Insilicos
Ensemble :: Cloud Army

Framework for Ensemble Machine Learning using Amazon Web Services

CONTENTS

1 What is Ensemble :: Cloud Army?

Ensemble :: Cloud Army (ECA), developed by Insilicos LLC, is a framework for performing ensemble machine learning calculations written in the R statistical programming language. ECA uses the Amazon Web Services (AWS) Elastic Compute Cloud (EC2) or Elastic Map Reduce (EMR) services to create a temporary compute cluster in the cloud. The ECA framework allows you to write R code to run in parallel on this cluster. You can also run the same R code in a non-parallel fashion on a local computer for prototyping purposes.

ECA supports two different modes of parallelization (the scripts you write don’t have to worry about this, though).

MPI: ECA uses the R package RMPI to support parallel execution and intracluster communication via the message passing interface (MPI) protocol.

HADOOP: You can also use Hadoop Map Reduce via the AWS Elastic Map Reduce (EMR) service.

Which one should you use? We find that Hadoop is the better choice when you want to run more low-cost calculation nodes – latency issues in EC2 make MPI somewhat unreliable except with the relatively expensive Cluster Compute instance types, and EMR is much more robust at higher node counts.

ECA is designed to hide most of the complexity of dealing with EC2 and EMR. Certain parts of the ECA framework could also be helpful for setting up MPI clusters for other purposes (for example, the X!!Tandem peptide search engine implementation from Insilicos uses ECA to launch its MPI ring on EC2).

MPI handling in this version of ECA has been greatly improved by using AMIs provided by the StarCluster project at MIT (http://web.mit.edu/stardev/cluster/). You don't need a local installation of the StarCluster cluster management system since ECA manages the cluster for you.

2 Getting started

This manual assumes you have already downloaded and installed ECA according to the instructions in the accompanying Quick Start Guide. It is highly recommended you also work through the three tests in the Quick Start Guide. This will confirm that installation went properly, and also familiarize you with the usual scenarios for setting up and running jobs, and retrieving results.

2.1 ECA launches from your local computer

Although ECA relies on a cloud cluster to perform its intensive parallel computations, the system as a whole is launched from and controlled by a Python script running on your local computer. This requires your local computer to have Python 2.6 or greater installed, along with Python modules `boto`, `simplejson`, and `paramiko`.

Windows users also need access to a couple of Linux-style commands like `cat` and `gzip`; these are available in the package `UnxUtils`.

There is a windows installer program that handles these issues for you. Step-by-step instructions for all these installs are found in the Quick Start Guide.

If you plan to prototype your R scripts on your local computer before running them on the cloud cluster (highly recommended), you need to install R locally as well.

2.2 ECA uses Amazon Web Services EC2 or EMR to run a cluster

Once you have an Amazon Web Services account, sign up for EC2 and S3 (Simple Storage Service). Also sign up for EMR (Elastic Map Reduce) if you wish to use the Hadoop version of ECA.

Visit http://aws.amazon.com/, and see the Quick Start Guide for more details.

2.3 What Version of R is Used?

The EC2/MPI version of ECA provides R 2.12 as of this writing.

As of this writing AWS EMR provides the rather ancient R version 2.7, and it is possible that your ensemble R script will use packages that need a more recent version (there seems to be an evolutionary jump at R 2.10). In this case add the line
`"update_EMR_R_install":”True”`
to your ECA general configuration file and your EMR nodes will update to the latest R on boot (this adds a minute or two to the run time, though).

ECA will scan your ensemble R script for `library()` calls and make sure any needed packages are installed on the cluster.

3 Using the ECA framework

3.1 Overall flow of control

You start an ECA job by running the Python launch script (`eca_launch_rmpi.py` for MPI or `eca_launch_mapreduce.py` for Hadoop) from the command line in a console on your local computer. This script has two required arguments.

The first argument is the name of a general configuration file, which provides information necessary to create an EC2 cluster (e.g. AWS credentials, S3 bucket to use).

The second argument is the name of a job configuration file, which gives details on the specific ensemble calculation to be run (e.g. input data files, size of ensemble, name of your R script). Optionally this file can instead contain a list of job configuration files, for running multiple jobs in a single cluster instance. See Section 3.3.3 for details.

Additional optional arguments may follow the required arguments (see Section 3.3).

The launch script first requests EC2 resources from AWS to create a cluster. The cluster is comprised of one head node and some number of client nodes. Once the launch script has confirmed the successful creation of an EC2 cluster, it transfers the locally stored input data files and R script to the designated S3 bucket, and passes control to the cluster’s head node. The head node enters a processing cycle; during each pass of the cycle it 1) directs the client nodes to run instances of the ensemble base classifier, and 2) stores predicted classifications returned by the client nodes. The client node operations occur in parallel, and the head node repeats the cycle until the desired number of base classifiers have been run. The head node then computes the ensemble predictions from those of the individual base classifiers, and returns this result and control to the launch script. In the final step, the launch script shuts down the cluster and releases the EC2 resources back to AWS.

3.2 Writing an ensemble R script

The R code that runs on the ECA cloud cluster is actually three separate R scripts. The framework script (`eca_rmpi_framework.R` or `eca_mapreduce_framework.R`) and its support script `eca_common_framework.R` work with the rest of the ECA machinery to establish a parallel computing environment suitable for generic ensemble calculations. Normally you will not have any reason to modify these scripts. The third script (hereafter referred to as the *ensemble R script*) is written by you, the user, following the template and conventions described in the remainder of this section. Your script will work equally well with either the RMPI or MapReduce framework.

ECA will scan your ensemble R script for `library()` calls and make sure any needed packages are installed on the cluster.

3.2.1 Ensemble R script: structure and flow

An ensemble R script consists of eight functions with predefined names, which are executed in a defined order by the framework script. The following lists the names of the functions and their order of execution. Note that execution often occurs in parallel.

1. `common_preamble` runs in parallel on both the head node and all client nodes.
2. `head_preamble` runs on the head node while `client_preamble` runs (in parallel) on all clients.
3. For n = 1 to number of head node processing cycles (see Section 3.1):
   All clients run `client_ensemblecalc`.
4. The head node runs `head_resulthandler` to process the return values from all runs of `client_ensemblecalc`.
5. `head_postscript` runs on the head node while `client_postscript` runs (in parallel) on all clients.
6. `common_postscript` runs in parallel on both the head node and all client nodes.

It is your responsibility to provide the contents of the eight functions in an ensemble R script. A simple "`hello world`" example, `eca_script_template.R`, is provided as a starting point. A completely functional example of an ensemble R script, `eca_decision_forest.R`, is also provided.

3.2.2 ECA helper functions

Several functions are available in `eca_common_framework.R` to call from your ensemble R scripts.

3.2.2.1 `eca_read_datafile( filename, rowlimit = -1 )`

Examines the named file to determine its format and reads it into a dataframe (or `rowlimit` lines of it, if `rowlimit` is specified and > 0). Currently this only handles files that can be read by R's `read.table` function. However, it will automatically identify the presence or absence of a header line and determine the value separator. The named file can also be gzipped. Do not use R's `read.table` function directly in an ensemble R script, as this generally causes problems.

3.2.2.2 `eca_log( text1, text2="", text3="", timeStamp=TRUE )`

Writes up to three text strings to a single logfile entry, with optional timestamp. When running in verbose mode (see `eca_verbose()`), client `eca_log()` entries will also appear in the head node's log.

3.2.2.3 `eca_debug( text1, text2="", text3="", timeStamp=TRUE )`
Like eca_log, but only produces a log message when running in verbose mode.

3.2.2.4 `eca_verbose()`
Returns `True` when ECA's `verbose` internal flag is set to `True`. See Section 3.3.2.2 for information on how to set this flag and its effects.

3.2.2.5 `eca_runLocal()`
Returns `True` when ECA's `runLocal` internal flag is set to `True`. See Sections 3.3.1 and 3.3.2.2 for information on how to set this flag and its effects.

3.2.2.6 `eca_asPositiveInt( value, rangeMax )`
Returns `value` interpreted as a positive integer, limited to the value `rangeMax`. If value is the string "all", returns rangeMax.

3.3 Launching an ECA job
An ECA job is launched from the command line of your local PC. Navigate to the directory where the launch script resides; this directory should also contain your general configuration and job configuration files. Then enter the following on the command line:

```
<eca_launch_script> <my_general_config>.json <my_job_config>.json
[ optional arguments ].
```

where `<eca_launch_script>` is either `eca_launch_rmpi.py` or `eca_launch_mapreduce.py`.

The "`<my_job_config>.json`" argument can be replaced by a file which instead contains a list of file names for such job configuration files, each of which will be executed on the cluster. This is much more cost effective than multiple invocations of the ECA launch script as it uses more of the one hour granularity of AWS billing. See section 3.3.3 for details.

At present, the only optional argument supported is `--local` (see Section 3.3.1), although you can also supply a JSON-formatted string to override any settings in the configuration files for the job.

3.3.1 Running an ECA script locally
When the ECA internal flag `runLocal` is set to `True`, ECA actually runs on your local machine instead of going out to a cluster. This means a single R process does all the work, and the head and client logic share the same R environment and global variables (be careful). The internal flag `runLocal` can be set to `True` with command line switch `--local` at launch. The value of `runLocal` can be determined at any time by calling `eca_runLocal()` from your ensemble R script.

3.3.2 ECA configuration files
ECA configuration files are JSON-formatted, which means they are just a list of *name*:*value* pairs separated by commas, with each *name* or *value* enclosed in quotes. The entire list is bracketed by curly braces. ECA configuration files can also contain

comments, which isn't standard JSON, but isn't unusual either. Here is a trivial example:

```
{
 # this is a comment.
 "Name":"Bob", # comments can follow "name":"value" pairs
 "ShoeSize":"12"
}
```

3.3.2.1 *ECA general configuration file*

This is where you keep information that's needed for every job, but generally won't change from job to job, for example, your AWS credentials. See `general_config_template.json` for a ready-to-use template for creating your own general configuration file. The following is the minimum set of parameters required to be in your general configuration file.

- `aws_access_key_id`: The access key ID you set up when you created your EC2 account.

- `aws_secret_access_key`: The secret access key you set up when you created your EC2 account. Don't share this with anyone!

- `RSAKeyName`: The name of the RSA keypair you set up when you created your EC2 account.

- `RSAKeyFileName`: The (relative) path to the local copy of your actual keypair .pem file containing the key.

- `s3bucketID`: The name of an S3 bucket for storing data files, job results, etc. If the bucket does not exist, it will be created. Remember that S3 bucket names must by unique – if you try to use a bucket name that somebody else on S3 is already using, you'll have to choose a different one.

3.3.2.2 *ECA job configuration file*

This is where you keep information that varies from job to job: data file names, ensemble R script name, etc. For working examples of job configuration files, see `decision_tree_satimage_job_configuration.json` and `decision_tree_jones_job_configuration.json`.

Parameters in the job configuration file fall into three groups, listed below. Pathnames in the parameters can be relative to the location of the job configuration file; they must use forward slash separators ("/"), not backslash ("\") (this is a quirk of `boto`).

3.3.2.2.1 Important parameters you normally need to specify

- `scriptFileName`: The path to the ensemble R script you have written to implement your ensemble calculation (contains `client_ensemblecalc`, `head_resulthandler`, etc.)

- `ensembleSize`: Determines the number of times to call the `client_ensemblecalc` function in your ensemble R script.

- `numberOfNodes`: Number of nodes you want to spin up in the cloud.

- `ec2_instance_type`: Specifies the instance type of your MPI nodes or EMR client nodes. Recognized values include `t1.micro`, `m1.small`, `m1.large`, `m1.xlarge`, `m2.large`, `m2.2xlarge`, `m2.4xlarge`, `c1.medium`, `c1.xlarge, cc1.4xlarge and cg1.4xlarge`. If not specified, defaults to `m1.large`. For EMR this sets the Hadoop client node type, master node defaults to `m1.small.` You can override the head and client instance types using the `ec2_head_instance_type` and `ec2_client_instance_type` parameters. See http://aws.amazon.com/ec2/instance-types/ for more information on EC2 instance types.

- `sharedFile_GZ_*`: This isn't a single parameter, but a style of parameter naming. Any configuration parameter that starts with `sharedFile_GZ_` (for example, `sharedFile_GZ_testdata)` causes the associated file to undergo special handling. The steps in handling the file are:
  - Locate file on local disk.
  - If file hasn't already got a `.gz` extension, make a gzipped copy.
  - Check the S3 bucket specified in `s3bucketID` to see if the gzipped file already exists in `s3://<s3BucketID>/<copy_of_local_path_to_file>/<filename>`.
  - Upload the gzipped file to the S3 bucket if it is absent or changed.

  The training and test datasets used by your ensemble R script should be specified with this style of parameter. Once they are in your S3 bucket, the ECA cloud nodes will automatically unzip them for use within the ECA cluster.

  There should never be a need for this, but you can prevent the gzipping of uploaded data files by using the alternate parameter prefix `sharedFile_*`.

3.3.2.2.2 Optional parameters that control operation of your ensemble R script

See the example job configuration files mentioned above. For instance, the parameter `numberFeaturesInSubset` controls the size of the feature subset randomly chosen for training each base classifier, and `numberInstancesInSubset` controls the size of the instance subset randomly chosen for training each base classifier. There are no restrictions on the names or values of parameters in this group, provided they make sense in the context of your ensemble R script.

Inside your ensemble R script, the value of a parameter set in the job configuration file can be accessed as a named component of the R variable `eca_config`, e.g.

```
x <- eca_config$<parameter_name>.
```

3.3.2.2.3 Other optional parameters

The following parameters are important, but their default values are usually satisfactory, and specifying them in the job configuration file is optional.

- `verbose`: Set to `True` to get more voluminous logging. Logs from the clients will be included in the head node log. In your ensemble R script you can access the value of this flag by calling `eca_verbose()`. Default: `False`.

- `baseName`: This helps determine the name used for your job directory on S3. The directory is named as a combination of `baseName` and a timestamp, e.g. `s3://`*`<bucketname>`*`/`*`<baseName>`*`/`*`<timestamp>`*`/`. If you don't specify `baseName`, ECA uses the name of the job configuration file instead.

- `resultsFilename`: ECA saves results in a file, `rpmi_computation.log`, in the S3 directory described just above under the `baseName` parameter. It also saves results to a local file named *`<basename>`*`_`*`<timestamp>`*`_results.txt`, unless you specify a different name with this parameter. If you specify an empty string (`"resultsFilename":""`) then no local file is written.

- `numberOfRTasksPerNode`: Unless specified, ECA runs one R task per core per client (so 1 for `m1.small`, 2 for `m1.large`, etc.). You may wish to increase this, depending on the memory requirements of your job.

- `RMPI_FrameworkScriptFileName`: Specifies the R script that works with RMPI and calls the 8 functions you supplied in your ensemble R script. Defaults to `eca_rmpi_framework.R`, but you can change this if, for example, you have a modified copy of the ECA framework you want to use.

- `frameworkSupportScript`: Specifies the R script that contains functions common to the RMPI and MapReduce ECA frameworks. Defaults to `eca_common_framework.R`, but you can change this if, for example, you have a modified copy of the ECA framework you want to use.

- `keepHead`: If set to `True`, the head node of the cluster is not shut down after the calculation completes. Can be useful for debugging. Default: `False`.

- `keepClients`: If set to `True`, the client nodes of the cluster are not shut down after the calculation completes. Can be useful for debugging. Default: `False`.

- `ami_32bit_id`: The ID of the AMI to be used in MPI runs for 32-bit instance types `m1.small` and `c1.medium`. If not specified, defaults to a publically available 32-bit StarCluster AMI in region us-east-1. For EMR runs this has no effect.

- `ami_64bit_id`: The ID of the AMI to be used in MPI runs for 64-bit instance types. If not specified, defaults to a publically available 64-bit StarCluster AMI in region us-east-1. For EMR runs this has no effect.

- `ami_cci_id`: The ID of the AMI to be used in MPI runs for Cluster Compute instance types. If not specified, defaults to a publically available StarCluster AMI in region us-east-1. For EMR runs this has no effect.

- `aws_placement`: It's usually best to leave this unspecified so that ECA can choose the least busy availability zone placement for you. If you do wish to specify, acceptable values are single lower case letters – for example, if your AMI is in region us-east-1 and you set this parameter to b, your job would run in us-east-1b.

- `dataDir`: Specifies where data files are deposited on the client nodes. Default: `/mnt/`.

- `ec2_endpoint`: Used to specify exotic EC2 endpoints, such as AWS private beta programs. If you specify this parameter, you also need to specify `aws_region`.

- `aws_region`: Used only in concert with `ec2_endpoint`, and ignored otherwise.

3.3.3 Spot Bids

The `spotBid` parameter allows you to bid for idle AWS compute resources that would otherwise go unused. This is less expensive, but there's a possibility of being interrupted before you're done (you aren't charged for partial hours when that happens, though). You can express your bid as a dollar amount per node per hour (e.g. "0.52") or as a percentage of the "demand instance" price (e.g. "75%%" - note the doubled % sign is needed for some systems command line processing). You generally pay less than you bid, so going high doesn't hurt. A job with a `spotBid` may take longer to start, since it has to wait for AWS to decide whether to accept the bid or not.

See http://aws.amazon.com/ec2/spot-instances/ for full information on how spot instances work.

3.4 Launching multiple ECA jobs in a single cluster instance

EC2 charges by the hour, so a cluster instance that runs for 15 minutes costs the same as one that runs for 60 minutes. Thus 3 jobs that take 15 minutes each would complete in under an hour, but your cost would be for 3 hours of cluster time if you ran them in separate cluster instances. Happily, it's easy to run them all in a single instance.

For example, you could replace the creation of these three cluster instances:

```
eca_launch_mapreduce.py my_general_config.json job1_config.json
eca_launch_mapreduce.py my_general_config.json job2_config.json
eca_launch_mapreduce.py my_general_config.json job3_config.json
```

with this single cluster instance that runs all three jobs before exit:

```
eca_launch_mapreduce.py my_general_config.json joblist.txt
```

where joblist.txt is a simple text file containing these lines:

```
job1_config.json
job2_config.json
job3_config.json
```

which will cost 1/3 as much.

4 Example Scripts Provided By ECA

4.1 eca_script_template.R

This is a very basic example of an ECA script for you to use as a jumping off point for your own scripts.

4.2 eca_decision_forest.R

Runs an ensemble of decision trees. Requires a training set.

4.3 eca_decision_forest_folds.R

Runs an ensemble of decision trees with k-fold cross-validation.

4.4 eca_neural_network_ensemble.R

Runs an ensemble of neural networks.

4.5 bangbang.R

An example of using the ECA framework to run general MPI programs on AWS - in this case, the X!!Tandem peptide search engine.