Overview of a virtual cluster

using OpenNebula and SLURM

Ismael Farfán Estrada
ifarfane0900@ipn.mx

October 27, 2011

Introduction

Use virtual machines as means for “dynamic fractional resource scheduling” in a batch cluster environment.
Software used:

SLURM A resource manager, it’s job is to receive batch scripts (job requests) and allocate them in the available (virtual) nodes for execution.

OpenNebula A virtual machine management service, makes it easy to work with VMs providing an easy way to create, destroy, migrate, contextualize, etc. virtual machines.

Physical connection

- The main/login server executes the SLURM resource manager and OpenNebula.
- The job of the worker nodes will be to host running VMs.
- The VMs execute the batch jobs.
- The VMs connect to the physical network using bridged connections.

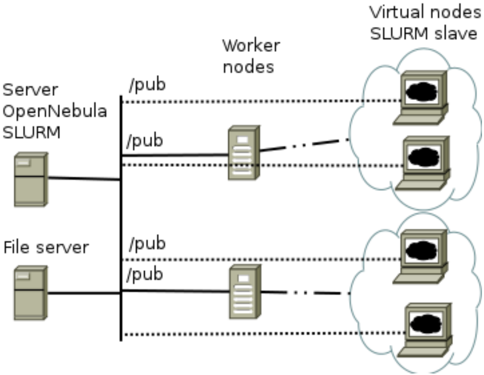
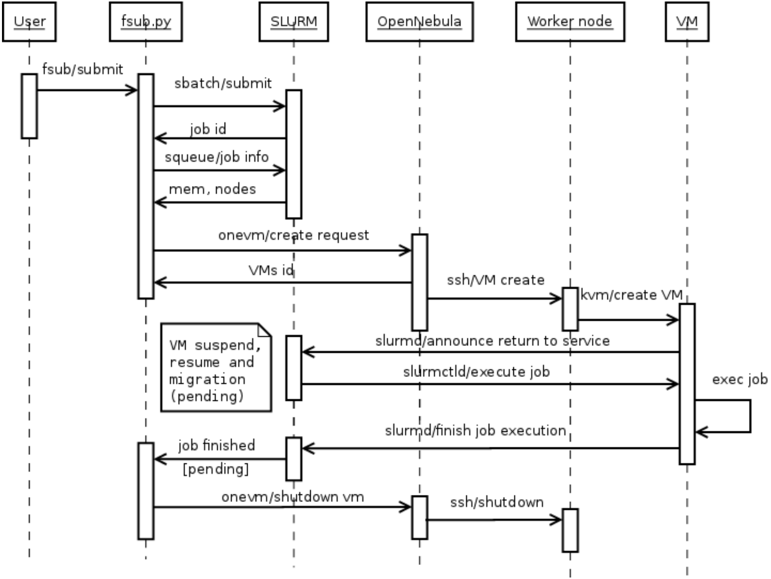


Figure: Client/server topology for the virtual cluster

Job execution sequence diagram



Job execution sequence explained

1. The user submits the job using fsub.py (Not necessary if it integrates with SLURM triggers.).
2. Call sbatch to send the job to the SLURM daemon.
3. Retrieve the job's requirements (nodes and memory).
4. Launch as many VMs as nodes requested that fit the requirements.
5. Wait for the VM to inform that its ready to execute the jobs.
6. Execute the job.
7. Suspend, resume, migrate VM (not working yet).
8. Inform job completion, cancel, kill, etc.
9. Shut down the VMs.

Getting job requirements

This command return the requested nodes and memory:
squeue -o ’%5D %7m’ -j job-id

```
def obten_requisitos_trabajo(tid):
    param = shlex.split( "squeue -o '%5D %7m' -j" + str(tid
    p = subprocess.Popen(param, stdout=subprocess.PIPE)
    rt = p.communicate()
    rt = rt[0].split("\n")[1].split()
    nodos    = int( rt[0] )
    memoria = int( rt[1] )
    if memoria < 50: memoria = 256
    return nodos, memoria
```

Launch the VMs

Create a VM template, execute the VM, register the tuple (VM-id, job-id) on the database.

```
def lanza_vms(vms, mem, tid):
    ruta = "/tmp/"
    for v in vms:
        f = open(ruta+"vm%d.one"%v, "w")
        f.write( definicion%(v, mem, v, v, v) )
        f.close()
        param = shlex.split( "onevm -v create " + f.name )
        p = subprocess.Popen(param, stdout=subprocess.PIPE)
        oid = p.communicate()
        oid = int ( oid[0].split()[1] ) # ('ID: 40\n', None
        con.execute( "update mapa set oneid=%d, tid=%d wher
```

Cleaning upon job termination

Shutdown the VMs and update the database.

```
def limpia_trabajo(tid):
    print "limpiando", tid
    vms = lista_vms("where tid=%d"%tid)
    for v in vms:
        v = v[1]
        print "apagando vm", v
        param = shlex.split( "onevm -v shutdown %d"%v )
        p = subprocess.Popen(param, stdout=subprocess.PIPE)
        print p.communicate()
        con.execute("update mapa set tid=-1, oneid=-1 where
```

Current test environment

- A laptop running Debian as a SLURM and OpenNebula server (with a public IP address).
- 2 worker nodes with 8-cores, 32Gb RAM and RHEL 6.0 on sierra.futuregrid.org.
- A shared directory in the NFS server of sierra for the VMs.
- Debian VMs as worker nodes.
- SQLite database for job/VM mapping purposes.

stream memory benchmark

s80r,s82r 8 core physical nodes
fg0,fg2 3 core VMs running in s80r
fg1 3 core VM running in s81r

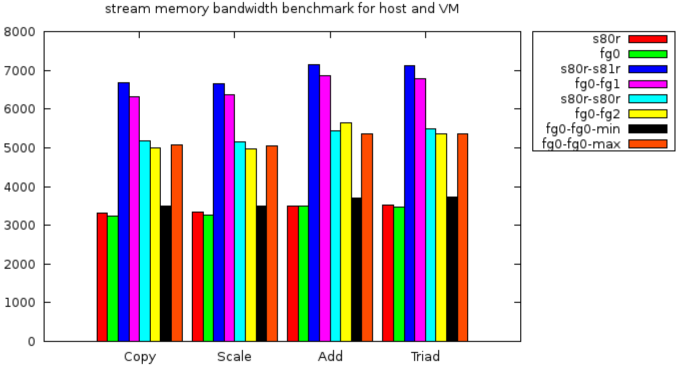


Figure: Original and MPI stream benchmark results

What (doesn't) works

y Create VMs

y Job requirement retrieval

y VM contextualization

y VM migration

N Live VM migration (problems with credentials, shared FS, configuration?)

N Suspending and resuming jobs and the VMs executing them (can't setup the support, SLURM reports “node failure” (and kills the job))