Como usar el cluster sin morir en el intento

Get it started

El cluster (*brick*) consta de tres nodos (*brick01, brick02 y brick03*). Lo que sigue describe la operativa básica para ejecutar tareas en estas maquinas de una manera ordenada

El directorio */nas* de detritus se monta como */home* en cada uno de los *bricks* por lo que todo lo que se modifique en *detritus:/nas/user* cambiara automaticamente en *brick0X:/home/user*

La presentacion inicial (hecha con pinpoint y exportada a pdf) se puede bajar de aqui: http://detritus.fundacioace.com/files/cluster.pdf

como hacer que los nodos no pidan password

We don't talk anymore

Esto es básico para el funcionamiento correcto del resto de las herramientas

Empezamos creando una clave RSA en nuestro directorio de detritus y moviendo la clave publica a un archivo que podamos identificar facilmente

```
$ ssh-keygen
$ mv $HOME/.ssh/id_rsa.pub $HOME/.ssh/detritus.pub
```

Ahora añadimos esta clave a la lista de claves autorizadas en el HOME de los nodos. Como hasta ahora no hay ninguna se puede hacer con *cp*. En caso de haber alguna se podría hacer con *cat*. Y no podemos olvidarnos de dar los permisos correctos!

En mi caso:

```
$ mkdir /nas/osotolongo/.ssh
$ chmod 700 /nas/osotolongo/.ssh
$ cp ~/.ssh/detritus.pub /nas/osotolongo/.ssh/authorized_keys
$ chmod 600 /nas/osotolongo/.ssh/authorized_keys
```

A partir de ahora, cuando se entra de *detritus* a cualquiera de los nodos no deberia ser necesario introducir ningun password.

como ejecutar la misma orden en varios nodos del cluster (pssh y mpssh)

Nobody Wants To Be Lonely

pssh

- primero crear un archivo con la lista de hosts:

```
[osotolongo@detritus ~]$ cat host.pssh
osotolongo@brick01
osotolongo@brick02
osotolongo@brick03
```

- despues ejecutar un comando en todas las maquinas

```
[osotolongo@detritus ~]$ pssh -h host.pssh 'ps ax | grep ssh'
[1] 13:11:46 [SUCCESS] osotolongo@brick03
[2] 13:11:46 [SUCCESS] osotolongo@brick02
[3] 13:11:46 [SUCCESS] osotolongo@brick01
```

mpssh

- la syntaxis cambia ligeramente, y la salida tambien

```
[osotolongo@detritus ~]$ mpssh -f host.pssh 'ps ax | grep ssh'
MPSSH - Mass Parallel Ssh Ver.1.3.3
(c)2005-2013 Nikolay Denev <ndenev@gmail.com>

  [*] read (3) hosts from the list
  [*] executing "ps ax | grep ssh" as user "osotolongo"
  [*] spawning 3 parallel ssh sessions

brick01 ->   1920 ?        Ss     0:00 /usr/sbin/sshd -D
brick01 ->  73087 ?        Ss     0:00 sshd: osotolongo [priv]
brick01 ->  73090 ?        S      0:00 sshd: osotolongo@notty
brick01 ->  73091 ?        Ss     0:00 bash -c ps ax | grep ssh
brick01 ->  73101 ?        S      0:00 grep ssh
brick02 ->   1887 ?        Ss     0:00 /usr/sbin/sshd -D
brick02 ->  21242 ?        Ss     0:00 sshd: osotolongo [priv]
brick02 ->  21245 ?        S      0:00 sshd: osotolongo@notty
brick02 ->  21246 ?        Ss     0:00 bash -c ps ax | grep ssh
brick02 ->  21256 ?        S      0:00 grep ssh
brick03 ->   1928 ?        Ss     0:00 /usr/sbin/sshd -D
brick03 ->  21270 ?        Ss     0:00 sshd: osotolongo [priv]
brick03 ->  21273 ?        S      0:00 sshd: osotolongo@notty
```

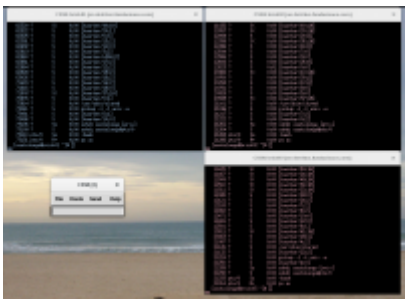
```
brick03 ->  21274 ?        Ss     0:00 bash -c ps ax | grep ssh
brick03 ->  21284 ?        S      0:00 grep ssh

  Done. 3 hosts processed.
```

o como hacerlo mas interactivo (cluster-ssh)

Dance again

Una imagen vale mas que mil palabras

```
$ cssh brick
```



Esta herramienta permite, cuando se escribe en la ventana comun, escribir lo mismo en todas las terminales abiertas o en una sola si se selecciona expresamente.

Slurm

This is what you came for

srun:

Este comando es capaz de lanzar un proceso en varios nodos del cluster al mismo tiempo

```
[osotolongo@detritus ~]$ srun -N3 -l /bin/hostname
0: brick01
2: brick03
1: brick02
```

sbatch:

Este comando lanza un script (tipicamente una lista de comandos *srun*) en el cluster. Es posible escoger el numero de procesos a lanzar e incluso los nodos en los cuales hacerlo

```
[osotolongo@detritus ~]$ cat test_script.sh
#!/bin/sh
#SBATCH --time=1
/bin/hostname
```

```
srun -l /bin/hostname
srun -l /bin/pwd
[osotolongo@detritus ~]$ sbatch -n6 -w "brick0[1-3]" -o test_stdout
test_script.sh
Submitted batch job 30
[osotolongo@detritus ~]$ cat /nas/osotolongo/test_stdout
brick01
1: brick01
2: brick02
4: brick03
3: brick02
5: brick03
0: brick01
0: /home/osotolongo
5: /home/osotolongo
3: /home/osotolongo
1: /home/osotolongo
4: /home/osotolongo
2: /home/osotolongo
```

squeue:

Este comando permite ver los trabajos que se han lanzado e informacion sobre ellos

```
[osotolongo@detritus ~]$ cat test_script.sh
#!/bin/sh
#SBATCH --time=1
/bin/hostname
srun -l /bin/hostname
sleep 20
srun -l /bin/pwd
[osotolongo@detritus ~]$ squeue
             JOBID PARTITION     NAME     USER ST       TIME  NODES
NODELIST(REASON)
[osotolongo@detritus ~]$ sbatch -n6 -w "brick0[1-3]" -o test_stdout
test_script.sh
Submitted batch job 35
[osotolongo@detritus ~]$ squeue
             JOBID PARTITION     NAME     USER ST       TIME  NODES
NODELIST(REASON)
                35     debug test_scr osotolon  R       0:01      3
brick[01-03]
```

scancel:

OK, y cuando la caguemos podemos usar *scancel* para parar los trabajos. El kill de toda la vida pero a traves de un workload manager

```
[osotolongo@detritus ~]$ cat test_script.sh
#!/bin/sh
/bin/hostname
```

```
srun -l /bin/hostname
sleep 200
srun -l /bin/pwd
[osotolongo@detritus ~]$ sbatch -n6 -w "brick0[1-3]" -o test_stdout
test_script.sh
Submitted batch job 36
[osotolongo@detritus ~]$ squeue
             JOBID PARTITION     NAME     USER ST       TIME  NODES
NODELIST(REASON)
                36     debug test_scr osotolon  R       0:03      3
brick[01-03]
[osotolongo@detritus ~]$ scancel 36
[osotolongo@detritus ~]$ squeue
             JOBID PARTITION     NAME     USER ST       TIME  NODES
NODELIST(REASON)
```

https://slurm.schedmd.com/quickstart.html

mas: https://wiki.fysik.dtu.dk/niflheim/SLURM#configure-slurm

---

Nota: Todo lo que estaba hecho en Perl, usando HPC, ha dejado de funcionar. Los modulos dan error. NO obstante, los docs estan aqui: HPC::Runner::Slurm (AKA the hat trick)

The real life (--multi-prog)

Fighter

Vamos a intentar usar todo esto para algo util. Digamos que queremos ejecutar un programa con diferentes argumentos en los nodos del cluster. Para ello usamos **sbatch**

```
$ sbatch test.sh
```

El script *test.sh* no es un programa real sino que es una orden de ejecucion de las tareas.

test.sh

```
#!/bin/bash
#SBATCH --time=100
#SBATCH --ntasks-per-node=20
#SBATCH -n 50
#SBATCH --mail-type=ALL
#SBATCH --mail-user=osotolongo
srun --exclusive --multi-prog test.conf
```

Este script lo unico que hace es configurar minimamente **sbatch** y lanzar el comando **srun**.

- La directiva *time* especifica cuantos minutos dejaremos correr el trabajo. Con *–time=0* no se aplica límite de tiempo.
- Las directivas importantes aqui son *–ntasks-per-node=20* y *-n 50*. La primera dice a **sbatch** que lance 20 tareas en cada nodo, la segunda que hay un total de 50 tareas. para que todo vaya bien n/ntask-per-node < 3. Por ejemplo si ponemos *–ntasks-per-node=10*, entonces dara error porque 50/10=5. *Esto se puede quitar pero si lanzamos 50 tareas las metera todas en brick01 y el ejemplo no servira de nada*.
- Las directivas *–mail-type=ALL* y *–mail-user=osotolongo* dicen que envie por email todo lo que ocurra al usuario *osotolongo*. Si el usuario de cada uno esta bien configurado, esto deberia funcionar para cada uno con solo cambiar el nombre.
- El comando **srun** lee el archivo *test.conf* y ejecuta las ordenes que hay dentro de este archivo.

El archivo *test.conf* tiene la siguiente estructura

```
0       test_runner.sh test/blah1.txt
1       test_runner.sh test/blah2.txt
2       test_runner.sh test/blah3.txt
3       test_runner.sh test/blah4.txt
4       test_runner.sh test/blah5.txt
5       test_runner.sh test/blah6.txt
.
.
.
45      test_runner.sh test/blah46.txt
46      test_runner.sh test/blah47.txt
47      test_runner.sh test/blah48.txt
48      test_runner.sh test/blah49.txt
49      test_runner.sh test/blah50.txt
```

La primer columna debe numerar las tareas, comenzando en **0** y terminando en **n-1**.La segunda columna no es mas que la orden a ejecutar (en el ejemplo *test_runner.sh*),con todos los argumentos correspondientes (en este caso un nombre de archivo).

Probando

```
[osotolongo@detritus cluster]$ sbatch test.sh
Submitted batch job 842
[osotolongo@detritus cluster]$ squeue
             JOBID PARTITION     NAME     USER ST       TIME  NODES
NODELIST(REASON)
               842     debug  test.sh osotolon  R       0:03      3
brick[01-03]
```

El programa a ejecutar es un script sencillo que recibe como argumento un nombre de archivo,

```
[osotolongo@detritus cluster]$ cat test_runner.sh
#!/bin/sh
file=$1
count=0
```

```
while [ $count -le 10 ]
do
        echo $count >> $file
        ((count++))
        sleep 50
done
echo Nice! >> $file
```

y tambien me he escrito un script para que me escriba el *test.conf*,

```
[osotolongo@detritus cluster]$ cat test_generator.sh
#!/bin/sh
count=0
while [ $count -lt 50 ]
do
        cn=$((count+1))
        echo "$count    test_runner.sh test/blah${cn}.txt" >> test.conf
        ((count++))
done
```

Luego, corro primero,

```
$ ./test_generator.sh
```

que me hace el *test.conf* y luego,

```
$ sbatch test.sh
```

Los archivos se generan correctamente,

```
[osotolongo@detritus cluster]$ ls test
blah10.txt  blah15.txt  blah1.txt   blah24.txt  blah29.txt  blah33.txt
blah38.txt  blah42.txt  blah47.txt  blah5.txt
blah11.txt  blah16.txt  blah20.txt  blah25.txt  blah2.txt   blah34.txt
blah39.txt  blah43.txt  blah48.txt  blah6.txt
blah12.txt  blah17.txt  blah21.txt  blah26.txt  blah30.txt  blah35.txt
blah3.txt   blah44.txt  blah49.txt  blah7.txt
blah13.txt  blah18.txt  blah22.txt  blah27.txt  blah31.txt  blah36.txt
blah40.txt  blah45.txt  blah4.txt   blah8.txt
blah14.txt  blah19.txt  blah23.txt  blah28.txt  blah32.txt  blah37.txt
blah41.txt  blah46.txt  blah50.txt  blah9.txt
```

y el contenido es el que debe ser,

```
[osotolongo@detritus cluster]$ cat test/blah1.txt
0
1
2
3
4
```

```
5
6
7
8
9
10
Nice!
```

Los emails también se envian correctamente,

Hard batching

Run to the hills

El procedimiento anterior funciona OK cuando se lanzan pocas tareas que consuman tiempo. Si hemos de lanzar muchas tareas rapidas lo mejor no es usar un solo *srun* con el switch *–multi-prog* sino lanzar un *sbatch* para cada proceso.

Ahora el *test_generator* es distinto, cada proceso se ejecutara con un sbatch, que hay que construir en un bucle. Luego basta con hacer un ejecutable que lance los *sbatch* consecutivamente,

test_generator2.sh

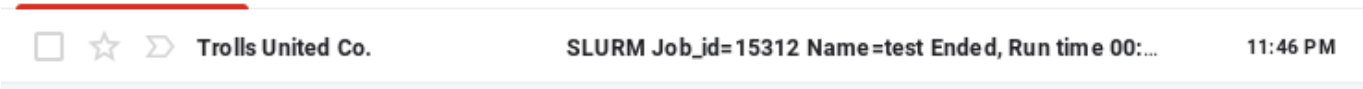
```
#!/bin/sh
#Creo los batch a ejecutar
count=0
while [ $count -lt 50 ]
do
        cn=$((count+1))
        echo "#!/bin/bash" > test_run_${cn}.sh
        echo "#SBATCH -J test" >> test_run_${cn}.sh
        echo "#SBATCH --mail-type=FAIL,TIME_LIMIT,STAGE_OUT" >>
test_run_${cn}.sh
        echo "#SBATCH --mail-user=`whoami`" >> test_run_${cn}.sh
        echo "srun test_runner.sh test/blah${cn}.txt" >>
test_run_${cn}.sh
        ((count++))
done
#Creo un archivo que ejecute los batch
count=0
echo "#!/bin/bash" > test.sh
while [ $count -lt 50 ]
do
        cn=$((count+1))
        echo "sbatch test_run_${cn}.sh" >> test.sh
```

```
        ((count++))
done
# Creo un script de aviso
echo "#!/bin/bash" > mailme.sh
echo "#SBATCH -J test" >> mailme.sh
echo "#SBATCH --mail-type=END" >> mailme.sh
echo "#SBATCH --mail-user=`whoami`" >> mailme.sh
echo "srun :" >> mailme.sh
echo "sbatch --dependency=singleton mailme.sh" >> test.sh
chmod +x test.sh
```

Nota: Cada proceso se ejecuta independientemente y solo envia emails en caso de fallo, para recibir un aviso cuando se termina la ejecucion es necesario crear y ejecutar un porceso que lo haga y que dependa especificamente de que todos los procesos previos terminen. La forma natural es ejecutarlo con el comando *sbatch –dependency=singleton*, pero ojo que esto se ejecutara cuando todos los procesos con elmismo nombre y bajo el mismo usuario hayan terminado, asi que ha de asignarse a cada proceso el mismo nombre (*swarm*). Si hay que lanzar dos *swarm* diferentes hay que hacerlo con un nombre distinto y correr porque seras condenado a la guillotina por el administrador del cluster. Ojo, que este proceso (que aparece como Dependency bajo *squeue*) en realidad no hace nada, solo manda un email cuando se ejecuta. Vamos a probarlo,

```
[osotolongo@detritus cluster]$ ./test_generator2.sh
[osotolongo@detritus cluster]$ ls
mailme.sh       test_run_10.sh  test_run_15.sh  test_run_1.sh
test_run_24.sh  test_run_29.sh  test_run_33.sh  test_run_38.sh
test_run_42.sh  test_run_47.sh  test_run_5.sh  test_runner.sh
test           test_run_11.sh  test_run_16.sh  test_run_20.sh
test_run_25.sh  test_run_2.sh   test_run_34.sh  test_run_39.sh
test_run_43.sh  test_run_48.sh  test_run_6.sh  test.sh
test.conf       test_run_12.sh  test_run_17.sh  test_run_21.sh
test_run_26.sh  test_run_30.sh  test_run_35.sh  test_run_3.sh
test_run_44.sh  test_run_49.sh  test_run_7.sh
test_generator2.sh  test_run_13.sh  test_run_18.sh  test_run_22.sh
test_run_27.sh  test_run_31.sh  test_run_36.sh  test_run_40.sh
test_run_45.sh  test_run_4.sh   test_run_8.sh
test_generator.sh   test_run_14.sh  test_run_19.sh  test_run_23.sh
test_run_28.sh  test_run_32.sh  test_run_37.sh  test_run_41.sh
test_run_46.sh  test_run_50.sh  test_run_9.sh
[osotolongo@detritus cluster]$ cat test_run_1.sh
#!/bin/bash
#SBATCH -J test
#SBATCH --mail-type=FAIL,TIME_LIMIT,STAGE_OUT
#SBATCH --mail-user=osotolongo
srun test_runner.sh test/blah1.txt
[osotolongo@detritus cluster]$ cat test.sh
#!/bin/bash
sbatch test_run_1.sh
sbatch test_run_2.sh
sbatch test_run_3.sh
sbatch test_run_4.sh
```

```
sbatch test_run_5.sh
.....................
sbatch test_run_50.sh
sbatch --dependency=singleton mailme.sh

[osotolongo@detritus cluster]$ ./test.sh
Submitted batch job 15262
Submitted batch job 15263
Submitted batch job 15264
Submitted batch job 15265
Submitted batch job 15266
Submitted batch job 15267
Submitted batch job 15268
...............

[osotolongo@detritus cluster]$ squeue
             JOBID PARTITION     NAME     USER ST       TIME  NODES
NODELIST(REASON)
             15312     devel     test osotolon PD       0:00      1
(Dependency)
             15262     devel     test osotolon  R       0:03      1 brick01
             15263     devel     test osotolon  R       0:03      1 brick01
             15264     devel     test osotolon  R       0:03      1 brick01
             15265     devel     test osotolon  R       0:00      1 brick01
             15266     devel     test osotolon  R       0:00      1 brick01
.....................
```



Paralelizando un archivo de ordenes

Podemos abstraer todo esto en un script simple para paralelizar las ordenes escritas en un solo archivo.

Ejemplo, este script,

slurmize.pl

```
#!/usr/bin/perl
use strict;
use warnings;

#Cambiar aqui el tiempo maximo de ejecucion
my $time = '3:0:0';

my $ifile = $ARGV[0];
my $wdir = 'slurm';
mkdir $wdir;
```

```
my $count = 0;
open IPDF, "<$ifile" or die "Could not open input file\n$!\n";
while (<IPDF>) {
        $count++;
        my $ofile = sprintf ("%s_%04d", 'sorder', $count);
        $ofile = $wdir.'/'.$ofile;
        open ORD, ">$ofile";
        print ORD '#!/bin/bash'."\n";
        print ORD '#SBATCH -J '.$ifile."\n";
        print ORD '#SBATCH -c 8'."\n";
        print ORD '#SBATCH --mem-per-cpu=4G'."\n";
        print ORD '#SBATCH --time='.$time."\n"; #si no ha terminado en
X horas matalo
        print ORD '#SBATCH --mail-type=FAIL,TIME_LIMIT,STAGE_OUT'."\n";
#que mande email solo cuando falle
        print ORD '#SBATCH --mail-user='."$ENV{'USER'}\n";
        print ORD '#SBATCH -p fast'."\n";
        print ORD '#SBATCH -o '.$wdir.'/'.$ifile.'-%j'."\n";
        print ORD "srun $_\n";
        close ORD;
        system("sbatch $ofile");
}
close IPDF;
my $orderfile = $wdir.'/'.$ifile.'_end.sh';
open ORD, ">$orderfile";
print ORD '#!/bin/bash'."\n";
print ORD '#SBATCH -J '.$ifile."\n";
print ORD '#SBATCH --mail-type=END'."\n"; #email cuando termine
print ORD '#SBATCH --mail-user='."$ENV{'USER'}\n";
print ORD '#SBATCH -o '.$wdir.'/'.$ifile.'_end-%j'."\n";
print ORD ":\n";
close ORD;
my $xorder = 'sbatch --dependency=singleton'.' '.$orderfile;
exec($xorder);
```

lee un archivo de entrada y ejecuta cada linea como una tarea de SLURM. Para esto basta hacer,

```
$ ./slurmize.pl ordenes.txt
```

donde *ordenes.txt* es cualquier listado de tareas.

ejemplo

- Las ordenes deben ejecutarse dentro del directorio /nas/ para que el sistema de archivos sea accesible a todos los nodos.
- Escribo las ordenes dentro de un archivo (en este caso *test*)
- Ejecuto el script dando como argumento el nombre del archivo de ordenes
- Las tareas seenvian a los nodos del cluster
- El output (STDOUT & STDERR) queda dentro del directorio *slurm* en un archivo distinto para

cada proceso

- Tambien pueden mirarse los scripts individuales que quedan en el mismo directorio

```
[osotolongo@detritus slurmit]$ pwd
/nas/osotolongo/slurmit
[osotolongo@detritus slurmit]$ cat test
sleep 2000;hostname
sleep 2000;hostname
sleep 2000;hostname
[osotolongo@detritus slurmit]$ ./slurmize.pl test
Submitted batch job 129583
Submitted batch job 129584
Submitted batch job 129585
Submitted batch job 129586
[osotolongo@detritus slurmit]$ squeue | grep test
            129586      fast     test osotolon PD       0:00      1
(Dependency)
            129583      fast     test osotolon  R       0:06      1 brick02
            129584      fast     test osotolon  R       0:06      1 brick02
            129585      fast     test osotolon  R       0:06      1 brick02
[osotolongo@detritus slurmit]$ tree
.
├── slurm
│   ├── sorder_0001.sh
│   ├── sorder_0002.sh
│   ├── sorder_0003.sh
│   ├── test-129583
│   ├── test-129584
│   ├── test-129585
│   └── test_end.sh
├── slurmize.pl
└── test

1 directory, 9 files
```