EQTL example with slurm

I. Bioconductor AMI

There are two AMI's for running this example, one made on the CfnCluster Centos6 AMI and the other made on the CfnCluster Ubuntu 16.04 AMI. (As discussed elsewhere, `cfncluster-1.3.2` can't run slurm on an Ubuntu 16.04 system.) The general procedure to build the AMI was (1) install R 2.4.2, (2) install the BiocInstaller and (3) install `gQTLstats`. Installing `gQTLstats` brings in a large number of very commonly-needed dependencies, e.g., `GenomicRanges`.

R requires multiple dependencies, but most of them are already installed on the CfnCluster AMI's.

Note: custom AMIs need to be rebuilt whenever a new version of CfnCluster is released.

CentOS6 AMI: Begin with the CfnCluster us-east-1 centos6 AMI `ami-a293c9b4`

```
$ sudo yum update                   # installs R-3.4.2
$ sudo yum install libcurl-devel    # need this for RCurl
$ sudo R
> source("https://bioconductor.org/biocLite.R")
> biocLite();
> biocLite("gQTLstats");
> install.packages("batchtools");
> quit()
$ sudo /usr/local/sbin/ami_cleanup.sh      # prepare for AMI
```

Stop instance, make AMI. Result: ami-e2dc4d98

II. EFS Mount Target

Using the AMI for R computations will inevitably necessitate installing more R packages. Rebuilding the AMI every time new packages are installed is time-consuming. Installing the cluster nodes' NFS will result in the packages being deleted with the cluster. An EBS volume can't be mounted by all cluster nodes simultaneously.

A good solution is to use an EFS file system. EFS is an AWS managed service that provides a shared, elastic, available and persistent file system. (Shared means all the nodes can mount it simultaneously. Elastic means it grows to accomodate the data stored in it. Available means there is no one server node (as in NFS) which can fail and bring down the whole system. Persistent means it is not deleted when the instances that mount it are deleted. It is also relatively affordable.)

Our EFS is `fs-58210511`. (Note: This is in my default VPC. Probably need to make a different one for the group.)

A security group needs to be created to give nodes permission to mount the EFS:

1. In the EC2 Instances console, click Security Groups in the left-side menu, click Create Security Group and create two security groups, one for the cluster and one for the efs mount targets.
2. My cluster SG is called `bioc-cfn-sg` and my EFS SG is called `bioc-efs-sg`. `bioc-cfn-sg` allows all incoming traffic from `bioc-efs-sg` and on port 22 from anywhere. (Port 22 is the SSH port. This is so we can SSH to the cluster nodes.) `bioc-efs-sg` allows incoming NFS traffic on port 2049 from `bioc-cfn-sg`.
3. In the EFS console, click the name of the file system, click “Manage file system access” and enter the EFS security group (`bioc-efs-sg`) in each Availability Zone row of the table.

We will make the cluster nodes mount the EFS in the cfncluster post-installation script.

III. Configuration File

```
[aws]
aws_access_key_id = ## AWS Key ID ##
aws_secret_access_key = ## AWS Secret Key ID ##
aws_region_name = us-east-1

[cluster bioc]
vpc_settings = public
key_name = ## Key Name ##
base_os = centos6
s3_read_resource = arn:aws:s3:::cfncluster-eqtl-bucket/*
post_install = s3://cfncluster-eqtl-bucket/postinstall.bioc.sh
custom_ami = ami-e2dc4d98
initial_queue_size = 1
max_queue_size = 1
maintain_initial_size = true
master_instance_type = t2.small
compute_instance_type = t2.small
scheduler = slurm

[vpc public]
vpc_id = ## Default VPC ##
master_subnet_id = ## Any subnet in the default VPC ##
vpc_security_group_id = sg-2c38ce59

[global]
sanity_check = true
update_check = true
cluster_template = bioc
```

Notes:

1. The `post_install script` will mount the EFS. The `s3_read_resource` allows the cluster nodes access to the S3 bucket in which the postinstall script is located.
2. The `custom_ami` is the CentOS6 Bioconductor AMI.
3. The `vpc_security_group_id` is the ID of the group `cfn-bioc-sg`. This enables the nodes to mount the EFS.
4. The installed Bioconductor packages are too large to run on a `t2.micro`, which has only a 1GB of memory.
5. The options initial_queue_size and max_queue_size are misnomers. They are the initial and maximum number of nodes the cluster will launch.

IV. Postinstall Script

The postinstall script, `postinstall.bioc.sh` simply mounts the EFS. The directory will be `/efs` on all nodes.

```
#!/bin/bash

sudo mkdir -p /efs
sudo mount -t nfs -o nfsvers=4.1,rsize=1048576,wsize=1048576,hard,timeo=600,retrans=2 fs-
58210511.efs.us-east-1.amazonaws.com:/ /efs
```

V. Files for running the EQTL Example

For simplicity, run in `/home/centos`. (Recall: `/home` is NFS shared among all nodes in the base Cfncluster AMIs.)

1. `batchtools.conf.R`

```
cluster.functions = makeClusterFunctionsSlurm("./simple.tmpl")
```

2. `simple.tmpl`

```
#!/bin/bash
<%

    # make directory in registry
    rdir <- paste(file.dir, "/rexe", sep="")
    if (!dir.exists(rdir))  {
      dir.create(rdir, recursive = TRUE)
    }

    # make R script for job to run - rscript that runs JobCollection
    rsrc = paste(
          "library(batchtools)",
          "library(gQTLstats)",
          "library(GenomicRanges)",
          sprintf("jc = readRDS('%s')", uri),
          sprintf("batchtools:::doJobCollection(jc, output = '%s')", log.file),
          sep=";")

    rscript <- fp(file.dir, "rexe", sprintf("%s.R", job.hash))
    cat(rsrc,file=rscript,sep="\n")
    Sys.chmod(rscript, mode = "0777")

%>

#SBATCH --job-name=<%= job.name %>
#SBATCH --output=<%= log.file %>
#SBATCH --error=<%= log.file %>
#SBATCH -p compute
#SBATCH -N 1
#SBATCH -n 1
#SBATCH -t 1:00

R CMD BATCH --no-save --no-restore "<%= rscript %>" /dev/stdout
```

3. `.Renviron`

Installed packages should go be default to the EFS, so they do not need to be re-installed when a new cluster is created.

```
R_LIBS="/efs/R/Library"
```

VI. Run EQTL Example

```
library(gQTLstats)
library(ldblock)
library(VariantAnnotation)
library(geuvPack)
library(batchtools)

data(geuFPKM)
ss <- stack1kg()
v17 <- ss@files[[17]]
someGenes <- c("ORMDL3","GSDMB", "IKZF3", "MED24", "CSF3", "ERBB2",
               "GRB7", "MIEN1", "GSDMA", "THRA", "MSL1")
se17 <- geuFPKM[ which(rowData(geuFPKM)$gene_name %in% someGenes),]

n <- 10
vr0 <- 39.5e6
vr1 <- 40.5e6
v <- seq(vr0, vr1, length=n+1)
vl <- zipup(v[1:n],v[2:(n+1)]-1)

run.job <- function(v, se, vcf)  {
  vr <- GRanges("17", IRanges(start=v[1], end=v[2], names=c("range")))
  results <- AllAssoc(se, vcf, vr)
}

concat.job <- function(gr1, gr2)  {
  gr <- c(gr1,gr2)
}

system("rm -Rf ./registry")
reg <- makeRegistry(file.dir="./registry", conf.file = "./batchtools.conf.R")
batchMap(fun = run.job, as.list(vl), more.args = list(se=se17, vcf=v17))

submitJobs()
waitForJobs()    # will take a while because we only have one node
all.results <- reduceResults(fun = concat.job)

save(all.results, file="all.results.RData")
```

VII. Next Steps

The results can be inspected at the R command prompt for correctness. Or they can be downloaded and compared to the results obtained by running the example on the local machine.

The next obvious step is to increase the size of the cluster and the size of the problem. The parallelism will become worthwhile only for reasonably large clusters and problems.

There is a question where to put the very large output of a large problem. An easy option is to put it in the EFS. A more manageable option may be to modify the reduction function to retain only “interesting” results, perhaps only those with a p-value below a certain threshold.