R

-rwxr-xr-x 1 hdiuser hdiuser 8129 Oct 14 17:06 reducer.1.R

-rwxr-xr-x 1 hdiuser hdiuser 5796 Oct 14 17:06 reducer.2.R

**Section A8: Transfer Data Files to Storage Account – Via Transfer Cluster**

Transfer ebird.abund\_<SPP NAME>\_nth.amer\_all.random.merge.txt to the target on the Azure storage account, a "subdirectory" under the cluster's default container. We have been setting the default container to "runs" then use the Species name or run name for the subdirectory (see above to create subdirectory).

1 on AZURE) spin up a small cluster for the sole purpose of transfering the data. Here called: small-cluster

2 on AZURE) be sure you''ve made a target subdirectory for the file on the storage account. E.g. I created /Willet\_nth.amer/ under the /runs/ directory / container using CyberDuck. I was not able to do this with the ACI tool.

3 on ATLAS) cd to where the prepped data lives

cd ~/stem\_hwf/runs/Willet/data

4 on ATLAS) add cluster to the ssh config (the id\_rsa\_cluster file is already on Atlas, if it weren't, you'd add it)

Host small-cluster-ssh.azurehdinsight.net

User hdiuser

IdentityFile ~/.ssh/id\_rsa\_clusters\_df

5 on ATLAS) scp file to cluster local drive

scp ebird.abund\_Willet\_nth.amer\_all.random.merge.txt hdiuser@small-cluster-ssh.azurehdinsight.net:~/

6 on AZURE) Now that the file is here, ssh into the small-cluster and then copy it to blob!

ssh hdiuser@small-cluster-ssh.azurehdinsight.net

hadoop dfs -copyFromLocal ebird.abund\_Willet\_nth.amer\_all.random.merge.txt wasb:///Willet\_nth.amer/

NOTE: Many, if not all?, of the Hadoop command line tools generate this error

DEPRECATED: Use of this script to execute hdfs command is deprecated.

Instead use the hdfs command for it.

All of the commands in this document work despite this warning.

**Section B1: MR1 Modeling Step: training and prediction of base models**

This step involves all stixel based work of model fitting and predictions.

#-------------------------------------------------------------------------------

# ssh onto Deployed cluster and paste the below command to submit job

### Note:

# -- use of compression on final output!

# -- special configuration parameter:

# -D mapreduce.reduce.memory.mb=2048 \

# demands that each container running a reduce task have at least this much memmory.

# This changes concurrent tasks from 4 per node to 2, making the reduces safe.

# See extensions (tighten memmory recquirements, increase concurruency) for further explanation

#-------------------------------------------------------------------------------

export HADOOP\_STREAMING\_JAR=/usr/hdp/current/hadoop-mapreduce-client/hadoop-streaming.jar

export SAMPLE\_N=50

export REP\_N=4

nohup hadoop jar ${HADOOP\_STREAMING\_JAR} \

\

-D mapred.task.timeout=2000000 \

-D mapred.compress.map.output=true \

-D mapreduce.output.fileoutputformat.compress=true \

-D mapreduce.output.fileoutputformat.compress.type="org.apache.hadoop.io.compress.SnappyCodec" \

-D mapreduce.map.output.compress.codec="org.apache.hadoop.io.compress.GzipCodec" \

-D mapred.map.tasks=2000\

-D mapred.reduce.tasks=2000\

-D mapreduce.reduce.memory.mb=2048 \

-input wasb:///Willet\_nth.amer/ebird.abund\_Willet\_nth.amer\_all.random.merge.txt \

-output wasb:///Willet\_nth.amer/200\_fold\_250\_nodes\_2k\_maps\_2k\_reduces \

-mapper mapper.1.R \

-reducer reducer.1.R \

-cmdenv REP\_N=$REP\_N \

-cmdenv SAMPLE\_N=$SAMPLE\_N \

-file /home/hdiuser/mapper.1.R \

-file /home/hdiuser/fold\_offsets.csv \

-file /home/hdiuser/reducer.1.R \

-file /home/hdiuser/reducer.1.modeling.library.abundance.R > 200\_fold\_250\_nodes\_2k\_maps\_2k\_reduces\_bighead.txt &

#-------------------------------------------------------------------------------

# debug

#-------------------------------------------------------------------------------

##

##need a local copy to do this testing!

##why? From storage account, can't do partial reads, and forced to do full cat

hadoop dfs -cat wasb:///tres-nth-amer-abund/ebird.abund\_TRES\_nth.amer\_all.random.merge.txt > debug\_local\_copy\_all\_data.txt

export SAMPLE\_N=1

export REP\_N=1

## quick check

head -2000 debug\_local\_copy\_all\_data.txt | ./mapper.1\_vectorized.R | sort | ./reducer.1.R

This phase is designed to allow truly independent treatment of stixels, exploiting structure provided by STEM's design and statistical theory. STEM's theory treats stixels as independent universes where rules about bird-landscape relationships are zealously formed, and all things outside the stixel boundaries are explicitly ignored. Map assigns points to stixels, and reduce performs model fitting and predictions on points that fell into that stixel.

More specifically, we enter the Map step with an interleaved stream of all data-- training, SRD and test. We pre-devise a set of regular time/space partitionings, for now, 200 of these. We then can work line by line through the data stream, taking each point and determining the stixel this point belongs to in each of the 200 partitions. The mapper emits all of these id instances-- the stixel label is the key, and the relevant point information is the value (location, species count, search process + landcover covariates). SRD points are actually expanded in this step by the weeks of the year, so SRD points produce 200\*52 labels. In short, the map step emits:

key- stixl id value- (data type, location, count, search covariates, landcover covariates)

The reduce step sees all points with this stixel, or, all points in a stixel. If there is enough training data, a model will be fit and then used to make predictions on all data in the stixel. Otherwise, all points receive 5 columns of NA's (to match with the 5 column output of the ZI model). Some may take issue with the term "reduce" in this step, as all points that enter this stage leave with a prediction. Maybe predictions can be considered a reduced representation of the covariates?

inputs:

-on storage account:

ebird.abund\_<job\_name>\_all.random.merge.txt

-on HDI cluster head node:

fold\_offsets.csv

outputs:

Map step:

(the widening)

200 \* obs points

200\* 52 srd points

k: stixel id, hashes on the 3 dimensions (lat lon time)

v: all original row information:

row\_id,

data type,

location information,

search information,

landcover information

... (some example points, post sort)

1-6-5-4 115412,-67.6839297621694,46.6737052678373,0.42,0,2013,NA,7,1,1,1,279,4,0,0,0,0,5.5556,94.4444,0,0,0,0,0,0,0,0,NA,"srd",TRUE

1-6-5-4 115412,-67.6839297621694,46.6737052678373,0.44,0,2013,NA,7,1,1,1,279,4,0,0,0,0,5.5556,94.4444,0,0,0,0,0,0,0,0,NA,"srd",TRUE

1-7-4-2 100162,-73.107302,44.3259771,0.2704918,1,2013,99,7.33,0.25,0,1,106,12,0,0,0,0,30.5556,5.5556,0,0,0,0,27.7778,36.1111,0,0,0,"train",FALSE

1-7-4-2 100794,-72.83204,41.7198,0.273224,0,2013,100,6.92,0.5,1.609,1,81,13,0,0,0,0,47.2222,5.5556,0,0,0,0,8.3333,30.5556,8.3333,0,6,"train",TRUE

Reduce step

(This output is as long as the end of map output)

(Here, the the search and landcover information has been dropped, predictions, added)

- ~ total, 337 GB, compressed (for North America, 200 folds)

- columns:

sample.id

lon

lat

date

obs

bs.sample

pred.pi

pred.tau

pred.mu

pred.truncated

pred.pi.mu

**notes on the computational strategy: stixel id assignment:**

When labeling stixels, note that MapReduce keys need be nothing beyond, within reason, an identical string for all points in a stixel. We use a 4 value key, 3 values are results from a hash-like computations on point location, and the 4 number is the partitioning id (1-200). This labeling now runs in constant time, and points are only sorted once, by the Hadoop framework in the shuffle step between mapping and reducing. In old STEM workflows, we sorted each point

Explained quickly: take location information (lat, lon, time). Divide each by stixel dimensions. The non-remainder result is the same for everything within a block of size dimension. Shift the gridding by adding a constant (0 - partition dimension) to each part of location information. Therefore, a partitioning is stored as the 3 offest constants added to all values, and all partionings can be defined with a (partioning.n x dimensions) matrix.

Explained at length: Membership within regular grid lines can be inferred by the non remainder results of division. The divisor establishes the dimensions. The partitiong, or where these seperating lines fall, can be shifted by simply adding a unique constant to the dimensions before dividing. Partitiongs are then defined as 3 membered sets of offsets for the 3 dimenions.

Here this principle will be illustrated by sorting 4 (A,B,C,D) points along a single dimension.

Take A-89, B-92, C-101, D-106

89--92--------101----106

A B C D

\*Divide by 10,

A-8.9, B-9.2, C-10.1, and D-10.4

\*Drop the decimal (don't round) and sort into groups

[8: A] [9: B] [10: (C, D)]

Inducing this cutting:

89-|-92-------|-101----106|

A B C D

\*\*\*\*\* Now watch as adding 6 to all and repeating will change the groupings.

A-89, B-92, C-101, D-106

... + 6, becomes

A-95, B-98, C-107, D-112

89--92--------101----106

A B C D

becomes:

------95--98--------107----112

A B C D

\*divide by 10

9.5, 9.8, 10.7, 11.2

\*drop decimal and sort into groups, you get:

[A,B] [C] [D]

or, this cutting

------95--98-|------107--|--112

A B C D

vs.

89-|-92-------|-101----106----|

A B C D

Tuning task numbers:

How to choose the correct number of "tasks" your problem is divided into, e.g.

-D mapred.reduce.tasks=1000\

-D mapred.map.tasks=1000\

(note, these parameters with a -D, are generic Hadoop configuration settings and need to be first in your run specification)

On the map side, rows of data are evenly divided into input chunks called splits, of size (input length / task.number). On the reducer side, tasks receives a set of keys to process, each key corresponding to a block of processing. The number of keys per task varies a little bit, though its relatively even.

Task number can affect job performance, but its an empirical sweetspot found by exploration given your problem and chosen architecture—the complexity of the Hadoop architecture precludes finding analytic optima. Certainly, you want these numbers to be at least as large as the number of concurrent tasks your cluster can run (given problem size). Older advice suggested keeping reducer numbers exactly equal to concurrent tasks, so all work and finish in one wave. I believe there are gains from requesting more tasks than that, to a certain point.

-what is too few reduce tasks (when work per task is too much)?

-We want to preserves reasonable write sizes (use compression here, shoot for maybe 500 mb per reducer)

-Waves of smaller writes allows writing to begin processing continues

- what is too many reduce tasks (when work per task is too little)

-ensure some non-NA stixels: some stixels don’t have enough training data to fit models, and all points receive NA for their predicted values. These keys have no model fitting, and thus complete much faster. A

-too few stixels per split, and you get bad amortized performance

-more range in stixels per split than one might think

-some splits, if few stixels, will have hardly any work to do

-suffer performance loss from unnecessary scheduling

these considerations would be important, but in our problem conditions are not determining factors

-be sure stixels are big enough so they take at least 60 seconds

(a less stringent stixel-per-split lower bound than NA distribution)

-ensure full subscription (enough-- at least cores \* nodes to give chance for widest computation)

(a less stringent stixel-per-split upper bound than "don't make writes too big, divide up!")

(especially the scale of our analyses)

some quirks:

Split determination.

I have seen that, if input to a new MR task is an MR output (2nd in a chained job), the number of pieces this job is stored in will be the number of splits. In that case, the files were compressed. (Aug 18th)- 2k were saved, I asked for 1k reducers tasks, but still split to 2k.

extensions

\*\* equally wide -- NA filtering

A very promising optimization seems to be dropping the points with NA predictions. About half of stixels reported in the North American extent seem to be NA's, without suffucient data. Here, pre screen for conditions that make the model predict NA's, and don't pass them forward to the next step. This should lighted up the writes coming out of reducer.1, and the subsequent reads by mapper 2, and shuffle step 2.

\*\*tighten memmory recquirements, increase concurruency

Summary: We impose a memmory constraint that induces 2 reduce tasks running per node of a possible 4. 4 seems to cause failure from memmory contraints, but its unclear if 3 would be ok.

I specify that containers recquire 2048 GB ram. This is greater than the defualt of 1024 GB, as 1024 cuased failure inducing collisions. This value is a mere doubling of the default, and something between 1024 and 2048 might actually be optimal. Namely, it seems like the safe state induced by this adjustment is that now 2 reduce tasks run concurrently per node, and it is unclear if 3 could be possible.

Explained more thorughly:

Note this line in submissions:

-D mapreduce.reduce.memory.mb=2048 \

YARN deploys map or reduce tasks to an independent partition of the available resources-- called a container. The above command demmands that each reduce contatiner have at least 2048 mb of memmory avilable to it, as opposed to the default of 1024. In the HDI settup, YARN will deploy maximial number of concurrent tasks based on the available memmory and cores.

I increased this value after observing failure from outstripping memmory recquirments on each node. I assumed this came from too many concurrent tasks running on the same node. Indeed, the error no longer appeared after increasing memmory per container. It looks like each node now runs 2 concurrent tasks instead of 4.

Note that 2048 is quite arbitrary, and a bit superstitious almost. The default was 1024, and I know sometimes systems components have some kind of relevence for powers of 2's, or that some resources or blocks or resources have some kind of significance of orgainzing in powers of 2...

Also, a quick note of confusion-- notice that the high water memmory recuirement reported is no worse than 910 MB, on the worse stixel. 4 \* 910 is 3.5 GB tough, which is well under 7. So some other overhead seems to be involved, or perhaps the memmory high water mark isn't a true indication of memmory recquired. 4 GB, perhaps, can best be called a low end of recquired memmory.

\*\*vectorize the reducer output

Reducer output recquries some parsing, which I do with looping through predictions. This could of course be cleaned up. That said, it doesn't seem to be the limiting step, which would be the fits and predictions themselves. So I have not made this cleanup a priority.

**Section B2:** **MR2: Summarization Step: summarize predictions across folds**

# MR step 2 (step 4 overall): Stixel Summary

# example submissions

#-------------------------------------------------------------------------------

# debug lines--

## I would say, definately do this before submitting!

#-------------------------------------------------------------------------------

## see if this file already exists!

hadoop dfs -cat wasb:///tres-nth-amer-abund/200\_fold\_40\_nodes\_2k\_maps\_2k\_reduces/part-00016 > debug\_local\_copy\_some\_post\_mr.1.txt

head -2000 debug\_local\_copy\_some\_post\_mr.1.txt | ./mapper.2.R | sort | ./reducer.2.R

# DF From the large-cluster command line:

hadoop dfs -cat wasb:///Willet\_nth.amer/200\_fold\_250\_nodes\_2k\_maps\_2k\_reduces/part-00016 > debug\_local\_copy\_some\_post\_mr.1.txt

head -2000 debug\_local\_copy\_some\_post\_mr.1.txt | ./mapper.2.R | sort | ./reducer.2.R

#-------------------------------------------------------------------------------

# actual submission!

# note:

# -D mapreduce.reduce.log.level=DEBUG \

# \*this was specially inserted by MS consultation, it makes retreiving log files for failed jobs purportely easier. That feature has not yet been used though, and it may fill up the disk with log files on smaller clusters.

# \*likely, this will need to be removed, especially if many reducers 2 tasks have followed through without issue

# --

#-------------------------------------------------------------------------------

export HADOOP\_STREAMING\_JAR=/usr/hdp/current/hadoop-mapreduce-client/hadoop-streaming.jar

export SAMPLE\_N=50

export REP\_N=4

nohup hadoop jar ${HADOOP\_STREAMING\_JAR} \

\

-D mapred.task.timeout=2000000 \

-D mapred.compress.map.output=true \

-D mapred.map.tasks=2000\

-D mapred.reduce.tasks=2000\

-D mapreduce.reduce.log.level=DEBUG \

-input wasb:///Willet\_nth.amer/200\_fold\_250\_nodes\_2k\_maps\_2k\_reduces \

-output wasb:///Willet\_nth.amer/ensemble\_summaries\_v2\_200\_fold\_2k\_m\_2kr \

-mapper mapper.2.R \

-reducer reducer.2.R \

-cmdenv REP\_N=$REP\_N \

-cmdenv SAMPLE\_N=$SAMPLE\_N\

-file /home/hdiuser/mapper.2.R \

-file /home/hdiuser/reducer.2.R > ensemble\_summaries\_v2\_200\_fold\_2k\_m\_2k\_r.txt &

• inputs,

• outputs,

• computational strategy,

• profiling and timing,

• extensions and next steps.

Inputs:

Ouputs:

Map

and here's an example, in which we have 12 rows: 2 points, each with 6 folds (3 samples, 2 replicates):

srd-0.28-1000091 3,-101.5715,40.58235,0.28,NA,TRUE,0.01587212,0.33,4.965614,0,0.0788148

srd-0.28-1000091 3,-101.5715,40.58235,0.28,NA,TRUE,0.008532849,0.25,66.34337,0,0.5660979

srd-0.28-1000091 2,-101.5715,40.58235,0.28,NA,TRUE,0.01948633,0.3,84.75649,0,1.651593

srd-0.28-1000091 2,-101.5715,40.58235,0.28,NA,TRUE,0.006210773,0.24,16.45691,0,0.1022101

srd-0.28-1000091 1,-101.5715,40.58235,0.28,NA,TRUE,0.02289438,0.32,15.27058,0,0.3496104

srd-0.28-1000091 1,-101.5715,40.58235,0.28,NA,TRUE,0.01613391,0.3,63.75134,0,1.028558

srd-0.28-1000109 1,-101.0113,40.58235,0.28,NA,TRUE,0.02237137,0.3,108.1323,0,2.419066

srd-0.28-1000109 1,-101.0113,40.58235,0.28,NA,TRUE,0.02420048,0.32,14.25783,0,0.3450463

srd-0.28-1000109 2,-101.0113,40.58235,0.28,NA,TRUE,0.01425395,0.3,95.85014,0,1.366243

srd-0.28-1000109 3,-101.0113,40.58235,0.28,NA,TRUE,0.003086406,0.25,66.39697,0,0.204928

srd-0.28-1000109 2,-101.0113,40.58235,0.28,NA,TRUE,0.00237792,0.24,15.30061,0,0.03638361

srd-0.28-1000109 3,-101.0113,40.58235,0.28,NA,TRUE,0.008838032,0.33,6.116678,0,0.0540594

Reduce:

data.type,

orig.row.id,

lon,

lat,

date,

obs,

pi.mean,

pi.median,

pi.q.10,

pi.q.90,

tau.mean,

tau.median,

tau.q.10,

tau.q.90,

mu.mean,

mu.median,

mu.q.10,

mu.q.90,

truncated.mean,

truncated.median,

truncated.q.10,

truncated.q.90,

pi.mu.mean,

pi.mu.median,

pi.mu.q.10,

pi.mu.q.90

...and here is the summarized output of the above 2 points:

srd,1000091,-101.5715,40.58235,0.28,NA,0.0148550603333333,0.0123316979,0.0128485515,0.0181810263,0.29,0.274,0.29,0.306,41.9240506666667,36.4257856,39.51096,48.387552,0,0,0,0,0.6294807,0.39578192,0.6890842,0.83933808

srd,1000109,-101.0113,40.58235,0.28,NA,0.0125213596666667,0.0064329622,0.008315935,0.020291927,0.29,0.274,0.29,0.306,51.009088,40.1205342,55.575375,60.071127,0,0,0,0,0.737621051666667,0.243857621,0.701313305,1.245907581

Comparing head node size:

summary task--

bighead

40 nodes, A3

D3 node

100 reducers

start

15/08/28 13:46:43

finish

15/08/30 12:32:35 INFO mapreduce.Job: map 100% reduce 100%

15/08/30 15:10:32 INFO mapreduce.Job: Job job\_1440739951604\_0001 completed successfully

total: 2 days, 1 hour 20 minutes

small head

won't finish even!

start

15/08/27 21:32:30 INFOh

now hanging:

15/08/30 01:39:53 INFO mapreduce.Job: map 100% reduce 100%

**Section B3: Package output / results from MR2**

**GETMERGE STEP: Put the Reduced PARTS back together retrieve them**

This should create the merged summary file on the root directory of the head node. Note that this should be done from the small "transfer" cluster, not the large "compute" cluster.

ssh hdiuser@small-cluster-ssh.azurehdinsight.net

# From the head node:

hadoop dfs -getmerge wasb:///Willet\_nth.amer/ensemble\_summaries\_200\_fold\_2k\_m\_2kr ensemble\_merged\_summaries\_200\_fold\_2k\_m\_2kr.txt

# Then scp it back home. E.g. From local machine:

scp hdiuser@small-cluster-ssh.azurehdinsight.net:~/ensemble\_merged\_summaries\_200\_fold\_2k\_m\_2kr.txt .

**"Unpeeling" the Results on Atlas**

Hadoop dumps all predictions into one file. At our production scale (North America, 52 weeks), this text file come out to ~30 GB. This of course is too cumbersome to really use, so we want to carve up the predictions into usable pieces. We do this with an awk script, that “unpeals” the prediction dump into 55 files—52 weeks of predictions on SRD data, test data, training in bag, and training out of bag.

How big do these tend to be at our production scale?

An srd week: 450 MB

Test: 80MB

Train: 800 MB

(notice how big the training data predictions are! That's why we use our involved prepping scripts in preparing the ingredients for MR 3.)

Get on ATLAS

qsub -I -q inter -l nodes=1:ppn=12

now cd to where ensmeble summary file lives

cd ~/stem\_hwf/runs/Willet/results/hadoop\_results/

--put the awk script there

-- (important!) mkdir unpealed\_srd\_200.fold

awk -f srdSlicing.txt ensemble\_merged\_summaries\_200\_fold\_2k\_m\_2kr.txt

# Here is the AWK script:

# --------------------------

# BEGIN{ #Prior to starting work

# FS = "," # set the input field separator as a comma

# OFS = "," # and the same for the output separator.

# }

# {

# #column 5, date fraction

# if ($1 == "srd")

# outname="unpealed\_srd\_200.fold/unpealed\_200.fold\_srd.time\_"$5".txt"

# print $0 > outname

# if ($1 == "train.in.bag")

# print $0 > "unpealed\_srd\_200.fold/train.in.bag.pred.ave.txt"

# if ($1 == "train.out.bag")

# print $0 > "unpealed\_srd\_200.fold/train.out.bag.pred.ave.txt"

# if ($1 == "test")

# print $0 > "unpealed\_srd\_200.fold/test.pred.ave.txt"

# }

**Section B4: Data Prep For MR3**

Use AWK script to break training out-of-bag data into 52 month-wide data pieces.

Here, a set of awk and R scripts extract a subset of predictions to use in our occurrence predictive performance assessment.

To run, first make a directory with the species name, and move the merged ensemble level summaries (from B3) inside here.

Then, run the outer script, which will call the 4 inner scripts:

$ ./awkslice\_train.weeks\_outer.sh <species\_name>

This routine has 4 main steps

1. Extract the occurrence predictions for the training out-of-bag data.
2. 3 scripts (trainWeekSlicing\_body/head/tail) carves out 52 pieces of this data set, each one containing a month of data surrounding a week. Three scripts are needed to deal with wrapping time around the year end dates.
3. An R script prepares a matrix of rows of parameters to test for evaluation in the third MR task. Proposed thresholds for each week are the 100 quantiles of the predictions on positive conversations in that week-centered month chunk
4. Place both the prediction data and the parameter listings in your storage account.

**Section B5: MR3 Threshold selection and occurrence predictive performance assessment**

Map only step does parameters sweep to find optimal thresholds and quantify predictive performance.

In this step, we gather occurrence binary predictive performance metrics for each week over a range of proposed thresholds. We can then use these metrics to select good occurrence thresholds for each week. These metrics also provide numeric baselines for predictive performance.

The Hadoop stream input here are lines of parameters for one operation, not the data that will processed like in our first two Map Reduce task—a week index and a threshold to evaluate. The data to be processed is separated into chunks by week and placed into the DFS (azure storage in this case). The mapping task here will read in the chunk of data that corresponds to the week.id. (Note that simply passing all of the data overwhelms RAM resources on the nodes.) Then we perform a sampling routine on this chunk (centered around a week, but containing a month’s worth of data) to produce test sets that are more uniformly distribution across space. Last, we take a number of measurements on the sampled test data.

The current metrics:

PCC

Sensitivity

Specificity

Kappa

AUC

True Skill Statistic

Brier uncertainty

Raw resolution

Relative resolution

Raw reliability

Relative reliability

Confusion matrix enumeration: TP, FP, TN , FN

observed negatives

observed positives

Some numbers on production scaled runs:

***2.25 mins:*** Approximate time to process each parameter line (week id and proposed threshold). Most of this time is spent performing the spatially even sampling routine.

***130k tests*** : (52 weeks \* 100 thresholds \* 25 MC draws) (see below for expansion)

***2.6 hours***: estimated time to do all on 240, 8-core machines

For each week, we test 100 thresholds. These are the 100 quantiles of the predicted probability of occurrence values on the out of bag training points.

For each threshold, we do 25 tests, performing the sampling for equal coverage on each.

Note: CHUNK\_LOCATION is an important environmental variable stating where the week-centered month chunks of prediction data live. Be sure the syntax is correct, one way is to try to “cat out” one of these file. For instance, try:

hadoop dfs –cat $CHUNK\_LOCATION/occurence.results\_week.5.txt

Submission looks like this, but be sure to modify the species name to the current run:

export HADOOP\_STREAMING\_JAR=/usr/hdp/current/hadoop-mapreduce-client/hadoop-streaming.jar

export SAMPLE\_N=50

export REP\_N=4

export RUN\_NAME=Kelling\_warbler\_total

export CHUNK\_LOCATION=wasb:///Kelling\_warbler\_total/occurence\_week\_chunks/

nohup hadoop jar ${HADOOP\_STREAMING\_JAR} \

\

-D mapred.task.timeout=2000000 \

-D mapred.map.tasks=1900 \

-D mapred.reduce.tasks=0\

-input wasb:///Kelling\_warbler\_total/ebird.abund\_Kelling\_warbler\_total\_threshold.test.instance\_data.frame.csv \

-output wasb:///Kelling\_warbler\_total/threshold\_results\_25\_draw\_100\_thresh\_pipe.method\_long.time.out \

-mapper mapper.3\_scaled.R \

-cmdenv REP\_N=$REP\_N \

-cmdenv SAMPLE\_N=$SAMPLE\_N\

-cmdenv CHUNK\_LOCATION=$CHUNK\_LOCATION\

-cmdenv RUN\_NAME=$RUN\_NAME\

-file /home/hdiuser/mapper.3\_scaled.R \

-file /home/hdiuser/mapper.3.threshold.testing.library.R > threshold\_results\_3\_draw\_25\_thresh\_TRES.txt &

**Section B6: Package MR3 Results**

Run with:

$ mkdir unpealld\_srd\_200.fold/

$ awk –f srdSlicing.txt

Hadoop dumps all predictions into one file. At our production scale (North America, 52 weeks), this text file come out to ~30 GB. This of course is too cumbersome to really use, so we want to carve up the predictions into usable pieces. We do this with an awk script, that “unpeals” the prediction dump into 55 files—52 weeks of predictions on SRD data, test data, training in bag, and training out of bag.

How big do these tend to be at our production scale?

An srd week: 450 MB

Test: 80MB

Train: 800 MB

(notice how big the training data predictions are! That's why we use our involved prepping scripts in preparing the ingredients for MR 3.)

**Appendix: Profiling**

Comparison of run times and resource use, by step, for WF3 and HWF. Magnolia Warbler vs Kelling Warbler.

Section A on Cluster deployment and data preparation is broken into the following subsections:

A1: Azure Initializations

A2: Local Initializations: RSA key setup

A3: Prepare eBird ERD & SRD data

A4: Azure Cluster Deployment

A5: Finish RSA Key Configuration Connecting Local Machine and Azure Cluster

A6: Check R Installation on Azure Cluster

A7: Transfer Files to Cluster Head Node via SCP

A8: Transfer Data Files to Storage Account – Via Transfer Cluster

The organization of Section B follows directly from the computational workflow:

B1: MR1 Modeling Step: training and prediction of base models

B2: MR2: Summarization Step: summarize predictions across folds

B3: Package output / results from MR2

B4: Data Prep For MR3

B5: MR3 Threshold selection and occurrence predictive performance assessment

B6: Package MR3 Results

**MR1:**

ATLAS:

Resource Used: 1 core/ bigmem node (12 cores/96GB RAM) @ 12 nodes

Time: 19 hr / fold / node(resource unit) \* 200 folds = 3800 hours total @ 12 resource unit (all bigmem queue) = **316 hours or 13 days**

AZURE:

Resource Used: 1 core with <3GB RAM/core @ 8 core & 24GB RAM / D4 worker nodes

Time: **5 hours** @ 241 nodes ~ 1900 cores running concurrently (based on hadoop containers used)

NOTE: Complexities of counting core hours. On ATLAS, the 3800 core hours is what is actually doing the computing. But we tie up, i.e. “use” many more: 144 cores on bigmem \* 13 days \* 24 hr / day = 44K cores hours used! Compare this to 9K core hours used on Azure cluster. Similarly for RAM.

NOTE: I am stating the resource use for MR1 at finest granularity.

**MR2:**

ATLAS:

Resource Used: 1 core/short node (12 cores/96GB RAM) @ 12 nodes

Time: 2 hr / core / SRD week 🡺 total of 6 hrs for SRD week summaries for Magnolia Warbler job (not sure how many nodes were deployed).  **🡺 total 8 hours (for SRD, Train & Test summaries with 24 cores running concurrently)**

AZURE:

Resource Used: 1 core with <3GB RAM/core @ 8 core & 24GB RAM / D4 worker nodes

Time:  **~2.5 hours** @ 241 nodes ~ 1900 cores running concurrently (based on hadoop containers used)

**MR3:**

ATLAS: (based on Northern Pintail)

Resource used: MC on bigmem uses 12 cores / node @ 12 nodes

Time: 14 hours

AZURE:

Some numbers on production scaled runs:

*2.25 mins:* Approximate time to process each parameter line (week id and proposed threshold). Most of this time is spent performing the spatially even sampling routine.

*130k tests* : (52 weeks \* 100 thresholds \* 25 MC draws)

*2.6 hours*: ***estimated*** time to do all on 241, 8-core machines

**Total Wall Clock for Computing (Excludes data prep, movement, etc)**

Azure < 12 hrs

ATLAS ~14 days

**Work time, ie serial execution time (Clock time x nodes at work)**

Atlas:

**6k hours** = **3.8 k hours** (step 1) + **192 hours** (step 2) + **2016 hours** (step 3)

Azure:

**19k hours =** 1900 \* 10

Notice Atlas is 3x faster! Why? There are several reasons

* Think horizontal scaling! These Azure VM’s are lower quality, by intent. This means Azure machines have various lower quality components, ie less powerful processors. You can see this differences in the single thread awk scripts, which take 2-4 times longer on an Azure node than on Atlas.
* Azure is VM’s! Atlas is M’s!
* Hadoop as a framework has costs, for instance increased movement of data through network pathways across machines. Also note that Atlas has extremely fast, Infiniband connections between nodes.

**Replacement cost analysis of STEM WFv3**

Atlas Bigmem queue:

12 x (12 core, 96 GB)

144 core, 1152 GB ram (1.152 TB ram)

Price if rented on Azure for MR1 only!

~$3k - $5k

closest match of machine, the D14

16 cores 112 GB RAM 800 GB Disk

12 for 323 hours at $1.31/hour=

$5,060.12

the next cheapest might not be relevant, as likely has too little ram to run:

8 cores 56 GB RAM 400 GB Disk $0.73/hr

12 for 323 hours at $.73/hour=

$2,812.04

Note: these estimates are conservative (ie, Atlas could have been shown to be worse) for 2 reasons:

* Atlas nodes appear to have faster processors than Azure machine, as demonstrated by timing the awk scirpts
* V3 is written for Atlas or a similar HPC with a shared network mounted filesystem! So without a major rewriting and development, it just wouldn’t work on this architecture.
* While the main ideas, structure of V3 might work as a map only job on Hadoop

**Appendix: “By-Hand” Installation of R and R packages on Azure HDI Cluster**

all done on cluster in ssh session

$ssh hdiuser@<cluster\_name>-ssh.azurehdinsight.net

a. Install pssh library on head node,

b. Gather a formated list of the names of cluster machines,

c. Copy then run an installation script for R and packages on all machines

this script also sets up some important environmental variables on all machines

#-------------------------------------------------------------------------------

# OLD R INSTALLATION - Useful notes on working with the cluster

#

# 2 cluster settup

# -- from cluster --

#

#-------------------------------------------------------------------------------

NOT NEEDED NOW!!!!

## copy settup script up to head node

cd ~/repos/work\_repos/hadoop\_sandbox/ebird\_azure/abundance/cluster\_settup

scp node\_r\_install.sh hdiuser@dfcluster1-ssh.azurehdinsight.net:~/

#Get on up to the cluster head node, now well stocked with the HWF files!

ssh hdiuser@dfcluster1-ssh.azurehdinsight.net

# install pssh tools (parrallel ssh) to operate on all worker nodes

sudo apt-get install -y pssh

# Create a list of data/worker and head nodes (two methods)

# The goal of this step is to create a list of head node and worker node ip's, with no spaces and no quotes.

## 1) the easy way, but often doesn't work, in which case use the Ambari API ( option 2 )

hadoop dfsadmin -Dfs.defaultFS=hdfs://headnode0 -report | grep 'Hostname: ' | sed 's/Hostname: //' > pssh-hosts

# Add the headnode

echo 'headnode0' >> pssh-hosts

# when fails, says:

# report: Operation category READ is not supported in state standby

#########

# 2) use the Ambari api, in one of 2 ways (2a or 2b)

########

## 2a and 2b just two methods to parse this output:

curl -u admin:<cluster\_admin\_password> http://localhost:8080/api/v1/hosts?fields=Hosts/ip

## e.g.

# curl -u admin:1Northamer! http://localhost:8080/api/v1/hosts?fields=Hosts/ip

# Also note: the cluster has nodes that are neither head nodes nor worker nodes, but are hadoop workflow nodes-- oozie and zookeeper. Here, you might want to differentiate those, keep them off the pssh hosts list. You can pull them off easitly with the manual method, 2a. 2b is automated, and makes the list with no handwork, but it will include these non-worker nodes. I think 2b is better, I just now have used it, and it seems fine, but just beware. 2a is a bit more transparant.

#\*\*\*\*\*\*\*\*

# 2a) manual host file curation

$ curl -u admin:<cluster\_password>! http://localhost:8080/api/v1/hosts?fields=Hosts/ip | grep "10.0.0\*"

# then paste into text file, and clean out bad nodes, spaces, and quotes

# when ready?

# $ touch pssh-hosts

# $vim pssh-hosts

## and paste in the cleaned up ip's

# \*\*\*\*\*\*\*\*\*\*\*\*

# 2b) automagic host list collation

# is, use an inscruitable bit of sed stream cleaning

# again, this will add all nodes, zookeepers too, to the pssh-hosts list

curl -u admin:1Northamer! http://localhost:8080/api/v1/hosts?fields=Hosts/ip | grep "10.0.0\*"| sed 's/.ip. : //' | sed 's/\"//g' | sed -e 's/^[ \t]\*//' -e 's/[ \t]\*$//' > pssh-hosts

# \*\*\*\*\*\*\*\*\*\*\*\*

# which ever route was taken, inspect the file

cat pssh-hosts

# Check that parallel-ssh works and cause the host keys to be saved

parallel-ssh ${psshSwitches} -i -h pssh-hosts -x "-o StrictHostKeyChecking=no" hostname | grep SUCCESS

# Copy the R/R-packages install script to worker nodes

parallel-scp -h pssh-hosts -x "-o StrictHostKeyChecking=no" node\_r\_install.sh ~/

# DF: Test to see if I can run commands on worker nodes - success!

parallel-ssh ${psshSwitches} -i -t 0 -h pssh-hosts -x "-o StrictHostKeyChecking=no" echo "hello, world"

parallel-ssh ${psshSwitches} -i -t 0 -h pssh-hosts -x "-o StrictHostKeyChecking=no" ls -la

# DF: Manually change permissions

parallel-ssh ${psshSwitches} -i -t 0 -h pssh-hosts -x "-o StrictHostKeyChecking=no" chmod ugo=rwx ~/node\_r\_install.sh

# DF: Manually run script

parallel-ssh ${psshSwitches} -i -t 0 -h pssh-hosts -x "-o StrictHostKeyChecking=no" ./node\_r\_install.sh

# run install script on all nodes

parallel-ssh ${psshSwitches} -i -t 0 -h pssh-hosts -x "-o StrictHostKeyChecking=no" ~/node\_r\_install.sh

# or, if you need to go somewhere?? as this actual installing of R is the most time consuming part, use nohup and leave:

nohup parallel-ssh ${psshSwitches} -i -t 0 -h pssh-hosts -x "-o StrictHostKeyChecking=no" ~/node\_r\_install.sh & > install\_r\_nohup.txt

# You can test to make sure that R has been installed on various nodes.

# From the head node:

# ssh 10.0.0.7

# R

**Appendix: Adding R Packages to Cluster**

Need to add additional R packages across the cluster?

Within Action Script: ADD TEXT

By-Hand:

* edit node\_r\_install.sh and rerun on all nodes with the pssh command,
* this is the line you would change: install.packages(c('pryr','PresenceAbsence', 'verification', 'gbm', 'scam', 'splines')repos="http://cran.revolutionanalytics.com", INSTALL\_opts=c('--byte-compile') )

Using Precompiled libraries to avoid time-out error during deployment

**Appendix: Ambari Cluster Monitoring (advanced)**

Web based job metrics: Ambari

Summary: there is a web based service that Microsoft offers with many metrics collated in one place, logs too. Unfortunately, its a bit finicky, with both tricky normal signin, and then a recquirement to set up an ssh tunnel. This could be useful to get familiar with, especially if a heavy user, but its also cumbersome.

This here is a main reference page I oft come back to:

https://azure.microsoft.com/en-us/documentation/articles/hdinsight-hadoop-manage-ambari/

* + explains Ambari, the monitoring UI, sign in. Note the unexpected usernames of "admin",
  + despite in all other cases this is "hdiuser" or something else specified at cluster deployment

https://azure.microsoft.com/en-us/documentation/articles/hdinsight-hadoop-linux-use-ssh-unix/

* Explanation of ssh-keys and tunneling on azure

Clusters can be monitored with a web UI via the Ambari API. This service gathers many different metrics on jobs, and reports warning.

Logs are quite hard to find on Azure-- it turns out many can only be accessed from this service. That said, if you want to try and figure out system issues happening and investigate Yarn sytem logs, you must pass through 3 layers of security! Its also quite finicky, where I find myself unable to make any password- user name combinations work. so I view this more as a luxury when it works

Go to:

https://<cluster\_name>.azurehdinsight.net

you will be askwed to authenticate once in a browser diaglogue box, and if succesful, then again from within the service

Use admin as user name for both, and also use the password that you supplied when deploying the cluster.

From here, you can navigate to many things. Some features, however, and arguably the best feaures, require setting up an SSH-tunnel. This includes reading failure logs and system logs on problem jobs, or job development. When navigating through the GUI, you'll hit a dead end, reporting "unable to load page." That likely means you have not set up the recquired tunnel!

Settup recquires three things—

1. add the forwardAgent line to the entry for the cluster in the .ssh file:

Host <cluster\_name>-ssh.azurehdinsight.net

User hdiuser

IdentityFile ~/.ssh/id\_rsa\_clusters

ForwardAgent yes

1. run this here command:

$ ssh -C2qTnNf -D 9876 hdiuser@ntamer3full-ssh.azurehdinsight.net

which means: sets up a tunnel, running in the background, through your machine's port 9876, out to the azure node

1. configure your browser to use the back channel port in certain settings, see this link for setting up Foxyproxy in chrome

https://azure.microsoft.com/en-us/documentation/articles/hdinsight-hadoop-manage-ambari/

**Appendix: Azure Storage Account Theory**

Useful Links

* <https://myaccount.blob.core.windows.net/mycontainer/myblob>
* Azure help pages on "Naming and Referencing Containers, Blobs, and Metadata"
* <https://msdn.microsoft.com/en-us/library/azure/dd135715.aspx?f=255&MSPPError=-2147217396>

Streaming input, HDFS vs. Azure Storage

We are putting in the place the main stream of data that will be fed through the HWF. On traditional Hadoop systems, this would data would placed from a cluster node onto the HDFS. When the cluster comes down, the information on the HDFS vanishes. Hadoop results are written onto the HDFS, so they must be pulled off and put in a more durable place.

Azure uses a special Hadoop setup, where its own tool, the Azure storage account, or BLOB storage, acts as your HDFS. You put things on your storage account in a similar way you would an STFTP service. Multiple clusters can simultaneously point to this account as their HDFS, and results persist without further action. The downside? Its a bit cumbersome to get data up into the storage accounts, and by saving all results, a developer needs to be careful about accumulating junk during experiments and mindful of storage costs.

On storage accounts and containers:

The top directory layer of storage account is called a container. Each HDI cluster has a default container, that you specify (and is created?) at cluster deployment. All pathnames on that cluster begin in the container-- see the below example.

In other words, you open up a Cyberduck connection to a storage account, in which you'll see several containers listed. For me, my storage account is "ebirddata", and when I open CyberDuck, I see my containers, in which I have many directories:

* mycontainer1
* testrinstall
* testnafilter

An HDI cluster will only point, by default, at one of these containers, and one storage account. Path names declared on an HDI node, therefore, ignore both storage account and containter name:

* wasb:///abund-profile-extent/

No mention of "ebirddata" or "testrinstall". That said, you can give directions in a cluster to point to other containers other than its default, even other storage accounts! But I haven't looked into that-- has not been necessary for me yet...

Note that URI’s can be obtained from CyerDuck by control + clicking on the script in the directory viewer, and selecting "get url -> "https URL"

From the azure help pages: <https://myaccount.blob.core.windows.net/mycontainer/myblob> So if my storage account is called “sd2” the container on sd2 is “scripts” and I uploaded “ebird-action-script.sh” to scripts. Azure help pages on "Naming and Referencing Containers, Blobs, and Metadata" <https://msdn.microsoft.com/en-us/library/azure/dd135715.aspx?f=255&MSPPError=-2147217396> You can also get the path name from CyerDuck by control + clicking on the script in the directeroy viewer, and selecting "get url -> "https URL"

**Appendix: File transfer: FTP Client Cyberduck**

Versions 4.7 and above have support built in for Azure storage accounts

A) Add a new connection in Cyber duck

1. in the drop down menu of types of connection, towards the bottom, select Azure Storage Account
2. insert nickname and username and then the username, which is the name of the blob your cluster is pointing at
3. put that same username, then, into this part:
   * 1. <storageaccount>.blob.core.windows.net
4. x out, and the name will seem redunant. For me, the url reads:
   * 1. # https://ebirddata@ebirddata.blob.core.windows.net
5. Click on that connection now, and you'll have to specify Primary Access Key. Get this from the Web UI.

B) Create “Subdirectory” in a Container on Azure

Open Duck connection:

In your HDI cluster's default container, make a directory for this run. We are using “runs” as the default directory for all clusters so that they point to the same place. Then we add the “run name”, often species name, possibly with tags for spatial and temporal designs.

E.g. I created /Willet\_nth.amer/ under the /runs/ container. (if the term "containers" in your Azure Storage Account is unknown or unclear, See **Appendix: Azure Storage Account Theory)**

C) Now transfer over the file

I've proceed in 2 ways, both are quite unsatisfying and kludgey abuses of CyberDuck, but they work.

(If this is unsatisfying or breaking, see the appendix on the Azure Command line interface tools.)

1. Open a Cyberduck window to the Atlas location where the prepped HWF product lives. Drag it from the ATLAS Cyberduck window over and into to the Azure Cyberduck window, into the directory you just made. I am not kidding. Yes, this has worked for me.
2. More stable: use your machine as an intermediate step: download the file to you're machine first, then drag from you're machine up to the Azure Duck window.

**Appendix: File transfer: Azure Command-line Interface (ACI)**

* This is the AK link - overview of commands
* <https://azure.microsoft.com/en-us/documentation/articles/virtual-machines-command-line-tools/#commands-to-manage-your-storage-objects>
* How to install on LINUX?
* <https://azure.microsoft.com/en-us/documentation/articles/xplat-cli-install/>

Azure Command-Line Interface

This article describes how to install the Azure Command-Line Interface (Azure CLI). The Azure CLI provides a set of open-source shell-based commands for creating and managing resources in Microsoft Azure.

Here is the web page with info on commands:

https://azure.microsoft.com/en-us/documentation/articles/virtual-machines-command-line-tools/#commands-to-manage-your-storage-objects

# Login with CU credentials

azure login -u df36@cornell.edu

# This command displays the storage accounts (BLOBS) on your subscription.

azure storage account list

# Storage Management Command Overview

azure help storage

When creating the storage account there is a choice

Select a Deployment model: The choices are “classic” and “resource manager'

Azure seems to suggest using the resource manager.

Note - that the CLI needs to be set in the correct mode to match the storage account. The following command will show you what mode the CLI is in:

azure config

This command changes to the "Azure Resource manager"

azure config mode arm

The old "classic" storage account needs "asm (Azure Service Management)"

azure config mode asm

# This command displays the storage container list for a specified storage account. I have specified the storage account using the storage account name and account key.

azure storage container list \

-a sd2 \

-k <KEY>

info: Executing command storage container list

+ Getting storage containers

data: Name Public-Access Last-Modified

data: ---------- ------------- -----------------------------

data: dfcluster1 Off Mon, 21 Sep 2015 17:37:00 GMT

data: ttt Off Wed, 07 Oct 2015 16:49:15 GMT

info: storage container list command OK

# This creates a container called "runs"

azure storage container create \

-a sd2 \

-k <KEY> \

runs

# This creates a container called "scripts"

azure storage container create \

-a sd2 \

-k <KEY> \

scripts

# I have not been able to figure out how to create subdirectories

# using the "/" notation ????

# This creates a blob/container virtual subdirectory called "runs/Western\_Tanager"

azure storage container create \

-a sd2 \

-k <KEY> \

runs/Western\_Tanager

# This uploads the file into the "runs" container

# NOTE: I was not able to upload the file into "runs/Western\_Tanager/"

azure storage blob upload \

-a sd2 \

-k <KEY> \

--container runs \

ebird.abund\_Western\_Tanager\_nth.amer\_all.random.merge.txt

# This uploads the file into the "ttt2" container

azure storage blob upload \

-a stemdata \

-k <KEY> \

--container ttt2 \

mean.0.25.1..004.January\_4.png

# This downloads file "debug.size\_ebird.abund\_Long-billed\_Curlew\_erd.test.data.csv" from the "ttt" container

azure storage blob download \

-a stemdata \

-k <KEY> \

--container ttt \

debug.size\_ebird.abund\_Long-billed\_Curlew\_erd.test.data.csv

**Appendix: AZURE Copying files across storage accounts**

10.26.2015

NEB is in DC, copying files across storage accounts

from:-- his ad.hoc, MS account West account

westattempts@ebirdweststore.blob.core.windows.net

to: -- the shared cornell account

runs@sd2.blob.core.windows.net

How?

First, deploy a cluster

from Ad.Hoc account portal

in additional options, you can supply an additional storage account with access key

(use this to get access to the cornell shared account!)

Then, ssh onto that cluster and use the hadoop distcp command to copy files

Note!!

-even if you have the cluster in the correct region, be sure you have the right region "resource group!"

I first made this mistake on a cluster, with resource group in South Central US, cluster in the West, and the transfer was very slow

Note 2!!

-copying the whole species subdirectory from the ad.hoc account seemed to have issues

collission issues that made the whole job fail

-so, you'll see that i just copy the needed subdirectory-- the MR2 results.

scp ~/.ssh/id\_rsa\_clusters/ hdiuser@transstor2-ssh.azurehdinsight.net:~/.ssh/id\_rsa

scp ~/.ssh/id\_rsa\_clusters/ hdiuser@transstor3-ssh.azurehdinsight.net:~/.ssh/id\_rsa

#-------------------------------------------------------------------------------

# pelican

#-------------------------------------------------------------------------------

nohup hadoop distcp wasb://westattempts@ebirdweststore.blob.core.windows.net/American\_White\_Pelican\_nth.amer/200\_fold\_241\_nodes\_1.9k\_maps\_1.9k\_reduces wasbs://runs@sd2.blob.core.windows.net/American\_White\_Pelican/ > copy.pelican\_on.better\_cluster.txt &

#-------------------------------------------------------------------------------

# Western tanager

#-------------------------------------------------------------------------------

hadoop distcp wasb://westattempts@ebirdweststore.blob.core.windows.net/Western\_Tanager\_nth.amer/200\_fold\_241\_nodes\_1.9k\_maps\_1.9k\_reduces wasbs://runs@sd2.blob.core.windows.net/Western\_Tanager/

#-------------------------------------------------------------------------------

# yellow-rumped warbler

#-------------------------------------------------------------------------------

nohup hadoop distcp wasb://westattempts@ebirdweststore.blob.core.windows.net/Yellow-rumped\_Warbler/200\_fold\_241\_nodes\_1.9k\_maps\_1.9k\_reduces wasbs://runs@sd2.blob.core.windows.net/Yellow-rumped\_Warbler/ > copy.across.accounts\_Yellow.rumped.warbler.txt &

#-------------------------------------------------------------------------------

# Blue-winged\_Teal

#-------------------------------------------------------------------------------

nohup hadoop distcp wasb://westattempts@ebirdweststore.blob.core.windows.net/Blue-winged\_Teal/200\_fold\_241\_nodes\_1.9k\_maps\_1.9k\_reduces wasbs://runs@sd2.blob.core.windows.net/Blue-winged\_Teal/ > copy.across.accounts\_Blue-winged.Teal.txt &

### 3 research grade species, where the intermediate files, MR1 results, are desired for research needs.

Succesfully getting everything over means NEB can blitz his ad hoc account...

or at least clear it out so its lightweight, it does have cores that can be mobilized for an emergency production run.

- wood thrush

- blackpole

- tree swallow

#-------------------------------------------------------------------------------

# blackpole

#-------------------------------------------------------------------------------

nohup hadoop distcp wasb://westattempts@ebirdweststore.blob.core.windows.net/blackpoll-nth-amer/200\_fold\_241\_nodes\_1.9k\_maps\_1.9k\_reduces wasbs://runs@sd2.blob.core.windows.net/Blackpoll\_Warbler/ > blackpoll.copy.over.txt &

#-------------------------------------------------------------------------------

# wood thrush

#-------------------------------------------------------------------------------

nohup hadoop distcp wasb://westattempts@ebirdweststore.blob.core.windows.net/wood-thrush-nth-amer/200\_fold\_241\_nodes\_1.9k\_maps\_1.9k\_reduces wasbs://runs@sd2.blob.core.windows.net/Wood\_Thrush/ > wood.thrush\_copy.over.txt &

#-------------------------------------------------------------------------------

# TRES

## do this one last, it has some threshold information too

#-------------------------------------------------------------------------------

tres-nth-amer-abund

## MR! results

nohup hadoop distcp wasb://westattempts@ebirdweststore.blob.core.windows.net/tres-nth-amer-abund/200\_fold\_150\_nodes\_2k\_maps\_2k\_reduces wasbs://runs@sd2.blob.core.windows.net/TRES/ > TRES.copy.over.mr1.redults.txt &

##threshold results too!

nohup hadoop distcp wasb://westattempts@ebirdweststore.blob.core.windows.net/tres-nth-amer-abund/threshold\_results\_3\_draw\_25\_thresh\_pipe.method\_long.time.out wasbs://runs@sd2.blob.core.windows.net/TRES/ > TRES\_copy.over.threshold.results.txt &