README

May 17, 2016

1 Example: Submitting R jobs on Exacloud via HTCondor

This example walks through how to submit an R script on the Exacloud via the HTCondor job scheduler.

The script, `parallelGLMTuning.R`, estimates logistic regression models on 100 simulated data sets with 1,000,000 (1e6) observations and 5 dependent variables. The runs are parallelized using various number of clusters/nodes/workers: 12, 8, 6, 5, 4, 3, 2, 1. The goal of the simulation is to identify an optimal number of clusters based on elapsed time.

The illustrated workflow is here.

The HTCondor submit script is the file `submit`. You'll notice that the submit script calls the `Rscript` executable and passes the .R script file `parallelGLMTuning.R` to the executable as an argument:

```
Executable        = /usr/bin/Rscript
Arguments         = parallelGLMTuning.R
```

Error, log, and output files are saved in the `pwd` (present working directory) with the following file names:

```
Log               = job-$(Cluster)-$(Process).log
Output            = job-$(Cluster)-$(Process).output
Error             = job-$(Cluster)-$(Process).error
```

`$(Cluster)` and `$(Process)` are system variables identifying the job. As defined, the files will not be overwritten if the submit script is resubmitted.

It is highly recommended to request resources (CPUs, memory, and disk storage) for your job. If you request too little, then your job will take longer to execute. If you request too much, then your job will sit in the queue waiting for the requested resources to be available. How can you estimate resource use? **Testing!** To test the `parallelGLMTuning.R` script, I altered lines 10-11 to run 100 models on data sets with 1e4 observations, instead of 1e6.

```
J <- 100  # This is the number of models to fit
N <- 1e4  # This is the size of the dataset
```

I captured the resource use from the `job-$(Cluster)-$(Process).log` file and extrapolated.

In this example, I'm requesting 12 CPUs, 240 GB of memory, and 10 GB of scratch storage.

```
Request_Cpus      = 12
Request_Memory    = 240 GB
request_disk      = 10 GB
```

`submit` is executed from the Exacloud bash shell prompt using

```
$ condor_submit submit
```

Details on `condor_submit` are here. There may be clues on how to write HTCondor submit scripts there.

Job status is checked using `condor_q`

```
$ condor_q | more
$ condor_q -submitter <username>
```

The parallelGLMTuning.R produced 3 outputs

1. parallelGLMTuning.txt — sink() output of script assertions
2. parallelGLMTuning.csv — summary of simulation runs as a data set
3. parallelGLMTuning.png — plot of elapsed time versus number of workers

1.1 What about R Markdown?

At the moment, Pandoc is not installed on the Exacloud. So dynamic documentation via R Markdown + knitr is not possible. Alternative workflows are below. The R Markdown steps (in orange) would be executed on a computer with Pandoc installed (e.g., your local computer or `chse.ohsu.edu`), while the big data steps (in blue) would be executed on Exacloud.

- Preprocess big data
- Big data sandwich

1.2 R code to make workflow diagrams

```
In [1]: library(DiagrammeR)
        nodeLabs <- c("HTCondor submit script",
                      "R script (.R)",
                      "HTCondor error",
                      "HTCondor log",
                      "HTCondor output",
                      "sink() output (.txt)",
                      "plot() or ggplot() output (.png, .jpg)",
                      "data objects (.RData, .csv)")
        nodes <- create_nodes(nodes=letters[1:length(nodeLabs)],
                              label = nodeLabs,
                              style = "filled",
                              fontcolor = "white",
                              color = rgb(1, 67, 134, maxColorValue=255),
                              shape = c(rep("oval", 2), rep("box", 6)))
        edges <- create_edges(from = c("a", "a", "a", "a", "b", "b", "b"),
                              to   = c("b", "c", "d", "e", "f", "g", "h"),
                              color = "black")
        G <- create_graph(nodes_df = nodes,
                          edges_df = edges,
                          node_attrs = "fontname=Lato",
                          graph_attrs = c("layout=dot", "rankdir=LR"))
        cat(render_graph(G, output="SVG"), file="workflowIllustrated.svg")

Loading required namespace: V8

In [2]: nodeLabsPandoc <- c("R Markdown script (.Rmd)",
                            "Rendered document (.html, .md)")
        i <- length(nodeLabs)
        nodesPandoc <- create_nodes(nodes=letters[(i + 1):(i + length(nodeLabsPandoc))],
                                    label = nodeLabsPandoc,
                                    style = "filled",
                                    fontcolor = "white",
                                    color = rgb(223, 122, 28, maxColorValue=255),
```

```
                                shape = c("oval", "box"))
        edgesPandoc <- create_edges(from = c("f", "g", "h", "i"),
                                to   = c("i", "i", "i", "j"),
                                color = "black")
        G <- create_graph(nodes_df = combine_nodes(nodes, nodesPandoc),
                          edges_df = combine_edges(edges, edgesPandoc),
                          node_attrs = "fontname=Lato",
                          graph_attrs = c("layout=dot", "rankdir=LR"))
        cat(render_graph(G, output="SVG"), file="workflowPreprocess.svg")
```

In [3]:
```
nodesPandoc <- create_nodes(nodes=letters[(i + 1):(i + length(nodeLabsPandoc) * 2)],
                                label = c(nodeLabsPandoc, nodeLabsPandoc),
                                style = "filled",
                                fontcolor = "white",
                                color = rgb(223, 122, 28, maxColorValue=255),
                                shape = rep(c("oval", "box"), 2))
        edgesPandocIn <- create_edges(from = c("k", "l"),
                                  to   = c("l", "a"),
                                  color = "black")
        edgesPandocOut <- create_edges(from = c("f", "g", "h", "i"),
                                   to   = c("i", "i", "i", "j"),
                                   color = "black")
        G <- create_graph(nodes_df = combine_nodes(nodes, nodesPandoc),
                          edges_df = combine_edges(edgesPandocIn, edges, edgesPandocOut),
                          node_attrs = "fontname=Lato",
                          graph_attrs = c("layout=dot", "rankdir=LR"))
        cat(render_graph(G, output="SVG"), file="workflowSandwich.svg")
```