TROY

Tiered Resource OverlaY

Andre Merzky

http://saga-project.github.com/troy/

Motivation for TROY

- provide uniform interface to pilot frameworks (PF)

Motivation for TROY

- provide uniform interface to pilot frameworks (PF)
- support for application level scheduling on those PFs

Motivation for TROY

- provide uniform interface to pilot frameworks (PF)
- support for application level scheduling on those PFs
- support for application level scheduling across those PFs

Use Case (i)

- NGS data on storage archive on XSEDE/lonestar

Use Case (i)

- NGS data on storage archive on XSEDE/lonestar
- compute pilots on XSEDE

Use Case (i)

- NGS data on storage archive on XSEDE/lonestar
- compute pilots on XSEDE
- scheduling of data allocations & transfer

Use Case (i)

- NGS data on storage archive on XSEDE/lonestar
- compute pilots on XSEDE
- scheduling of data allocations & transfer
- scheduling compute node allocation & execution

Use Case (ii)

- NGS data on storage archive on XSEDE/lonestar

Use Case (ii)

- NGS data on storage archive on XSEDE/lonestar
- compute pilots on XSEDE and EGI

Use Case (ii)

- NGS data on storage archive on XSEDE/lonestar
- compute pilots on XSEDE and EGI
- scheduling of data allocations & transfer

Use Case (ii)

- NGS data on storage archive on XSEDE/lonestar
- compute pilots on XSEDE and EGI
- scheduling of data allocations & transfer
- scheduling compute node allocation & execution

Use Case (iii)

- develop new scheduling algorithm / data placement policy

Use Case (iii)

- develop new scheduling algorithm / data placement policy
- run workload from UC(i) with backend scheduler (i.e. Bigjob)

Use Case (iii)

- develop new scheduling algorithm / data placement policy
- run workload from UC(i) with backend scheduler (i.e. Bigjob)
- run same workload with application level scheduler

Use Case (iii)

- develop new scheduling algorithm / data placement policy
- run workload from UC(i) with backend scheduler (i.e. Bigjob)
- run same workload with application level scheduler
- compare, analyze

Use Case (iii)

- develop new scheduling algorithm / data placement policy
- run workload from UC(i) with backend scheduler (i.e. Bigjob)
- run same workload with application level scheduler
- compare, analyze
- ...

Use Case (iii)

- develop new scheduling algorithm / data placement policy
- run workload from UC(i) with backend scheduler (i.e. Bigjob)
- run same workload with application level scheduler
- compare, analyze
- ...
- publish ;-)

TROY Placement and Scope

- is-a application framework
  - application defines resource requirments
  - application defines data and compute workload

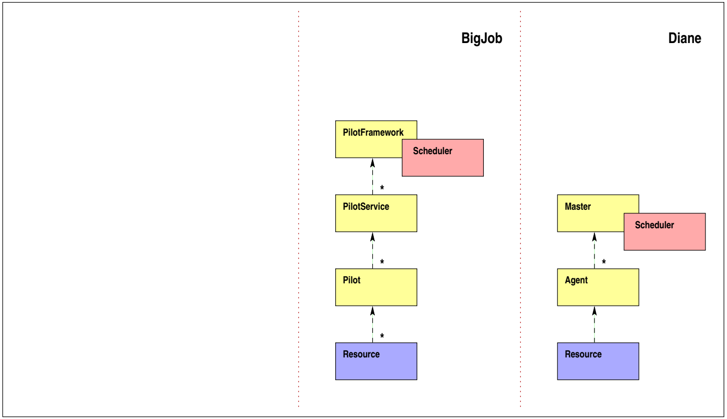
TROY Placement and Scope

- is-a application framework
  - application defines resource requirments
  - application defines data and compute workload
- is-a scheduling framework
  - hosts scheduling algorithms
  - interfaces to external schedulers
  - enacts scheduling decisions

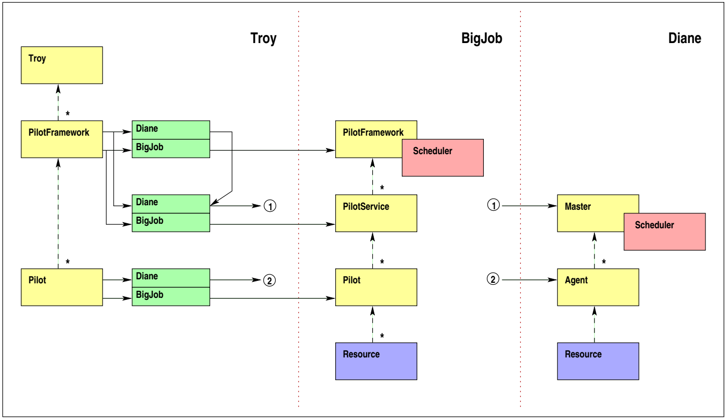
TROY Placement and Scope

- is-a application framework
  - application defines resource requirments
  - application defines data and compute workload
- is-a scheduling framework
  - hosts scheduling algorithms
  - interfaces to external schedulers
  - enacts scheduling decisions
- interface to pilot job frameworks
  - assumes P*, and possibly Pilot API

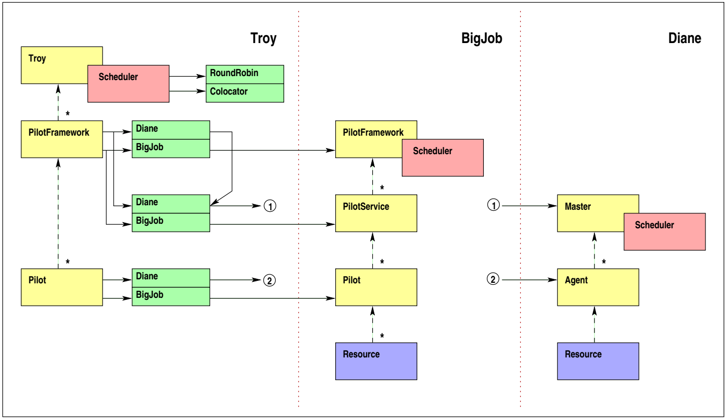
Architecture



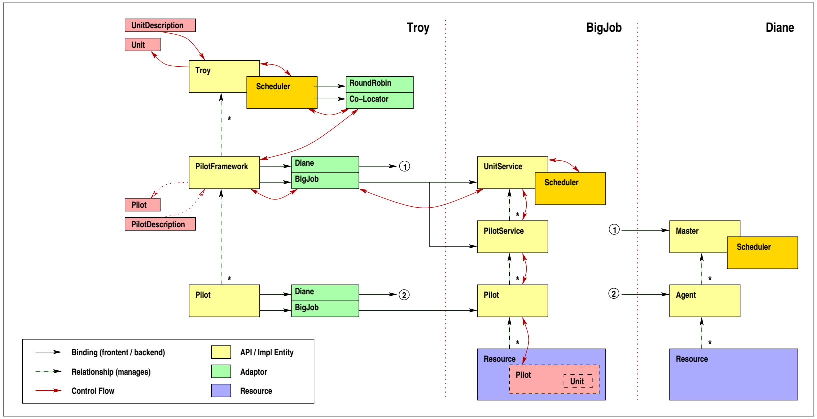
Architecture



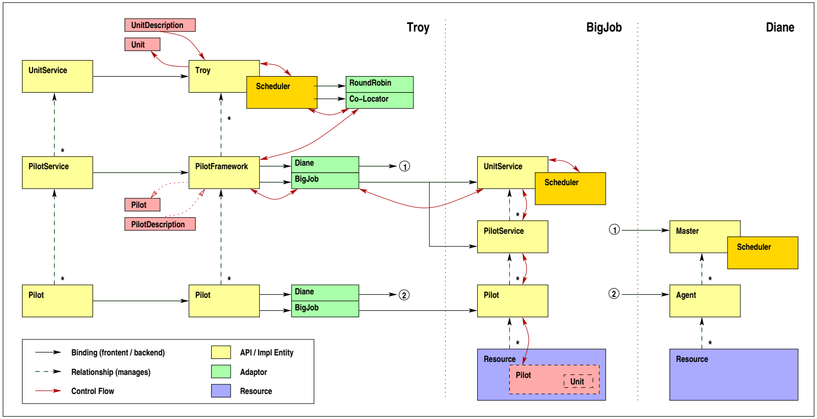
Architecture



Architecture



Architecture



Troy API Classes

Troy classes interfacing to backend pilot systems:

- `troy.Scheduler`
- `troy.PilotFramework`
  interfaces to the `XXXUnitService` and `XXXPilotService` classes of Pilot API
- `troy.ComputePilot`
- `troy.ComputeUnit`
- `troy.DataPilot`
- `troy.DataUnit`

Troy API

API Example

```
t   = troy.Troy ()
t.PilotFramework ('bigjob//lonestar')
t.PilotFramework ('bigjob//kraken')

s   = troy.Scheduler ('Random')
t.add_scheduler (s)

cpd  = troy.ComputePilotDescription ()
pf1.submit_pilot (cpd)
pf2.submit_pilot (cpd)
```

Troy API

API Example Cont.

```
cud  = troy.ComputeUnitDescription ()

cud['executable'] = '/bin/sh'
cud['arguments']  = ['-c', 'touch /tmp/hello_troy_pj && sleep 10']

cu  = t.submit_unit (cud)
```

Troy API

API Example Cont.

```
s_  = cu.state

while s_ != troy.State.Done and \
      s_ != troy.State.Failed   :

    print "cu : %s"  %  (str(s_))
    time.sleep (1)
    s_ = cu.state

print "cu : %s"  %  (str(s_))

cp1.cancel ()
cp2.cancel ()

pf.cancel ()
```

Troy API

Scheduler

```
def my_scheduler (troy, ud) :

    pf_ids = troy.list_pilot_frameworks ()
    pilots = []

    for pf_id in pf_ids :
        pf    = troy.PilotFramework (pf_id)
        p_ids = pf.list_pilots ()

        for p_id in p_ids :
            if _ud_is_compute (ud) :
                pilots.append (troy.ComputePilot (p_id))
            else :
                # ignore non-compute ud's
                pass
```

Troy API

Scheduler Cont.

```
idx = random.randint (0, len  (pilots) - 1)
p   = pilots[idx]

return p.submit_unit (ud)
```

Troy API

API Example + Scheduler

```
t    = troy.Troy ()
pf   = troy.PilotFramework ('bigjob//')
t.add_pilot_framework (pf)

s    = troy.Scheduler ('Random')
t.add_scheduler (s)

cpd  = troy.ComputePilotDescription ()
cp1  = pf.submit_pilot (cpd)
cp2  = pf.submit_pilot (cpd)
```

Troy API

API Example + Scheduler

```
t    = troy.Troy ()
pf   = troy.PilotFramework ('bigjob//')
t.add_pilot_framework (pf)


t.add_scheduler (my_scheduler)

cpd  = troy.ComputePilotDescription ()
cp1  = pf.submit_pilot (cpd)
cp2  = pf.submit_pilot (cpd)
```