# ARC Control Tower

## A flexible generic distributed job management framework

Jon Kerr Nilsen
David Cameron
Andrej Filipčič

norden

NordForsk

neic

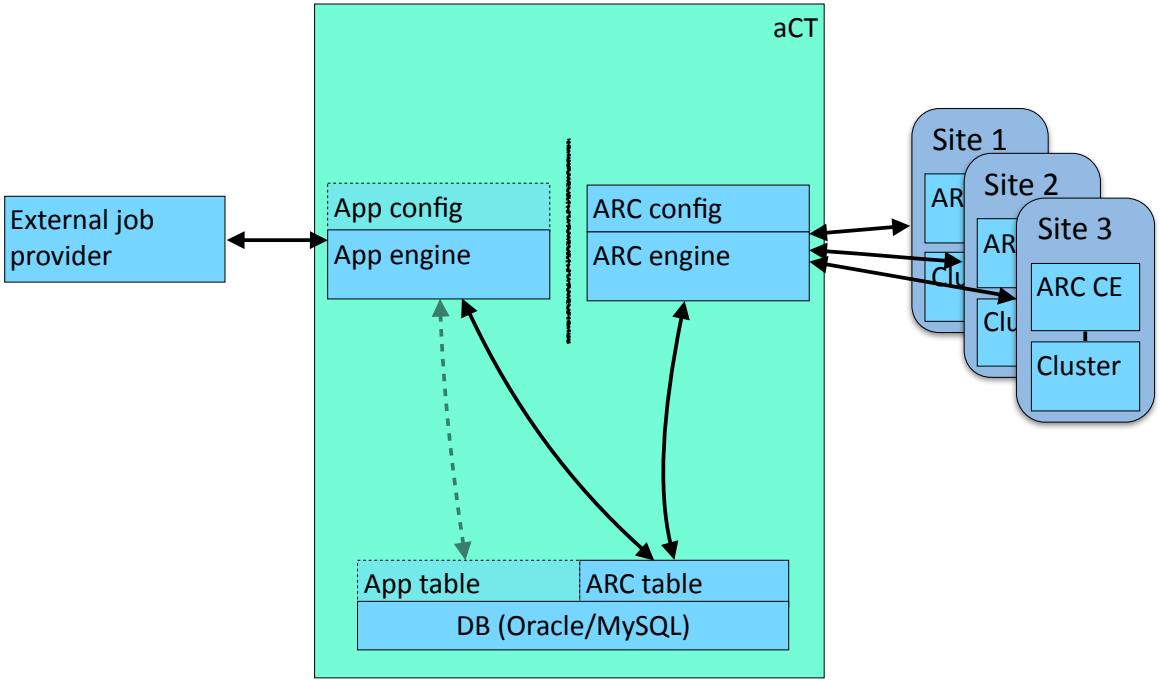
Nordic e-Infrastructure Collaboration

# Why do we need aCT?

- ATLAS Pilot model is not suitable for all available hardware
  - See Andrej's talk (#145)
- Middlewares in general come with rather minimal job management systems
  - Client tools are too simplistic for handling large sets of jobs
  - Tools for handling large amounts of jobs are left to the user/project/experiment to develop

# What is aCT?

- ARC Control Tower is a job management layer in front of ARC CEs
  - Picks up job descriptions from external job provider
  - Converts them to XRSL job description
  - Submits and manages jobs on ARC CEs
  - Fetches logfiles, validates output, handles common failures and updates job status

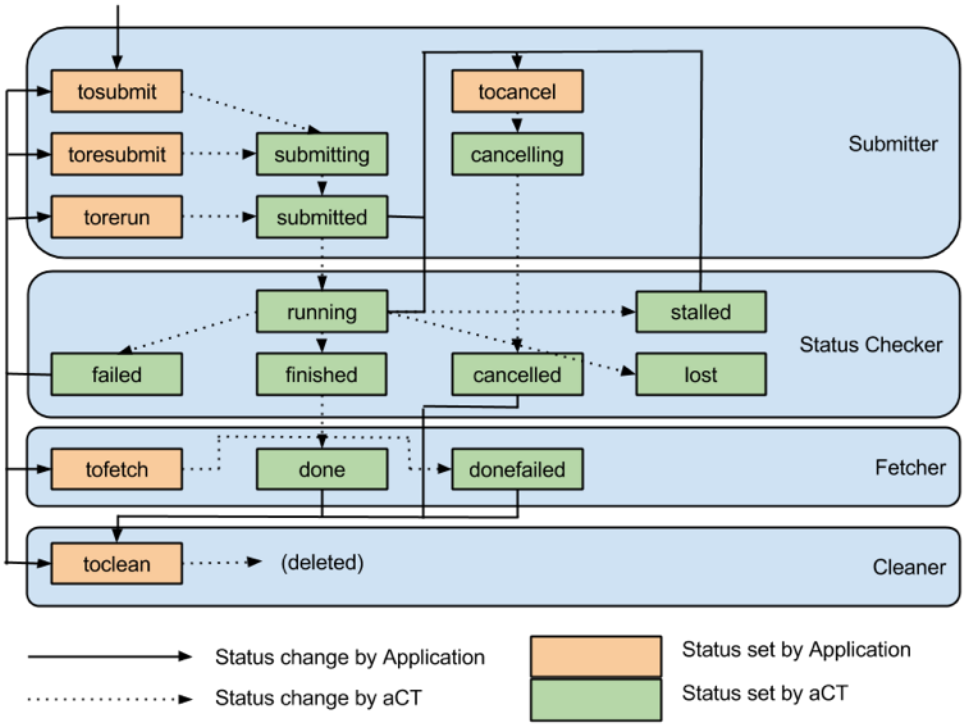


# aCT actors and processes

- aCT consists of actors working on job states
- An actor processes jobs in specific states and moves them to the next state
- One set of ARC actors per cluster
  - For performance and fault tolerance – one site having network/ hardware issues only affects jobs at that cluster
- Application actors interface between external job providers and aCT
- Additionally, some processes to manage the actors, keeping proxies valid, etc.

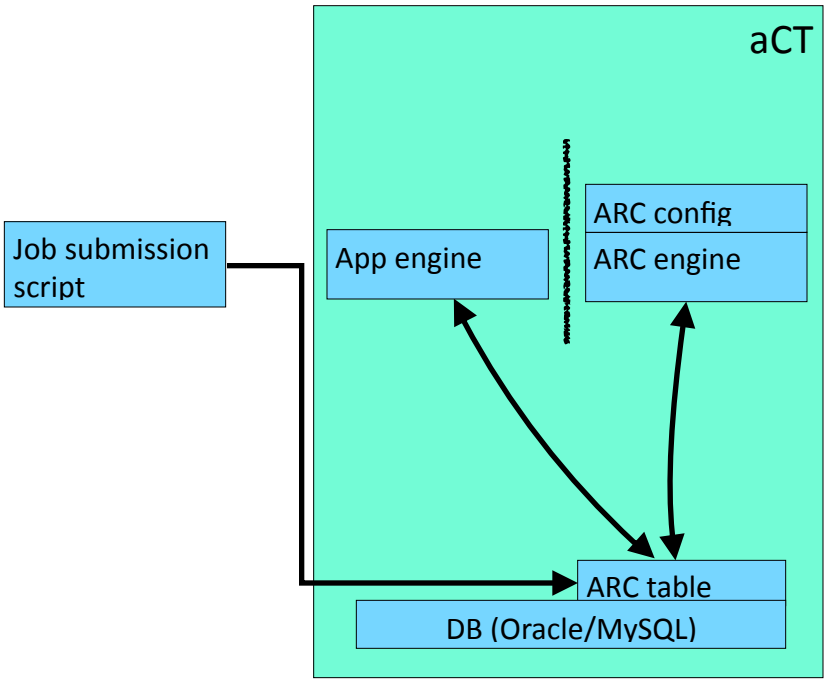
# Acting on job states

- Actors move jobs between states
  - **Submitter** checks jobs to be *submitted* or *cancelled*, and submits or cancel them
  - **Status checker** checks jobs that are *submitted*, *running* or *cancelling* and moves them to *running*, *finished*, *failed*, *lost*, *cancelled*…
  - **Fetcher** checks jobs that are finished and moves them to done. Successful jobs are fetched per default, failed jobs are only fetched if app engine sets state to *tofetch*.
  - **Cleaner** checks jobs to be cleaned and deletes them from system



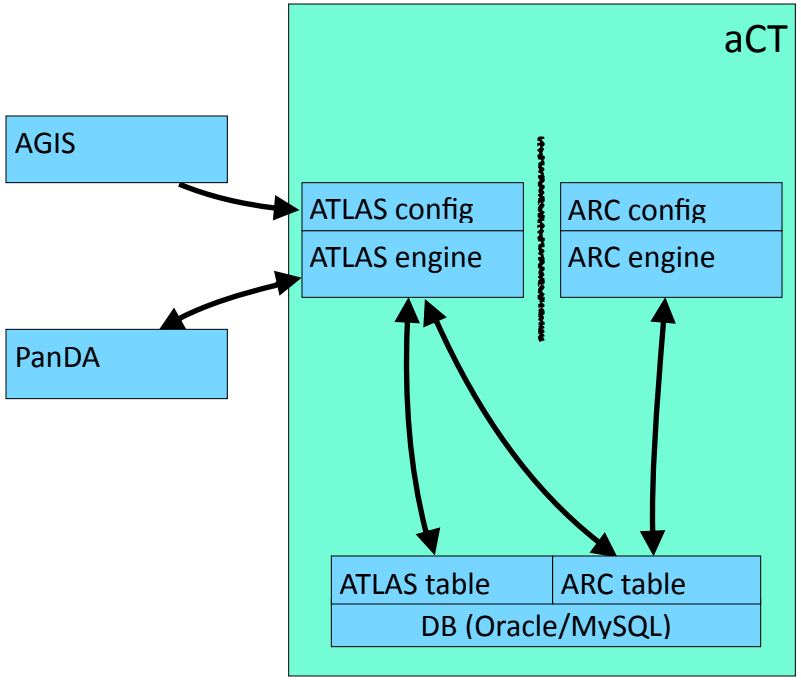
# Example 1: Simple Job Manager

- Job provider is just a simple Python script injecting XRSL directly to ARC table
- No application table needed
- App engine consists of two actors
  - **Status checker** checks if there are *failed* jobs and asks the ARC engine to fetch them. When fetched, the jobs are moved to *toclean*
  - **Validator** checks for *finished* jobs, validates the result and moves the jobs to *toclean*
- (See extra slides for demo and code details)



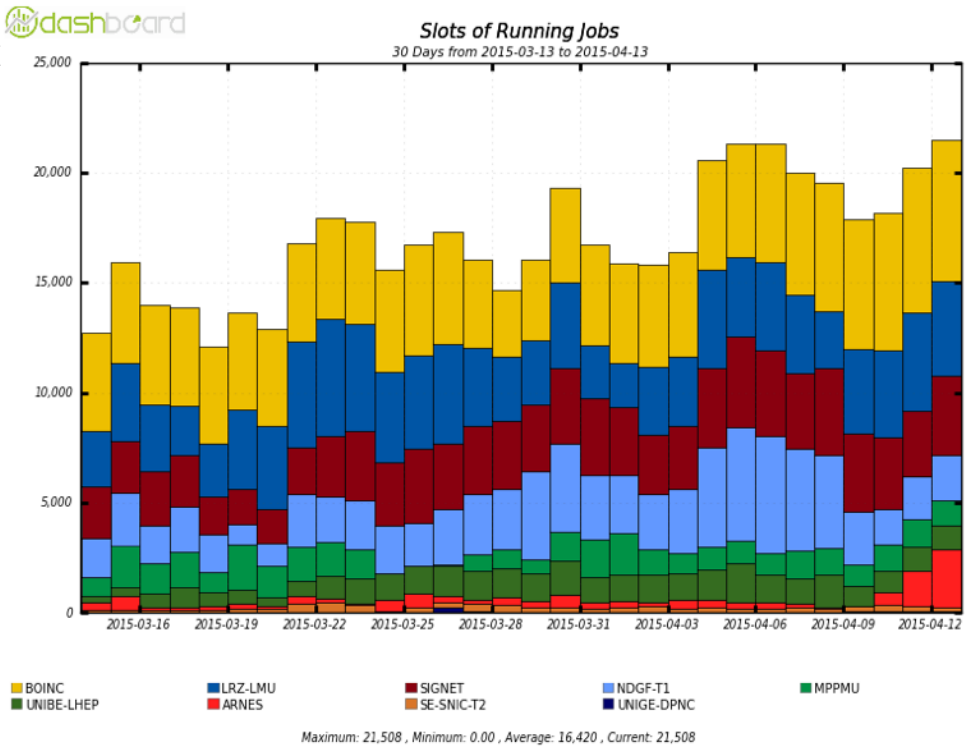
# Example 2: ATLAS application

- Jobs are provided by PanDA, site info is fetched from AGIS
- ATLAS app engine has 6 actors and processes to fetch, validate and check status of jobs and to communicate with external job provider
- Separate app table to handle PanDA job states, application specific job states, PanDA heartbeats, etc.



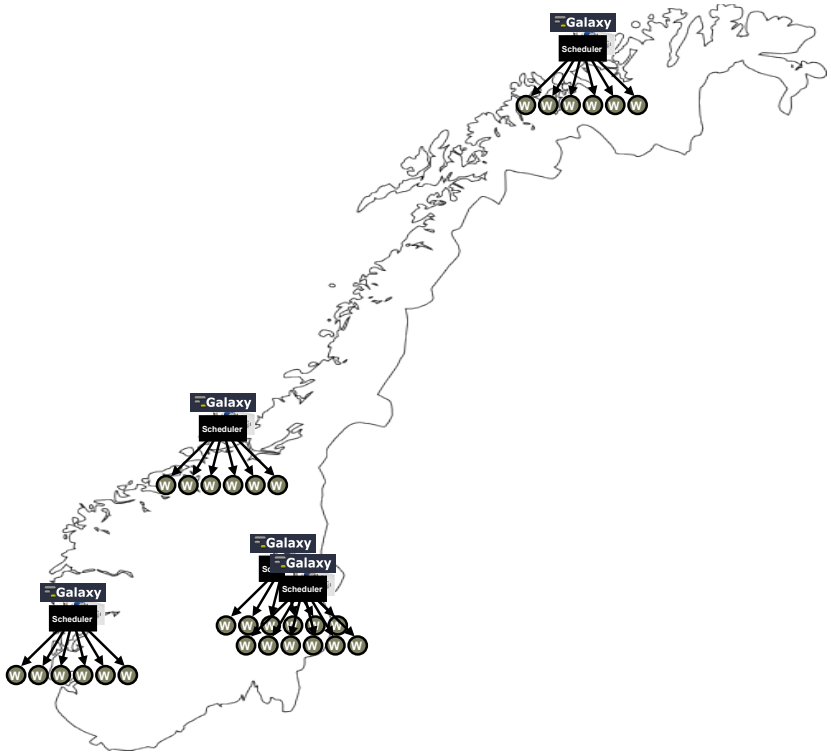
# Example 2: ATLAS application

- Serves ~20k cores
- Utilizes both pledged and opportunistic hardware
  - WLCG grid sites
  - HPC sites (see posters B/92, A/153, A/161)
  - BOINC (see David's talk #170)
- True-pilot mode (see Andrej's talk #145)
  - makes aCT drop-in replacement for ATLAS Pilot Factory
- No «WMS mode» for ATLAS, aCT takes care direct submission, payload distribution
- Automatically fills available resources according to their capacities
  - No. of queued jobs = running*0.15+100



# Example 3: Connecting Norwegian HPC sites for Life Sciences

- NeLS (Norwegian e-Infrastructure for Life Sciences) aims to build a Norwegian node in the ELIXIR infrastructure
- NeLS partners provide HPC resources for Life Science applications at five sites
- Uses Galaxy for enabling access to the resources
  - Galaxy is an open, web-based portal for data intensive biomedical research

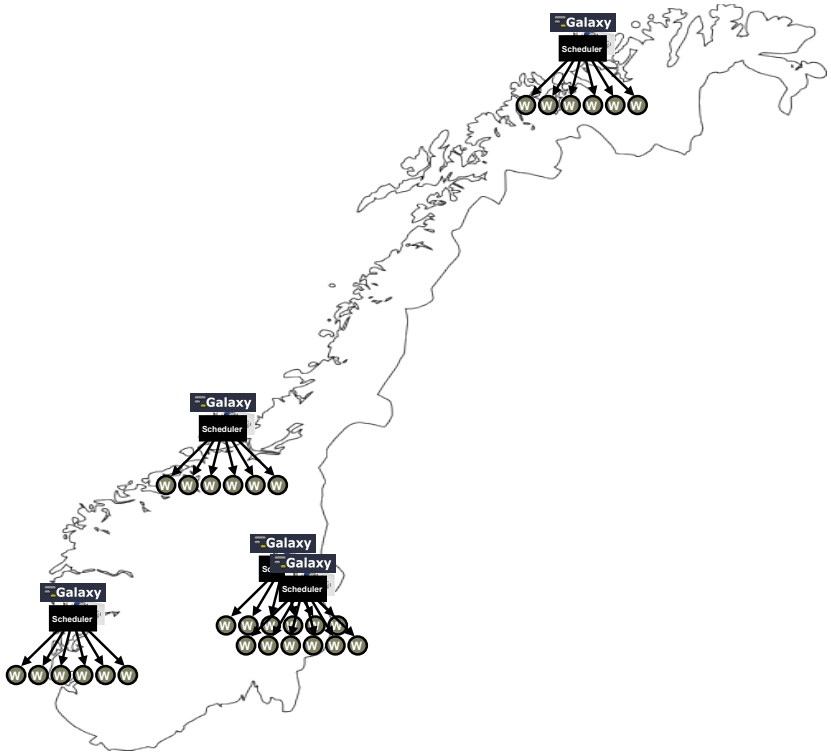



NTNU
Kunnskap for en bedre verden
Norwegian University of Life Sciences
The Research Council of Norway

Figures from Abdulrahman Azab (NeLS)

# Example 3: Connecting Norwegian HPC sites for Life Sciences

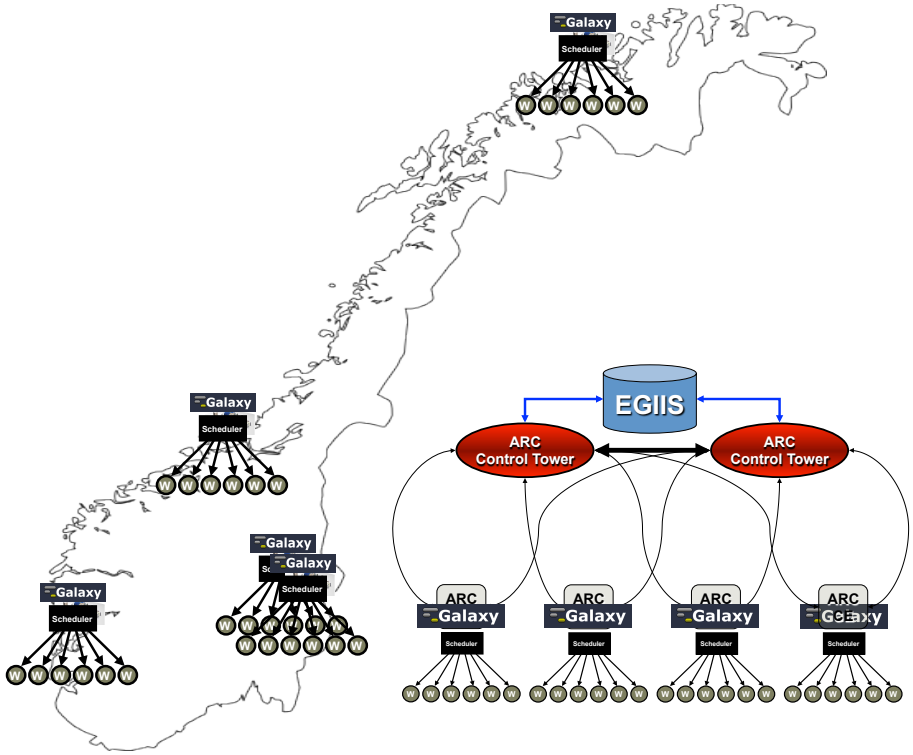
- Problem: Galaxy only supports one cluster per instance -> no load balancing between clusters




Figures from Abdulrahman Azab (NeLS)

# Example 3: Connecting Norwegian HPC sites for Life Sciences

- Problem: Galaxy only supports one cluster per instance -> no load balancing between clusters
- Solution:
  - Each site installs ARC CE
  - Galaxy pushes jobs to aCT
  - aCT takes care of load balancing as part of job management
- Requires developing an aCT plugin in Galaxy

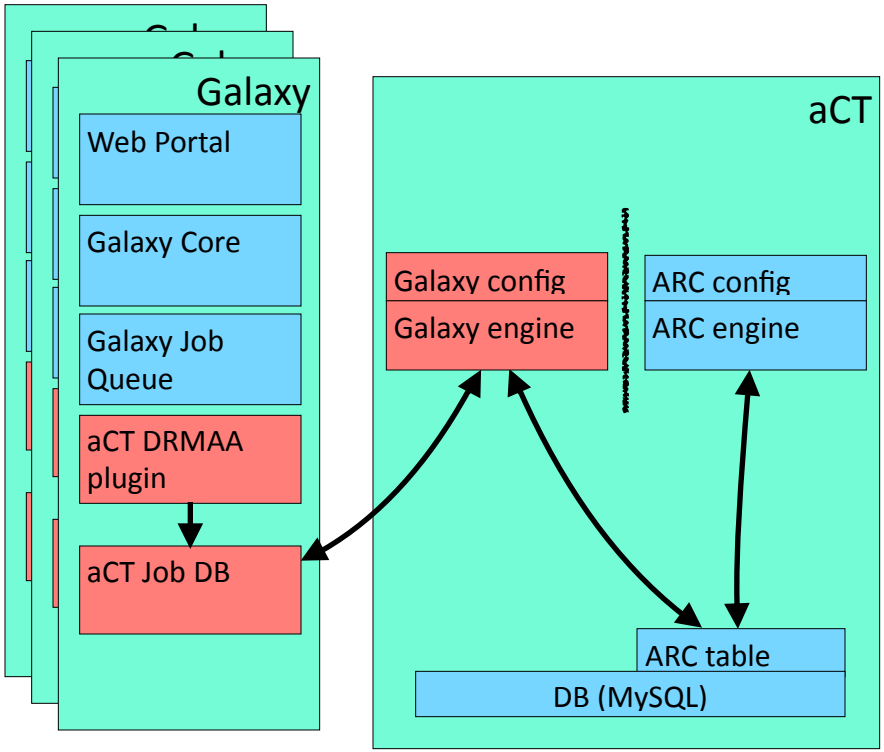





Figures from Abdulrahman Azab (NeLS)

# Example 3: Connecting Norwegian HPC sites for Life Sciences

- aCT DRMAA plugin to Galaxy
  - Inserts jobs into aCT Job DB
- aCT Galaxy engine pulls jobs from given aCT Job DBs remotely and inserts them into ARC tables
- One aCT Job DB per Galaxy instance
- One or more aCT instances pull from multiple Galaxy instances



# Conclusions

- Modular design makes it easy to adapt aCT to various usecases
- In case of ATLAS, provides access to non-traditional resources (BOINC, HPC)
- True-pilot mode makes aCT drop-in replacement for ATLAS pilot factory
- Plans to use aCT as a load balancer for Norwegian HPC sites

# Thank you!

# Extra slides

# Random demo

- Goal: generate distributed random numbers
  - A random generator on a machine is only pseudo-random
  - If we pick random numbers from several machines it should be more random (any mathematicians around?)
  - So if we run a bunch of grid jobs, each picking a random number we should get a bunch of really random numbers
- How:
  1. Make a Simple Job Manager
  2. Create submission script
  3. Run

# aCTSimpleStatus

```
import arc

from aCTProcess import aCTProcess

class aCTSimpleStatus(aCTProcess):

    def checkFailed(self):
        """
        Check for jobs with status failed and donefailed.
        - If failed, set tofetch to allow manual inspection later.
        - If donefailed, the job is fetched and can be cleaned up.
        """
        select="arcstate='failed'"
        desc = {"arcstate":"tofetch", "tarcstate": self.db.getTimeStamp()}
        self.db.updateArcJobs(desc, select)

        select="arcstate='donefailed'"
        desc = {"arcstate":"toclean", "tarcstate": self.db.getTimeStamp()}
        self.db.updateArcJobs(desc, select)


    def process(self):

        # clean jobs
        self.checkFailed()


if __name__ == '__main__':
    st=aCTSimpleStatus()
    st.run()
    st.finish()
```

# aCTSimpleValidator

```
class aCTSimpleValidator(aCTProcess):

    def getResult(self, jobid):
        localdir = str(self.conf.get(['tmp','dir'])) + jobid[jobid.rfind('/'):]
        with open(os.path.join(str(self.conf.get(['tmp','dir'])), "res.txt"), "a") as of:
            with open(os.path.join(localdir, "stdout"), "r") as out:
                of.write(out.read())

    def cleanLocal(self, jobid):
        localdir = str(self.conf.get(['tmp','dir'])) + jobid[jobid.rfind('/'):] + '/'
        shutil.rmtree(localdir)

    def validateFinished(self):
        """
        Check for jobs with status done. If done, dump stdout to result file.
        """
        select="arcstate='done'"
        columns = ["id", "JobID"]
        donejobs = self.db.getArcJobsInfo(select, columns)

        for job in donejobs:
            self.getResult(job['JobID'])
            self.cleanLocal(job['JobID'])
            self.db.updateArcJob(job['id'], {"arcstate": "toclean",
                                             "tarcstate": self.db.getTimeStamp()})
```

# Registering the new actors

- Add the following to aCTProcessManager.py:

```
# dictionary of processes:aCTProcessManager of which to run a single instance
 self.processes_single = {'aCTSimpleStatus':None,
                          'aCTSimpleValidator':None,
                          ‘aCTProxyHandler’:None}
```

# Submit script: random.py

```
from aCTLogger import aCTLogger
from aCTDBArc import aCTDBArc
from aCTProxy import aCTProxy
import sys

def getProxyId(p):
    dn="/O=Grid/O=NorduGrid/OU=fys.uio.no/CN=Jon Kerr Nilsen"
    voms="atlas"
    attribute="" # e.g. attribute="/atlas/Role=production"
    proxyid = p.getProxyId(dn, attribute)
    if not proxyid:
        proxypath="/tmp/x509up_u47107"
        validTime=5*3600
        proxyid = p.createVOMSAttribute(voms, attribute, proxypath, validTime)
    return proxyid

logger=aCTLogger("random")
log=logger()
db = aCTDBArc(log, "act")

xrsl = '''&(executable=/bin/echo)
           (arguments="$RANDOM")
           (stdout=stdout)
           (rerun=2)
           (gmlog=gmlog)
           (wallTime="1")
           '''

proxyid = getProxyId(aCTProxy(log))

try:
    clusters=sys.argv[2]
except:
    clusters=''

for i in range(int(sys.argv[1])):
   db.insertArcJobDescription(xrsl, maxattempts=2, proxyid=proxyid, clusterlist=clusters)
```

# Run

- Fire up aCT

```
> python $ACT_LOCATION/aCTMain.py start
```

- Run the script (10k jobs, random sites)

```
> python random.py 10000
```